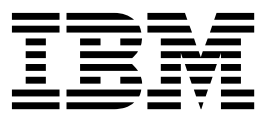


Netcool/Impact
Version 6.1.1.4

Solutions Guide



Netcool/Impact
Version 6.1.1.4

Solutions Guide



Note

Before using this information and the product it supports, read the information in “Notices”.

Edition notice

This edition applies to version 6.1.1.4 of IBM Tivoli Netcool/Impact and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2006, 2016.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Solutions Guide vii

Intended audience	vii
Publications	vii
Netcool/Impact library	vii
Accessing terminology online	vii
Accessing publications online	viii
Ordering publications	viii
Accessibility	viii
Tivoli technical training	viii
Support for problem solving	ix
Obtaining fixes	ix
Receiving weekly support updates	ix
Contacting IBM Software Support	x
Conventions used in this publication	xii
Typeface conventions	xii
Operating system-dependent variables and paths	xii

Chapter 1. Solutions overview 1

Solution components	1
Data models	1
Working with services	1
Policies	1
Solution types	2
Event enrichment solution	2
X events in Y time solution	2
Event notification solution	2
Event gateway solution	3
Setting up a solution	3
Creating a data model	3
Setting up services	3
Creating policies	3
Running a solution	4

Chapter 2. Working with data models . . . 5

Data model components	5
Data sources	5
Configuring data types	5
Working with data items	6
Working with links	6
Setting up a data model	6
Data model architecture	7
Data model examples	7
Enterprise service model	8
Web hosting model	9
Working with data sources	10
Data sources overview	10
Data source categories	10
Data source architecture	12
Setting up data sources	13
Working with data types	13
Data types overview	13
Data type categories	14
Data type fields	16
Data type keys	18
Setting up data types	18

Data type caching	32
Working with links	33
Links overview	33
Link categories	34
Static links	34
Dynamic links	34
Setting up static links	35
Setting up dynamic links	36
Working with event sources	36
Event sources overview	36
ObjectServer event sources	37
Non-ObjectServer event sources	37
Event source architecture	37
Setting up ObjectServer event sources	38

Chapter 3. Working with services . . . 41

Services overview	41
Predefined services	41
User-defined services	42
Database event reader service	42
OMNIBus event reader service	42
OMINbus event reader architecture	43
OMNIBus event reader process	43
OMNIBus event reader configuration	44
Database event listener service	50
Setting up the database server	50
Database event listener service configuration window	53
Sending database events	54
Writing database event policies	67
OMNIBus event listener service	69
Setting up the OMNIBus event listener service	69
How to check the OMNIBus event listener service logs	70
Creating Triggers	70
Using the ReturnEvent function	71
Subscribing to individual channels	72
Controlling which events get sent over from OMNIBus to Netcool/Impact using Spid	72
Working with other services	73
Policy activator service	73
Policy logger service	74
Hibernate policy activator service	76
Command execution manager service	76
Command line manager service	77

Chapter 4. Working with policies . . . 79

Understanding policy language components	79
Policy log	79
Policy context	79
Policy scope	79
Printing to the policy log	80
User-defined variables	80
Array	81
Context	83

If statements	84
While statements	85
User-defined functions.	87
Scheduling policies	89
Running policies using the policy activator.	89
Running policies using schedules	90

Chapter 5. Handling events 95

Events overview.	95
Event containers.	95
EventContainer variable	95
Event field variables	95
Event state variables	96
User-defined event container variables	96
Accessing event fields	96
Using the dot notation.	96
Using the @ notation	96
Updating event fields	96
Adding journal entries to events	97
Assigning the JournalEntry variable	97
Sending new events	98
Deleting events	99
Examples of deleting an incoming event from the event source	99

Chapter 6. Handling data 101

Working with data items	101
Field variables	101
DataItem and DataItems variables	101
Retrieving data by filter	101
Working with filters	101
Retrieving data by filter in a policy	105
Retrieving data by key	107
Keys	107
Key expressions	107
Retrieving data by key in a policy	108
Retrieving data by link	109
Links overview.	109
Retrieving data by link in a policy	109
Adding data.	110
Example of adding a data item to a data type	111
Updating data	111
Example of updating single data items	112
Example of updating multiple data items	112
Deleting data	113
Example of deleting single data items	113
Example of deleting data items by filter	114
Example of deleting data items by item.	114
Calling database functions	114

Chapter 7. Handling hibernations . . . 117

Hibernations overview	117
Hibernating a policy	117
Examples of hibernating a policy	117
Retrieving hibernations	118
Retrieving hibernations by action key search	118
Retrieving hibernations by filter	119
Waking a hibernation.	119
Retrieving the hibernation	119
Calling ActivateHibernation	120

Example	120
Removing hibernations	120

Chapter 8. Sending email 121

Sending email overview	121
Sending an email	121

Chapter 9. Setting up instant messaging 123

Netcool/Impact IM	123
Netcool/Impact IM components	123
Netcool/Impact IM process.	123
Message listening	123
Message sending	124
Setting up Netcool/Impact IM.	124
Writing instant messaging policies	124
Handling incoming messages	124
Sending messages	124
Example	124

Chapter 10. Executing external commands 127

External command execution overview	127
Managing the JRExec server	127
Overview of the JRExec server	127
Starting the JRExec server	127
Stopping the JRExec server.	128
The JRExec server configuration properties	128
JRExec server logging	129
Running commands using the JRExec server	129
Using CommandResponse	129

Chapter 11. Handling strings and arrays 131

Handling strings	131
Concatenating strings	131
Finding the length of a string	131
Splitting a string into substrings	132
Extracting a substring from another string.	132
Replacing a substring in a string	133
Stripping a substring from a string	133
Trimming white space from a string.	133
Changing the case of a string	134
Encrypting and decrypting strings	134
Handling arrays	134
Finding the length of an array.	134
Finding the distinct values in an array	135

Chapter 12. Event enrichment tutorial 137

Tutorial overview	137
Understanding the Netcool/Impact installation	137
Understanding the business data	138
Analyzing the workflow.	138
Creating the project	138
Setting up the data model	139
Creating the event source	139
Creating the data sources	140
Creating the data types	140
Creating a dynamic link.	141

Reviewing the data model	142
Setting up services	142
Creating the event reader	143
Reviewing the services	143
Writing the policy	143
Looking up device information	143
Looking up business departments	144
Increasing the alert severity	145
Reviewing the policy	146
Running the solution	146

Chapter 13. Configuring the Impact policy PasstoTBSM. 149

Overview.	149
Configuration	149
Exporting and Importing the ForImpactMigration project.	150
Creating a policy	150
Creating a policy activator service	152
Create a new template and rule to collect weather data	153
Create the CityHumidity rule for the CityWeather template	154
Create a city service	155
Customizing a Service Tree portlet	156
Adding a custom Services portlet to a freeform page	157

Chapter 14. Working with the Netcool/Impact UI data provider 159

Getting started with the UI data provider	159
UI data provider components	160
Configuring user authentication	161
Data types and the UI data provider	162
Integrating chart widgets and the UI data provider	163
Names reserved for the UI data provider	163
General steps for integrating the UI data provider and the console	164
Accessing data from Netcool/Impact policies.	166
Configuring user parameters	166
Accessing Netcool/Impact object variables in a policy	168
Accessing data types output by the GetByFilter function	168
Accessing data types output by the DirectSQL function	169
Accessing an array of Impact objects with the UI data provider	171
UI data provider and the IBM Dashboard Application Services Hub	173
Filtering data in the console	173
Integrating the tree widget with an Impact object or an array of Impact objects	174
Integrating data from a policy with the topology widget	177
Displaying status and percentage in a widget	179
Visualizing data from the UI data provider in the console	181
Example scenario overview.	182

Visualizing data from the Netcool/Impact self service dashboards	200
Installing the Netcool/Impact Self Service Dashboard widgets	200
Editing an Input Form widget.	202
Editing a Button widget.	203
Reference topics	205
Large data model support for the UI data provider	205
UI data provider customization	207
Accessing the Netcool/Impact UI data provider	209
Accessing data sources from a UI data provider	209
Accessing data sets from a UI data provider	210
Known issues with JavaScript and the UI data provider	211
Running policies and accessing output parameters	211
UI data provider URLs	212

Chapter 15. Working with OSLC for Netcool/Impact. 213

Introducing OSLC.	214
OSLC resources and identifiers	215
OSLC roles	215
Working with data types and OSLC.	216
Accessing Netcool/Impact data types as OSLC resources	216
Retrieving OSLC resources that represent Netcool/Impact data items	217
Displaying results for unique key identifier	219
OSLC resource shapes for data types	219
Configuring custom URIs for data types and user output parameters	221
Working with the OSLC service provider	223
Creating OSLC service providers in Netcool/Impact	224
Registering OSLC service providers with Netcool/Impact	226
Registering OSLC resources	227
Working with Netcool/Impact policies and OSLC	236
Accessing output user parameters as OSLC resources	236
Configuring custom URIs for policy results and variables	246
Passing argument values to a policy.	248
Configuring hover previews for OSLC resources	248
Hover preview properties for OSLC resources	250
Example scenario: using OSLC with Netcool/Impact policies.	252
OSLC reference topics	253
OSLC urls	253
OSLC pagination	254
OSLC security	255
Support for OSLC query syntax	255
RDF functions	262

Chapter 16. Service Level Objectives (SLO) Reporting 277

SLO terminology overview	278
SLO reporting prerequisites	279

Installing and enabling SLO report package	279
Defining service definition properties	281
Service definition properties file	282
Configuring the time zone	285
Configuring business calendars	287
Creating common properties in business calendars	287
Business calendar properties file	289
Retrieving SLA metric data	290
SLO reporting policies	291
SLO reporting policy functions	291
Using the getDataFromTBSMAvailability sample policy	294
Configuring getDataFromTBSMAvailability	295
Reports	295
Example SLO reporting configuration	296
Properties files examples	298
Operational hours service level example	298
Single SLA example	299
Time zone example	299
Simple service definition example	300
Multiple identities in a service definition example	301
Common US calendar properties	301
US Calendar example	302
Common calendar properties file example	302
Canada calendar example	303
SLA Utility properties	303
SLO Utility Functions	304
Maintaining the reporting data in the SLORPRT database	304
Removing service, SLA, and calendar definitions	304
Exporting service and calendar definitions	305
Removing specific outage data	306
Restoring outage data	307
Setting SLO configuration values	308

Chapter 17. Configuring Maintenance Window Management 311

Activating MWM in a Netcool/Impact cluster	311
Configure the MWM_Properties policy	311
Configuring MWMActivator service properties	312
Logging on to Maintenance Window Management	313
About MWM maintenance windows	313

Chapter 18. Configuring Event Isolation and Correlation 317

Overview	317
Installing Netcool/Impact and the DB2 database	317
Installing the Discovery Library Toolkit	318
Event Isolation and Correlation policies	319
Event Isolation and Correlation operator views	319
Configuring Event Isolation and Correlation data sources	319
Configuring Event Isolation and Correlation data types	320
Creating, editing, and deleting event rules	321
Creating an event rule	321
Configuring WebGUI to add a new launch point	322
Launching the Event Isolation and Correlation analysis page	322
Viewing the Event Analysis	323

Appendix. Accessibility 325

Glossary 327

A	327
B	327
C	327
D	327
E	328
F	329
G	329
H	329
I	329
J	330
K	330
L	330
M	331
N	331
O	331
P	331
S	331
U	333
V	333
W	333
X	333

Index 335

Solutions Guide

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*, SC14755900
Provides information about installing, running and monitoring the product.
- *User Interface Guide*, SC27485100
Provides instructions for using the Graphical User Interface (GUI).
- *Policy Reference Guide*, SC14756100
Contains complete description and reference information for the Impact Policy Language (IPL).
- *DSA Reference Guide*, SC27485200
Provides information about data source adaptors (DSAs).
- *Operator View Guide*, SC27485300
Provides information about creating operator views.
- *Solutions Guide*, SC14756000
Provides end-to-end information about using features of Netcool/Impact.
- *Integrations Guide*, SC27485400
Contains instructions for integrating Netcool/Impact with other IBM® software and other vendor software.
- *Troubleshooting Guide*, GC27485500
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 Quick Start Guide. 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.doc6.1.1/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/tivoli/documentation>. You can 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.ibm.link.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.ibm.link.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 325.

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 toolbar. 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 world 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. Solutions overview

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

This section contains information about creating data sources, data types, services and policies to set up event management. It also contains end-to-end information about the following features in Netcool/Impact:

- Event enrichment
- Netcool/Impact using the PassToTBSM function to show data in TBSM gathered from an Netcool/Impact policy.
- Netcool/Impact as a UI data provider.
- Visualizing data from the UI data provider in the IBM Dashboard Application Services Hub in Jazz for Service Management.
- Visualizing data from the Netcool/Impact self service dashboards in the IBM Dashboard Application Services Hub in Jazz for Service Management.
- Working with OSLC and Netcool/Impact.
- Setting up Service Level Objectives (SLO) Reporting.
- Configuring Event Isolation and Correlation.
- Configuring Maintenance Window Management.
- SLO reports.

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 (IPL).

Solution types

You can use Netcool/Impact 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 12, “Event enrichment tutorial,” on page 137.

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

For information about removing disassociated files that result from XinY policy, see the *Troubleshooting Guide*.

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 email require an email reader and the email sender service.

The first category of services is built in services like the event processor and the command-line service manager. You can have only a single instance of this type of service in Netcool/Impact. 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 4, “Working with policies,” on page 79.

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 2. 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 Netcool/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.

Configuring 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.

Working with 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.

Working with 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.

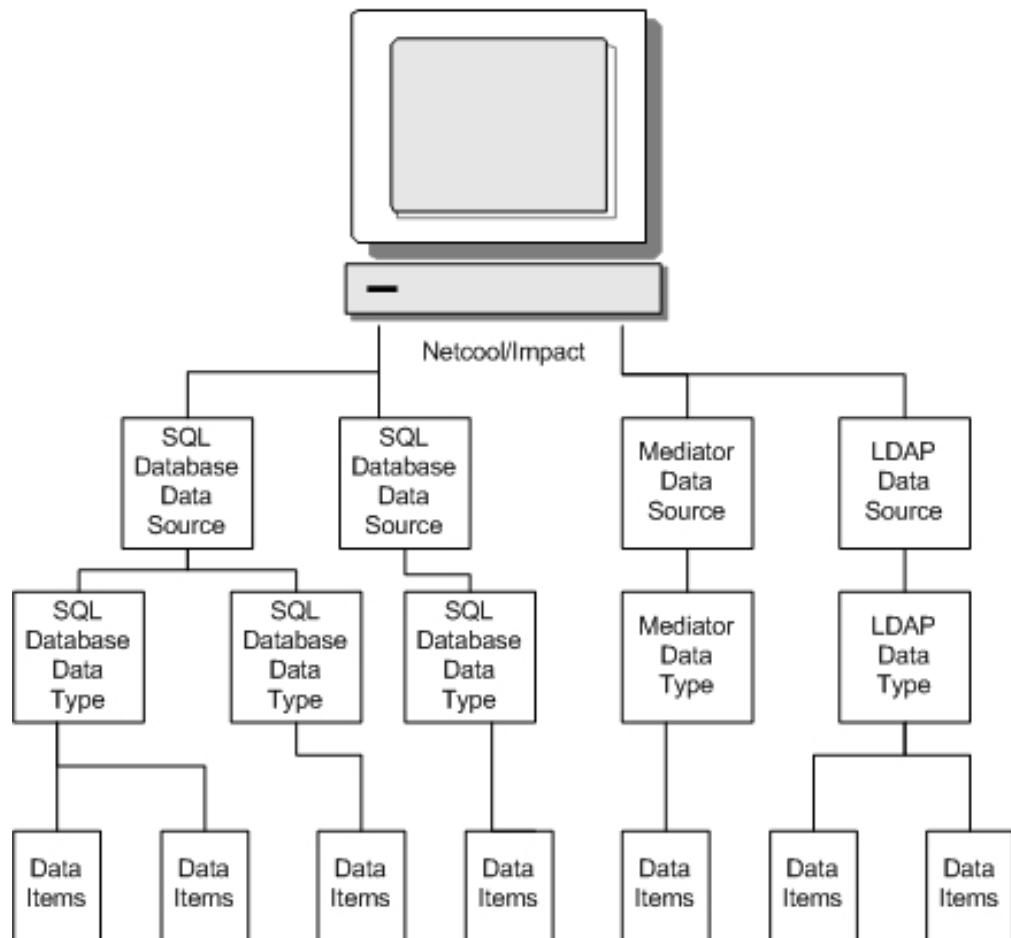


Figure 1. Data Model Architecture

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

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 Lightweight Directory Access Protocol (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 Java™ Message Service (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.

A wide variety of 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 Lightweight Directory Access Protocol (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 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 Java Message Service (JMS) data source abstracts the information that is required to connect to a JMS Implementation.

This data source is used by the JMSMessageListener service, 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.

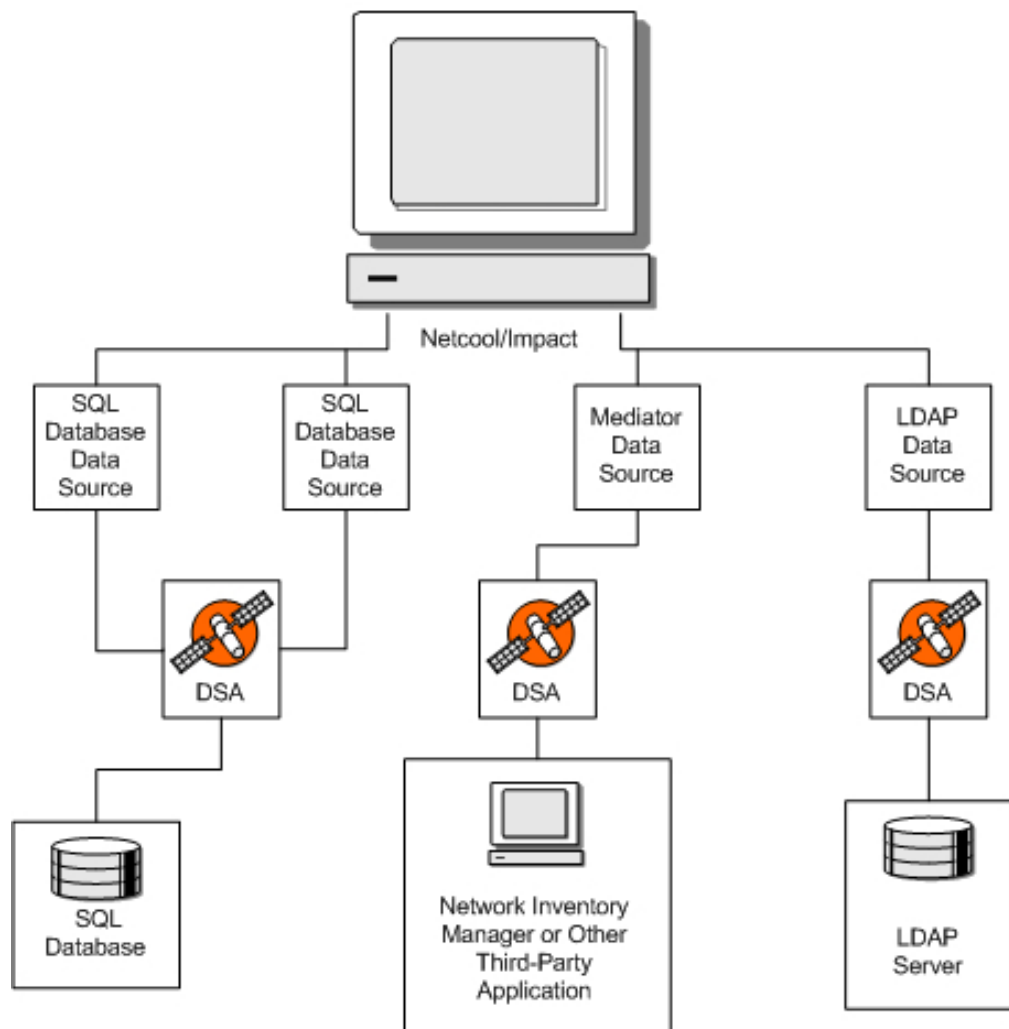


Figure 2. Data Source Architecture

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.

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 you can use the Tivoli Integrated Portal GUI to freely edit this field, it must be identical to how it displays in the data source. If these fields are not identical, an error occurs when the data type is accessed.

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.
TIMESTAMP	Represents a timestamp in the following format, YYYY-MM-DD HH:MM:SS . Restriction: The Microsoft SQL server table treats the TIMESTAMP field as a non-date time field. The JDBC driver returns the TIMESTAMP field as a row version binary data type, which is discovered as STRING in the Microsoft SQL server data type. To resolve this issue, in the Microsoft SQL server table, use DATEITEM to display the property time format instead of TIMESTAMP.
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 shows in the GUI as a string of asterisks, rather than the actual password text.

Display name

You can use the display name attribute 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 functions 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

You can use the description attribute to specify a short description for the field.

This description is only visible when you use the GUI to edit the data type. Like the display name, it does not otherwise affect the functions 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 “Working with links” on page 6.

Internal data type configuration window:

Use this information to configure an internal data type.

*Table 2. General settings on the **New Internal Data Type Editor Custom Fields** tab*

Editor element	Description
Data Type Name	<p>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.</p> <p>If you receive an error message when you save 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 must change it.</p>
State: Persistent	<p>Leave the box checked as Persistent (permanent) to permanently store the data items that are created for this data type. When the server is restarted, the data is restored. 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.</p> <p>Persistent data types are always written to file. Therefore, making internal data types temporary is faster.</p>
New Field	Click to add a field to the table.

*Table 2. General settings on the **New Internal Data Type Editor Custom Fields** tab (continued)*

Editor element	Description
Access the data through UI data provider: Enabled	To ensure that the UI data provider can access the data in this data type, select the Access the data through UI data provider: Enabled check box. When you enable the check box, the data type sends data to the UI data provider. When the data model refreshes, the data type is available as a data provider source. The default refresh rate is 5 minutes. For more information about UI data providers, see the Solutions Guide.

*Table 3. Additional settings on the **New Internal Data Type Editor Custom Fields** tab*

Editor element	Description
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, 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.

*Table 4. UI data provider settings on the **Internal Data Type Editor Custom Fields** tab*

Editor element	Description
Define Custom Types and Values (JavaScript)	To show percentages and status in a widget, you must create a script in JavaScript format. The script uses the following syntax. <pre>ImpactUICustomValues.put ("FieldName,Type",VariableName);</pre> Add the script to the Define Custom Types and Values (JavaScript) area.
Preview Script Sample Result	Click the Preview Script Sample Result button to preview the results and check the syntax of the script. The preview shows a sample of 10 rows of data in the table.

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 5. 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	<p>In this tab you can create links to other data types, both external and internal, to establish connections between information.</p> <p>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.</p> <p>For more information about dynamic links tab, see “Working with links” on page 6.</p>
Cache Settings	<p>In this tab, you can set up caching parameters to regulate the flow of data between Netcool/Impact and the external data source.</p> <p>Use the guidelines in “SQL data type configuration window - Cache settings tab” on page 27, plus the parameters for the performance report for the data type to configure data and query caching.</p>

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 6. General settings for the Table Descriptions tab of the SQL data type configuration window

Editor element	Description
Data Type Name	<p>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.</p> <p>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 tried to save, you need to change it.</p>
Data Source: Name	<p>This field is automatically populated, based on the data source you selected in the data sources tab. If you have other SQL data sources that are configured to use with Netcool/Impact, you can change the name to any of the SQL data sources in the list, if necessary.</p> <p>If you enter a new name, a message window prompts you to confirm your change.</p> <p>Click OK to confirm the change. If you change your mind about selecting a different data source, click Cancel.</p>
State: Enabled	<p>Leave the State check box checked to activate the data type so that it is available for use in policies.</p>
Access the data through UI data provider: Enabled	<p>To ensure that the UI data provider can access the data in this data type, select the Access the data through UI data provider: Enabled check box. When you enable the check box the data type sends data to the UI data provider. When the data model refreshes, the data type is available as a data provider source. The default refresh rate is 5 minutes. For more information about UI data providers, see the Solutions Guide.</p>

Table 7. Table description settings for the Table Descriptions tab of the SQL data type configuration window

Window element	Description
Base Table	<p>Specify the underlying database and table where the data in the data type is stored.</p> <p>The names of all the databases and tables are automatically retrieved from the data source so that you can choose them from a list.</p> <p>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.</p>
Refresh	<p>Click Refresh to populate the table.</p> <p>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.</p>
Add Deleted Fields	<p>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.</p>
New Field	<p>Use this option if you need to add a 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.</p> <p>Make sure that the field name you add has the same name as the field name in the data source.</p> <p>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.</p> <p>For more information, see “SQL data type configuration window - adding and editing fields in the table” on page 25.</p>

Table 7. Table description settings for the Table Descriptions tab of the SQL data type configuration window (continued)

Window element	Description
Key field	<p>Key fields are used when you retrieve data from the data type in a policy that uses the GetByKey function. They are also used when you define a GetByKey dynamic link.</p> <p>Important: You must define at least one key field for the data type, even if you do not plan to use the GetByKey function in your policy. If you do not, Netcool/Impact does not function properly.</p> <p>Generally, the key fields you define correspond to key fields in the underlying database table.</p> <p>To specify a key field, click the check box in the appropriate row in the Key Field column. You can add multiple key fields.</p>
Display Name Field	<p>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.</p>
Automatically Remove Deleted Fields	<p>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.</p>

Table 8. Data filtering and ordering settings for the Table Descriptions tab of the SQL data type configuration window

Window element	Descriptions
Filter	<p>Type a restriction clause to limit the types of data items that are seen for the data type. For example, to limit the rows in a field that is called <i>City</i> to <i>New York</i>, you would enter:</p> <p>City = "New York"</p> <p>For example, to limit the rows to the <i>New York</i> or <i>Athens</i>, you would enter:</p> <p>City = "New York" OR City = "Athens"</p> <p>You can use any sql Where clause syntax.</p>
Order By	<p>Enter the names of one or more fields to use when sorting data items retrieved from the data source.</p>

Table 9. UI data provider settings on the Table Descriptions tab of the SQL data type configuration window.

Editor element	Description
Define Custom Types and Values (JavaScript)	To show percentages and status in a widget, you must create a script in JavaScript format. The script uses the following syntax. <pre>ImpactUICustomValues.put ("FieldName,Type",VariableName);</pre> Add the script to the Define Custom Types and Values (JavaScript) area.
Preview Script Sample Result	Click the Preview Script Sample Result button to preview the results and check the syntax of the script. The preview shows a sample of 10 rows of data in the table.

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 10. 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.
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 is displayed in the data source. Otherwise, Netcool/Impact reports an error when it tries to access the data type.

Table 10. External data type Editor - New field window (continued)

Window element	Description
Format	<p>For SQL database data types, Netcool/Impact automatically 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 <i>Working with data types</i> chapter in the Solutions Guide.</p> <p>Restriction: The Microsoft SQL server table treats the TIMESTAMP field as a non-date time field. The JDBC driver returns the TIMESTAMP field as a row version binary data type, which is discovered as STRING in the Microsoft SQL server data type. To resolve this issue, in the Microsoft SQL server table, use DATEITEM to display the property time format instead of TIMESTAMP.</p> <p>Select a format from the following list:</p> <ul style="list-style-type: none"> • STRING • LONG_STRING • INTEGER • PASSWORD_STRING • LONG • FLOAT • DOUBLE • DATE • TIMESTAMP • BOOLEAN • CLOB
Display Name	<p>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.</p> <p>If you do not enter a display name, Netcool/Impact uses the ID field name by default.</p>
Description	Type some text that describes the field. This description is only visible when you edit the data type in 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; sequence.NEXTVAL.

Table 10. External data type Editor - New field window (continued)

Window element	Description
Insert Statements: Exclude this field	<p>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.</p> <p>Sybase data types:</p> <p>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.</p> <p>ObjectServer data types:</p> <p>The Tally field automatically selects the Exclude this Field check box to be excluded from inserts and updates for the object server data type since this field is automatically set by Netcool®/OMNIBus to control deduplication of events.</p> <p>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.</p>
Type Checking: Strict	<p>Click to enable strict type checking on the field. Netcool/Impact checks the format of the value of the field on insertion 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 check the value on insertion 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.</p>

SQL data type configuration window - Cache settings tab:

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

Table 11. 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 Caching	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.

Table 11. External Data Type Cache Settings tab - caching types (continued)

Cache type	Description
Polling Interval	Select a polling interval for measuring performance statistics for the data type.
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 that is 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 expression:

```
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 12. 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 Info tab of the LDAP data type configuration window.”
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 “Working with links” on page 6.
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 27.

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 Info tab of the LDAP data type configuration window:

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

Table 13. General settings in the LDAP Info Tab on the LDAP Data Type editor

Editor element	Description
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.

Table 14. LDAP settings in the **LDAP Info Tab** on the **LDAP Data Type** editor

Editor element	Description
Data Source Name	<p>Type the name of the underlying data source.</p> <p>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.</p> <p>If you enter a new name, the system displays a message window that asks you to confirm your change.</p>
Search scope	<p>Select the search scope:</p> <ul style="list-style-type: none"> • OBJECT_SCOPE • ONLEVEL_SCOPE • SUBTREE_SCOPE
Base Context	Type the base context that you want to use when you search 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 <i>LDAP Filter</i> information in the <i>Policy Reference Guide</i> .

Table 15. Attribute configuration in the **LDAP Info Tab** on the **LDAP Data Type** editor

Editor element	Description
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.

Important: In order for data caching to work, the KeyFields in the data type must be unique.

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.

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 you can 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 enables 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 need to create static links.) You can create dynamic links for all types of data types (internal, external, and predefined). See “Configuring data types” on page 5 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.

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.

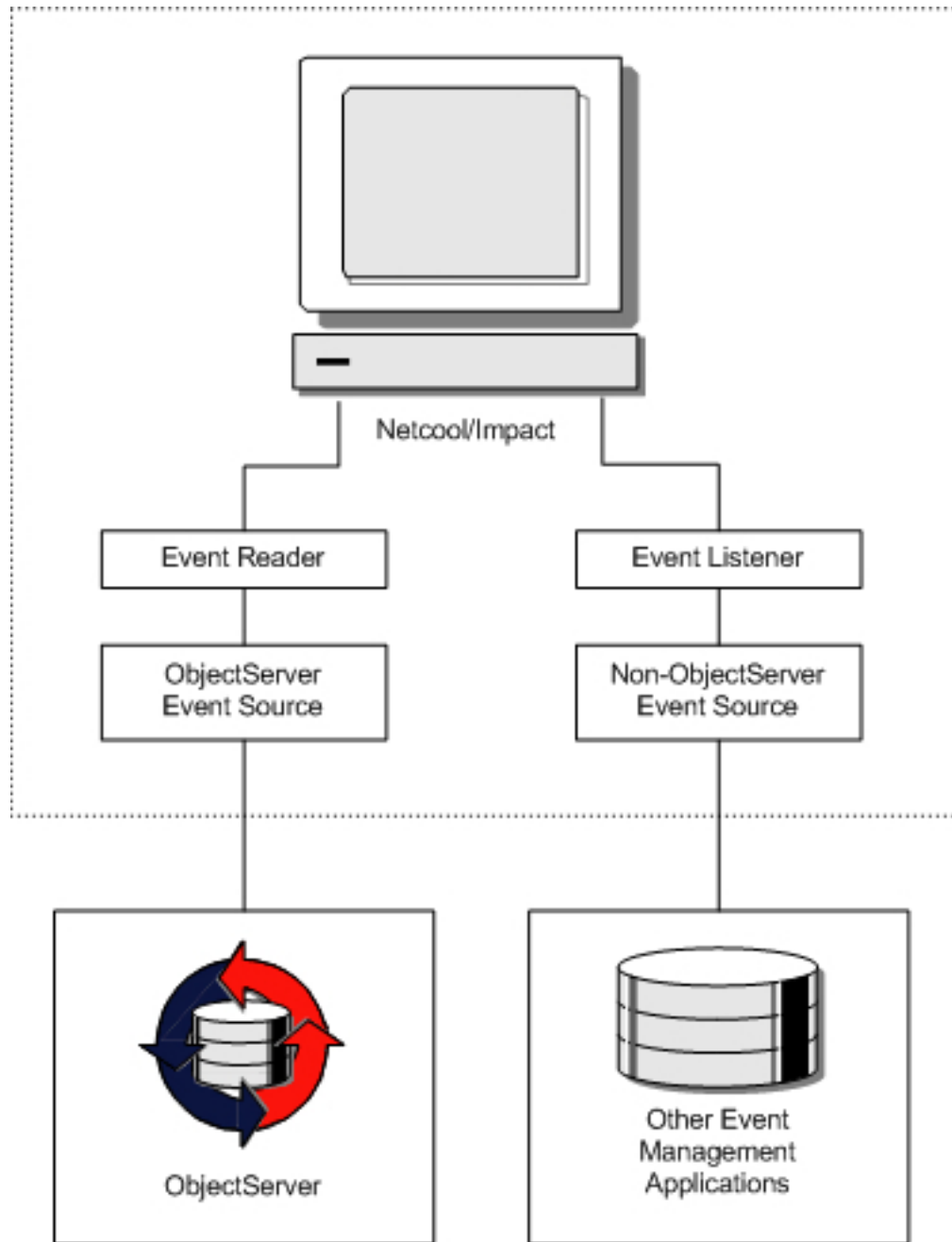


Figure 3. Event source architecture

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 3. 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

Database event reader service

The database event reader service is a service that polls supported, external SQL data sources at regular intervals to get business events in real time. The service is configured via the GUI.

You can add an additional property to the `NCI_XXX.props` file in order to match the date and time format of the timestamp field in the external database. Where NCI is the name of the impact instance and XXX is the name of the database event reader.

An example of this property is:

```
impact.XXX.formatpattern=dd-MMM-yy hh.mm.ss.SSS aaa
```

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.

OMNIBus event reader architecture

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

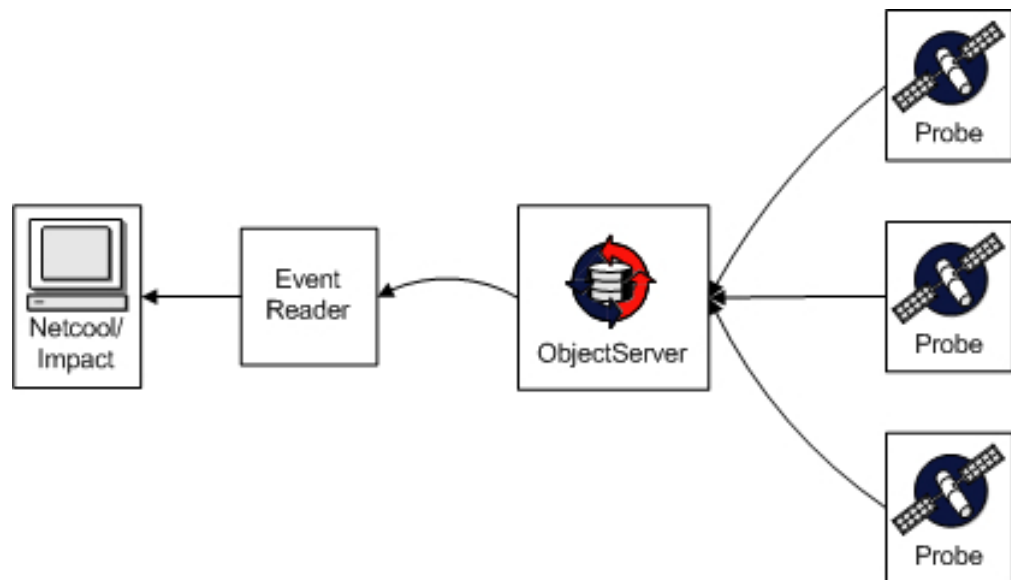


Figure 4. Event reader architecture

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” on page 44.

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 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. For more information, see “Event queuing.”

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 16. EventReader service - general settings tab

Table Element	Description
Service name	Enter 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	<p>The polling interval is the interval in milliseconds at which the event reader polls the ObjectServer for new or updated events.</p> <p>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).</p>
Restrict Fields: Fields	<p>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.</p> <p>Click the Fields button to access a list of all the fields available from the selected ObjectServer data source.</p> <p>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.</p>
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	<p>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.</p> <p>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.</p>

Table 16. EventReader service - general settings tab (continued)

Table Element	Description
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 17. Event Mapping tab

Window element	Description
Test events with all filters	<p>Select this option to test events with all filters and run any matching policies.</p> <p>If an event matches more than one filter, all policies that match the filtering criteria are triggered.</p>
Stop testing after first match	Select this option to stop testing after the first matching policy, and trigger only the first matching policy.

Table 18. Actions on the Event Mapping tab

Window element	Description
Get updated events	<p>Select to receive updated events and new events from the ObjectServer. All new events are automatically sent. See also the description of the Order By field for more information.</p> <p>If you do not select Get Updates Events, Netcool/Impact uses Serial instead. You can configure the OMNibusEventReader service to fetch only new events and to work with a ObjectServer failover/failback pair in the eventreader.props file.</p> <p>Important: Adding properties to the eventreader.props file overrides selecting or clearing the Get Updates Events check box in the UI.</p> <ul style="list-style-type: none"> • If you plan to use this approach in an ObjectServer failover scenario, see the section <i>Managing the OMNibusEventReader with an ObjectServer pair for New Events or Inserts</i> in the <i>Troubleshooting</i> section. • If you do not select Get Updated Events, Netcool/Impact uses the Serial field to query Netcool/OMNibus. Serial is an auto increment field in Netcool/OMNibus and has a maximum limit before it rolls over and resets. For information about to set up Netcool/Impact to handle Serial rollovers, see the section <i>Handling Serial rollover</i> in the <i>Troubleshooting</i> section.
Get status events	Select to receive the status events that the Self Monitoring service inserts into the ObjectServer.

Table 18. Actions on the **Event Mapping** tab (continued)

Window element	Description
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 that 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	<p>Select if you want to use event order locking and type the locking expression in the Expression field.</p> <p>Event locking allows a multi-threaded event processor to categorize incoming alerts that are based on the values of specified alert fields and processes them one at a time.</p> <p>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.</p> <p>When event locking is enabled a value of true is stored for enableorderevents, in the property file for the event reader, for example:</p> <pre>impact.omnibuseventreader. enableorderevents=true</pre> <p>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 that are running simultaneously.</p>
Expression	<p>The locking expression consists of one or more alert field names.</p> <p>To lock on a single field, specify the field name, for example:</p> <pre>Node</pre> <p>To lock more than one field, concatenate them with the + sign, for example:</p> <pre>Node+Severity</pre> <p>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.</p> <p>The expression is stored in the property file for the Event Reader as eventlockingexpression, for example:</p> <pre>impact.omnibuseventreader. eventlockingexpression= Node+Severity</pre>

Table 18. Actions on the **Event Mapping** tab (continued)

Window element	Description
New Mapping	Click to add a new filter.
Order by	<p>If you want to order incoming events that are retrieved from the ObjectServer, type the name of an alert field or a comma-separated list of fields. The event reader sorts incoming events in ascending order by the contents of this field.</p> <p>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.</p> <p>For example, to sort incoming events by the contents of the Node field, type Node.</p> <p>To sort events first by the contents of the Node field and then by the contents of the Summary field, type Node, Summary.</p> <p>You can also specify that the sort order is ascending or descending by using the ASC or DESC key words.</p> <p>For example, to sort incoming events by the contents of the Node field in ascending order, type the following Node ASC.</p> <p>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 Updates 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.</p>
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 there 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.

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. Use the configuration properties 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.”

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 52.

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 52.

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 53.

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 NC11, and NC12 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 19 shows the properties in the listener properties file:

Table 19. Database client listener properties file

Property	Description
<code>impact.cluster.name</code>	Name of the Impact Server cluster where the database event listener is running. The default value for this property is <code>NCICLUSTER</code> .
<code>impact.dblistener.debug</code>	Specifies whether to run the database client in debug mode. The default value for this property is <code>true</code> .
<code>impact.dblistener.delim</code>	Specifies the delimiter character that separates name/value pairs in the <code>VARRAY</code> sent by Java stored procedures to the database client. The default value for this property is the pipe character (<code> </code>). You cannot use the colon (<code>:</code>) 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 6.1.1, remove any preexisting Java archive (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. Use the following command to install the `nameserver.jar` file:

```
loadjava -user username/password -resolve nameserver.jar
```
 - b. Use the following command to install the `impactdblistener.jar` file:

```
loadjava -user username/password -resolve impactdblistener.jar
```
 - c. Use the following command to install the `nameserver.props` file:

- ```
loadjava -user username/password -resolve nameserver.props
```
- d. Use the following command to install the `impactdblistener.props` file:
- ```
loadjava -user username/password -resolve impactdblistener.props
```

Important: You must follow this order of installation, otherwise `loadjava` cannot 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 20. Event mapping settings for database event listener service configuration window

Window element	Description
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.
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 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 60.

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 57.

```
// 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 21 on page 68 shows the event fields that you must populate in the trigger.

Table 21. 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);
```

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 Netcool/OMNIBus through the Accelerated Event Notification feature of Netcool/OMNIBus. It receives notifications through the Insert, Delete, Update, or Control (IDUC) channel. To set up the OMNIBus event listener, you must set its configuration properties through the GUI. You can use the configuration properties to specify one or more channels for which events get processed and also one or more policies that are to be run in response to events received from Netcool/OMNIBus.

Important:

- The OMNIBus event listener service works with Netcool/OMNIBus to monitor ObjectServer events.
- If the Impact Server and OMNIBus server are 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 host name of the OMNIBus server.

For more information about Netcool/OMNIBus triggers and accelerated event notification, and the **Iduc.ListeningHostname** property in the OMNIBus server, see the *Netcool/OMNIBus Administration Guide* available from this website:

Tivoli Documentation Central

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 one or more IDUC channels to get notifications for inserts, updates, and deletes.

How to check the OMNIBus event listener service logs

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
Service Started
Attempting to connect for IDUC notifications
Established connection to the Data Source defaultobjectserver
IDUC Connection: Established:
Iduc Hostname : nc050094
Iduc Port      : 58003
Iduc Spid      : 2

```

Creating 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, see the *IBM Tivoli Netcool/OMNIBus Administration Guide* available from the following website:

<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 from Node called ImpactNode:

```

create or replace trigger ft_insert1
group trigger_group1
priority 1
after insert on alerts.status
for each row
begin
    if (new.Severity = 5 AND new.Node = 'ImpactNode')
    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 reinsertion 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, see 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 by using the ReturnEvent function:

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

Subscribing to individual channels

How to subscribe to one or more OMNIBus channels for which Netcool/Impact processes events received from Netcool/OMNIBus.

Procedure

1. In the OMNIBusEventListener configuration window, in **One or more Channels** field, add the channel from which Netcool/Impact processes events.
2. To subscribe to more than one channel, add a comma between each channel name.
3. To change the channel name or to add or remove one or more entries add the changes and restart the OMNIBusEventListener service to implement the changes.

Results

When Netcool/Impact receives a Fast Track event from a channel that matches one of the configured channels, the OMNIBusEventListener service log displays the following message:

```
Received Fast Track Message from channel: <channel_name>
```

When Netcool/Impact receives a Fast Track event that does not match any configured channels, the OMNIBusEventListener service log displays the following message:

```
Fast Track Message from channel:  
<channel name> did not match any configured channels.
```

Restriction: Filtering messages by channel is only supported for Fast Track messages that are sent by using the **iduc evt f** command. For signal messages sent by using the **iduc sndmsg** command, Netcool/Impact does not filter the messages by which channel they originated from. For information about these commands, see the *IBM Tivoli Netcool/OMNIBus Administration Guide* available from the following website:

<https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/OMNIBus>.

Controlling which events get sent over from OMNIBus to Netcool/Impact using Spid

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 displays 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* available from this website:

<https://www.ibm.com/developerworks/wikis/display/tivolidoccentral/OMNIBus>

Working with other services

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

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 22. Create New Policy Activator Service configuration window

Window element	Description
Service name	Enter a unique name to identify the service.

Table 22. Create New Policy Activator Service configuration window (continued)

Window element	Description
Activation Interval	Select how often (in seconds) the service must activate the policy. The default value is 30 on the policy activator service that comes with Netcool/Impact. When you create your own policy activator server the default value is 0.
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 23. Policy Logger Service configuration window

Window element	Description
Error-handling Policy	<p>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.</p> <p>Note: If you have a Policy Activator service and you want it to utilize a default exception handler policy, you must specify the following property in the <code><servername>_<activatorservicename>.props</code> file: <code>impact.<activatorservicename>.errorhandlername=<policy name to run></code></p>

Table 23. Policy Logger Service configuration window (continued)

Window element	Description
Highest Log Level	<p>You can specify a log level for messages that you print to the policy log from within a policy using the Log function.</p> <p>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.</p>
Log what	<p>Select what you want to appear in the log:</p> <ul style="list-style-type: none"> • 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	<p>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</p> <p>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.</p>
Service log: Write to file	Select to write log information to a file.
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	<p>Select to enable data collection for the Policy Reports.</p> <p>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.</p> <p>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:</p> <ol style="list-style-type: none"> 1. Add impact.policylogger.reportloglevel=1 to the NCHOME/impact/etc/<servername>_policylogger.props property. 2. Restart the Impact Server to implement the change.

Changing the size of the policylogger log file:

How to change the size of the policy logger log file and how to create more policy logger files.

Procedure

1. Stop your Impact server and open the `IMPACT_HOME/etc/<servername>_policylogger.props` file.
2. To change the size of the **policylogger.log** file, add the following properties to the file:
`impact.policylogger.maxlogsizebytes=[value in bytes]`
3. To change the number of **policylogger.log** files, add the following properties to the file:
`impact.policylogger.maxbackupindex=[number of backup files]`
4. Restart the server to implement the changes.
5. In a cluster setup, you must restart the secondary server so that it replicates these customized properties on startup from the primary server.

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 24. 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 30 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 and 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 25. 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 4. 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.

Understanding policy language components

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. You can 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 runtime 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 a runtime 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.

Any JavaScript reserved word or predefined JavaScript object and class names must not be used as variable names in a JavaScript 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 in the example, 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 inside 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 is 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 is 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[3]);
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.

You can use the `while` statement 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 define a function, you can call it in the same way as the built-in action and parser functions. Variables that are 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 runtime parameters. The list of runtime parameters is followed by a statement block that is 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 by using the supplied host name.

```
Function GetNodeByHostName(Hostname) {  
  
    DataType = "Node";  
    Filter = "Hostname='" + Hostname + "'";  
    CountOnly = False;  
  
    MyNodes = GetByFilter(DataType, Filter, CountOnly);  
    MyNode = MyNodes[0];  
}
```

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.

```
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.

Important:

When you write an Impact function, check that you do not call the function within the function body as this might cause a recursive loop and cause a stack overflow error.

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” on page 91.

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 91.

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 91.

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 92.

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 92.

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 5. 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 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 97. For information about using `DeleteEvent`, see “Deleting events” on page 99.

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 a runtime 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. Set the `EventReaderName` member variable:

```
MyEvent.EventReaderName = "defaulteventreader";
```

3. Populate the event fields by assigning values to its event field variables.

For example:

```
MyEvent.EventReaderName = "defaulteventreader";
MyEvent.Node = "192.168.1.1";
MyEvent.Summary = "Node down";
MyEvent.Severity = 5;
MyEvent.AlertKey = MyEvent.Node + ":" + MyEvent.Summary;
```

4. Send the event to the data source using the `ReturnEvent` function.

Call the `ReturnEvent` function, and pass the new event container as a runtime 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.EventReaderName = "defaulteventreader";
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 6. 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.

Working with 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.

Dataltem and Dataltems 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."

Working with 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 18.

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 you can 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 runtime 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 a runtime 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 7. 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 that 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 that is 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 hibernates for 60 seconds before it is 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 hibernates for 2 minutes before it is 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 in two ways:

- Action key search
- Filter

Retrieving hibernations by action key search

You can use the `GetHibernatingPolicies` function to retrieve hibernations using a lexicographical search of action key values.

About this task

`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

You can use the `GetByFilter` function to retrieve hibernations using a filter.

About this task

`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 101.

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

There are two ways that you can wake 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

The first step in waking a hibernation is to retrieve it from the Hibernation data type using `GetHibernatingPolicies` or `GetByFilter`.

About this task

This step is described in the previous section of this guide.

Calling ActivateHibernation

After you retrieve the hibernation, you can call the `ActivateHibernation` function and pass the data item as a runtime 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
// a runtime 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 a runtime parameter.

The following example shows how to remove a hibernation. In this example, the action key for the hibernation is `ActionKeyABC`.

```
RemoveHibernation("ActionKeyABC");
```

Chapter 8. Sending email

Netcool/Impact allows you to send email from within a policy.

Sending email overview

You can use the feature of sending emails from within a policy to send email 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 email, you must make sure that an SMTP server is available in your environment. The Netcool/Impact email sender service must also be running before a policy can successfully send email.

Sending an email

To send email you call the SendEmail function and pass information as runtime parameters.

About this task

To send email you call the SendEmail function and pass the following information as runtime parameters:

Procedure

- The email address of the recipient
- The subject line text for the email
- The body content of the email
- The name of the email sender

Results

The following example shows how to send an email. In this example, you send the email to the address srodriguez@example.com.

```
// Call SendEmail and pass the address, subject and message text  
// as runtime 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 9. Setting up 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 you can use 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. You can use this feature to send an instant message 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 two phases, message listening and 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

You use instant messages in a Netcool/Impact policy to send messages and handle incoming messages

Handling incoming messages

When the Jabber reader receives an incoming message, it starts the policy that is specified in the Jabber reader service configuration and passes the contents of the message to the policy using the `EventContainer` variable.

About this task

The policy can then handle the incoming message in the same way it handles information that is 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

You send instant messages from within a policy using the `SendInstantMessage` function.

About this task

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 10. 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 use the JRExec server or the command and response feature to run external commands.

The JRExec server is a runnable component of Netcool/Impact that you can use to run external commands on the system where the Netcool/Impact server is located. Command and response is a more advanced feature that you can use to run interactive and non-interactive programs on both local and remote systems.

You can run 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.

Managing the 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_jreexec` process, and a JAVA process that was started by the `nci_jreexec` process.

1. Obtain their IDs using these commands:

- `ps -eaf | grep nci_jreexec`

This command returns the PID of the `nci_jreexec` 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 `<ServerName>_server.props` file, where `<ServerName>` 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.

1. Create a properties file called `jrexecserver-log4j.properties` in the `$NCHOME/impact/etc` directory.

2. Define the following properties in the properties file:

```
log4j.rootCategory=INFO
log4j.appender.JRExec=org.apache.log4j.RollingFileAppender
log4j.appender.JRExec.threshold=DEBUG
log4j.appender.JRExec.layout=org.apache.log4j.PatternLayout
log4j.appender.JRExec.layout.ConversionPattern=%d{DATE} %-5p [%c{1}] %m%n
log4j.appender.JRExec.append=true
log4j.appender.JRExec.file=<$NCHOME>/impact/log/nci_jrexec.log
log4j.appender.JRExec.bufferedIO=false
log4j.appender.JRExec.maxBackupIndex=3
log4j.appender.JRExec.maxFileSize=10MB
```

Ensure that you use the full path for the `<$NCHOME>` value.

3. You also must set `DEBUG` as the default priority for all micromuse and IBM loggers in the same file:

```
log4j.category.com.micromuse=DEBUG,JRExec
log4j.additivity.com.ibm.tivoli=false
log4j.additivity.com.micromuse=false
```

4. Create a log file that is called `nci_jrexec.log` in the `$NCHOME/impact/logs` 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 runtime 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 11. Handling strings and arrays

Read the following information about handling strings and arrays in a policy.

Handling strings

You can use the Netcool/Impact policy 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

To concatenate strings, you use the addition operator (+).

About this task

You can concatenate two strings or multiple strings at the same time. 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

You can use the Length function to find the length of a string.

About this task

The Length function returns the number of characters in 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

You can use the `Split` function to split a string into substrings.

About this task

The `Split` function takes a string and a set of delimiter characters as runtime 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

You can use the word position or regular expression matching to extract a substring from another string.

Extracting a substring using the word position

To use the word position to extract a substring, 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 `REExtract` function. The `REExtract` function takes a string and a regular expressions pattern as runtime parameters. It returns the first matching substring that 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 runtime parameters. It returns an array that contains all the matching substrings.

Replacing a substring in a string

You can use the `Replace` function to replace a substring in a string.

About this task

The `Replace` function takes the string, the substring to replace and its replacement as runtime parameters. The function returns the string after it creates the replacement.

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

You can use the `Strip` function to strip a substring from a string.

About this task

The `Strip` function takes the string and the substring you want to strip as runtime parameters. It returns the string after the substring is 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

You can use the `Trim` function to trim leading and trailing white space from a string.

About this task

The `Trim` function takes the string as a runtime 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

You can use the `ToLower` function to change the case of a string. You can also use the `ToUpper` function to change the case of a string to all uppercase.

Example

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.

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

The policy language provides a feature that you can use to encrypt and decrypt strings.

About this task

This feature is useful if you want to handle password data within a Netcool/Impact policy.

You can use the `Encrypt` function to encrypt a string. This function takes the string as a runtime 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 a runtime parameter and returns the plaintext version.

The following example shows how to decrypt a string.

```
MyString = Decrypt("AB953E4925B39218F390AD2E9242E81A");
```

Handling arrays

You can use the Netcool/Impact policy language to find the length of an array and to find distinct values in an array.

Finding the length of an array

You can use the `Length` function to find the number of elements in an array.

About this task

The `Length` function takes the array as a runtime 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

You can use the `Distinct` function to find the distinct values in an array.

About this task

The `Distinct` function takes the array as a runtime 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 12. 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 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 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 is 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 the Node, Summary, Severity, AlertKey, and Identifier fields.

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 that 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 find 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 that are installed in the environment, they are displayed 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 email.

For the sake of this tutorial, we assume that, among other tasks, the operators perform the following actions for each high severity alert. 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 is 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

After you finish analyzing the workflow, the next step is to create a project in the Tivoli Integrated Portal GUI.

About this task

You can use this project to store the data model, services, and policies that are used in this solution. The name of this project is NCI_TUT_01.

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 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 create 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 that are required by the event enrichment solution. It also consists of a dynamic link that is used to define the relationship between the data types.

You use the Tivoli Integrated Portal GUI to perform all the tasks in this step.

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

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.

About this task

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

The next task in setting up the data model is to create the data sources.

About this task

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 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

The next task in setting up the data model is to create the data types.

About this task

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 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 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. Select `ORACLE_01` from the **Data Source Name** drop down menu.
6. Ensure that the **Enabled** check box is selected. It is selected by default.
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

The next step is to create a dynamic link between the `Device` and `Department` data types.

About this task

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 are located 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. After you create the dynamic link, you can use the `GetByLinks` function to traverse it within a policy.

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

After you create 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.

About this task

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

The OMNIBus event reader for this solution must check the NCOMS ObjectServer every 3 seconds and retrieve any new events.

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 **OMNIBusEvent Reader** 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

After you create the event reader, you can use the Tivoli Integrated Portal GUI to verify that you completed all the tasks correctly.

About this task

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 displays in the Service Status panel.

Writing the policy

After you 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.

You can use the **EnrichEvent** policy to complete the following tasks:

- Look up information about the device that is causing the alert.
- Look up the business departments that are served by the device.
- If one of the business departments 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

The first task that you want the policy to perform is to look up device information that is related to the alert in the network inventory database.

About this task

Specifically, you want the policy to retrieve technical specifications for the device that is causing the alert, and information about the facility and the rack number where the device is located.

To do this, the policy must 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.

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 is able to return more than one data item 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

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.

About this task

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 must take the data item that it previously retrieved from the Device data type and traverse the links between it and the Department data type.

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 that are 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

The final task that you want the policy to perform is to increase the severity of the alert.

About this task

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 is mission critical are the data center and transaction processing units.

To perform this task, the policy must iterate through each of the Department data items that are 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 that were 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

After you finish writing the policy, you can review it for accuracy and completeness.

About this task

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.

Before you begin

Before you run the solution, you must configure the TUT_READER_01 OMNIBus event reader service so that it triggers the EnrichEvent policy. To configure TUT_READER_01:

1. Open Netcool/Impact and select **System Configuration > Event Automation > Services**
2. Select the the TUT_READER_01 service and click **Edit**.
3. Click the **Event Mapping** tab.
4. To create a mapping, click the **New Mapping** button.
5. If you want to trigger the **EnrichEvent** policy for all events, leave the **Filter Expression** field empty. If you want to trigger the **EnrichEvent** policy for specific events, enter the values for these events.
6. Select **EnrichEvent** in the **Policy to Run** field.
7. Click the **Active** check box.
8. To save the configuration, click **Ok**.
9. To save the changes to the TUT_READER_01 service, click the save icon.

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 13. 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 26. 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 27. 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 28. Settings for CityTemperature rule

Entry fields	Value
Instance Name	city
Expression	temperature <
Select the Status checkbox.	
Marginal	80
Bad	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 29. 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 30. 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 31. 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.
- Click the **Save** button in the **Edit Service** tab toolbar.
 - The Service Navigation portlet displays the new service in the **Services** tree.
 - 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.
 - 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

- 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.
- 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.
- Select the **Services** portlet. Click **Next**.
- On the General page, enter Weather by City in the **Name** field.
- Scroll through the thumbnail icon choices for the portlet, and choose



according to the figure below.

- Choose the Description Image for the new portlet as shown in the figure below:



- Select **TBSM** and click the **Add >** button to add the new portlet to the TBSM catalog.
- Click **Next**. The Security page is launched.
- On the Security page, select **User** from the Selected Roles list.
- Click **Add** to view a list of roles that can access this page.
- 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 14. Working with the Netcool/Impact UI data provider

You can use the UI data provider in Netcool/Impact to provide data to UI data provider compatible clients.

Integration with Netcool/Impact

The UI data provider accesses data from data sources, data types, and policies in Netcool/Impact.

The console is a component of Jazz for Service Management called IBM Dashboard Application Services Hub. Jazz for Service Management is bundled with Netcool/Impact. The IBM Dashboard Application Services Hub is referred to as the console in the rest of this documentation.

You can use the UI data provider to visualize data from Netcool/Impact in the console. You can use the console to create your own widgets or you can use one of the customizable self service dashboards.

Jazz for Service Management

The UI data provider requires Netcool/Impact 6.1.1 or higher. The UI data provider also requires Jazz for Service Management, which is available for download from the same page as Netcool/Impact 6.1.1. You use the installer that is provided with Jazz for Service Management to install it separately.

We recommend that you install Netcool/Impact and Jazz for Service Management on separate servers. If you do install Netcool/Impact and Jazz for Service Management on the same server, you must change the default port numbers to avoid a conflict between the versions of Tivoli Integrated Portal 2.2.x.x used by the GUI Server and the IBM Dashboard Application Services Hub, referred to as the console in this documentation. For example, a GUI Server with Tivoli Integrated Portal installed uses port 16310 as the default port. The console dashboards in Jazz for Service Management use the same port. In this case, you must change the port that is used by the console dashboards in Jazz for Service Management, for example to 18310.

For more information about Jazz for Service Management, see http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.psc.doc_1.1.0/psc_ic-homepage.html

Getting started with the UI data provider

Before you can use the Netcool/ImpactUI data provider, you must complete the prerequisites.

Prerequisites

Before you can use the UI data provider, you must ensure that the correct components are installed.

You must configure the user authorization.

Finally, you must create the data source and data types or the policy that you want to use to provide data to the UI data provider.

Visualizing data in the console

To visualize data in the console, you must complete the following tasks:

1. Create the remote connection between the UI data provider and the console.
2. Create the data model or policy that provides the data from Netcool/Impact.
3. Create a page in the console.
4. Create a widget on the page in the console.

UI data provider components

Before you can use the UI data provider, you must ensure that you install all the required components.

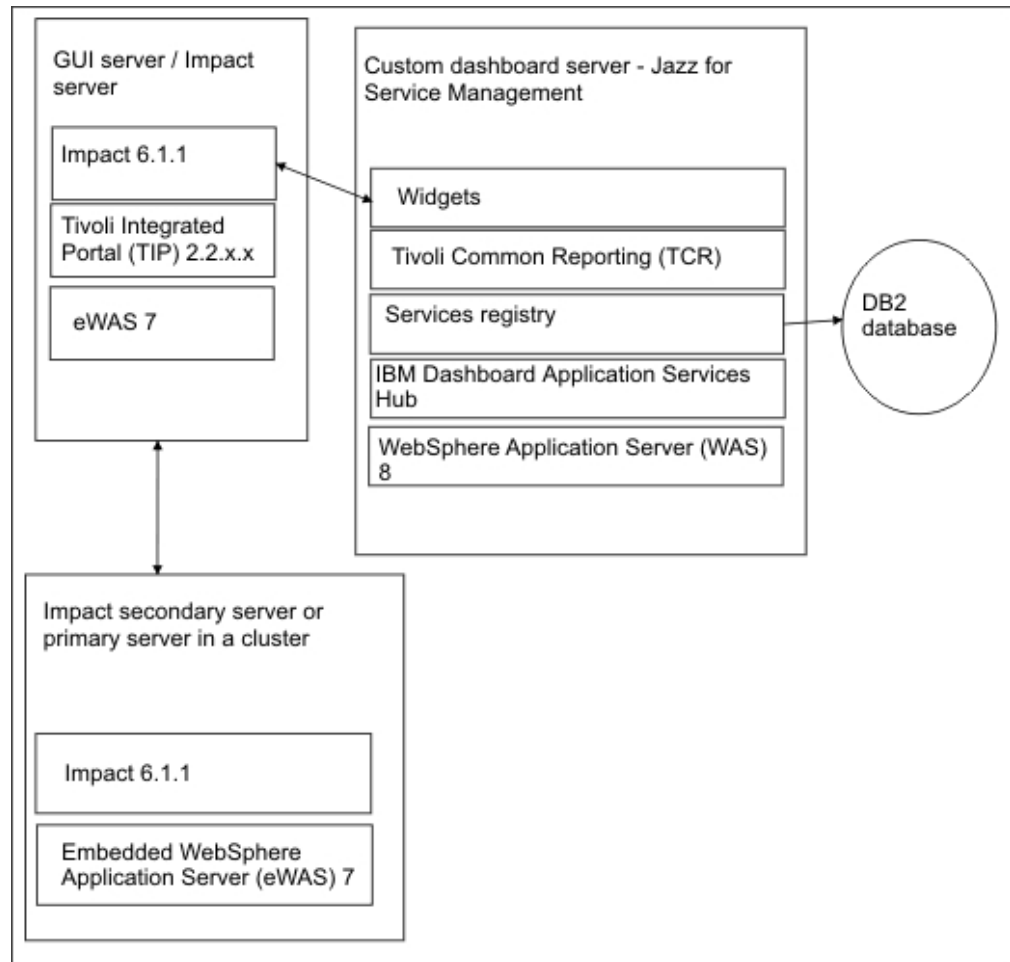
Required Jazz for Service Management components

Before you can use the UI data provider, you must install the IBM Dashboard Application Services Hub (the console) component of Jazz for Service Management.

If you want to use the Netcool/Impact self service dashboard (SSD) widgets, you must install the SSD widgets on the Dashboard Application Services Hub Server. For more information, see “Installing the Netcool/Impact Self Service Dashboard widgets” on page 200.

Component overview

The following graphic outlines the components that are required for using the UI data provider.



The system landscape is made up of a GUI Server or Impact Server, a secondary server or a primary server in a clustered environment, and a custom dashboard server.

Netcool/Impact 6.1.1 is installed on the GUI Server, the primary Impact Server and the secondary Impact Server. The GUI Server is installed as part of Tivoli Integrated Portal (TIP) 2.2.x.x.

The custom dashboard server uses the widgets that are created in TIP 3.1 to connect to the GUI Server. The custom dashboard uses the Registry Services component that is provided by Jazz for Service Management to connect to a DB2 database. For more information about the Registry Services component, see http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.psc.doc_1.1.0/psc_ic-homepage.html.

The custom dashboard server runs Tivoli Integrated Portal (TIP) 3.1 in contrast to the other servers that run Tivoli Integrated Portal (TIP) 2.2.x.x. This is because you can only create the widgets that facilitate the integration in Tivoli Integrated Portal (TIP) 3.1.

Configuring user authentication

Before you can use the UI data provider, you must assign one of the following roles.

About this task

User authentication is controlled by **impactUIDataProviderUser** role. This role is administered in the Tivoli Integrated Portal profile.

Procedure

You must assign one of the following roles to users to enable access to the UI data provider:

- **iscadmins**
- **impactAdminUser**
- **impactFullAccessUser**
- **impactUIDataProviderUser**

For more information about this role, see the information about the **impactUIDataProviderUser** role in the working with roles section of the *User Interface Guide*.

Data types and the UI data provider

To ensure that the data type can send data to the UI data provider, you must ensure that the following settings exist. Internal and SNMP data types require additional settings.

When you create a data type, you must consider the following settings:

- For all data types, except the internal data type, you must also select the **Key Field** check box for the key field. The key field identifies the uniqueness of the data that is displayed by the widget in the console.
- You must enable the data type so that it can send data to the UI data provider. To ensure that the data type can be accessed by the UI data provider, open the data type editor and select the **Access the data through UI data provider: Enabled** check box. After the data model refreshes, the data type is available as a data provider source. The default refresh rate is 5 minutes. However, this setting can be changed. For more information about how to change the refresh rate, see “UI data provider customization” on page 207.
- You must select a display name that does not contain any special characters or spaces. To select the display name, in the navigation tree, expand **System Configuration > Event Automation** and click **Data Model** to open the **Data Model** tab. Expand the data source that the data type belongs to and click the data type that you want to use. Select the field that you want to use as the display name from the **Display Name Field** list.
- If the data type key field value contains quotation marks ("), the console cannot support the widget that is based on the data type. This means that you cannot click the row that is based on the key field, use it to send an event to another widget or to provide hover preview information. You must use a key field that does not contain quotation marks.

Internal data types

If you use Internal data types, you must select one of the fields that belongs to the data type from the **Display Name Field** list in the data type editor. You must not select **KEY**. You must select an existing field in your schema definition. The UI data provider uses the value in this field as the item identifier. You must choose a unique value for the key field, otherwise the UI data provider overwrites your chosen key with the most recent value.

SNMP data types

If you use SNMP data types, you must define a value in the **Field Name** field in the data type editor. The UI data provider uses the value from the **Field Name** field as the item identifier. If more than one entry for the same value in **Field Name** field exists, the UI data provider uses the entry that was created most recently. If you want the UI data provider to use an item identifier that is unique, enter a unique value in the **Field Name** field for the data type.

Integrating chart widgets and the UI data provider

If you use the pie or line chart widgets to visualize data in the console, you must change the number of items per page from **All** to a number to ensure that the console can display the data.

Procedure

1. Open a page in the console or create a new one.
2. Select a widget. To select a widget, click **All** and drag the widget into the content area.
3. Configure the widget data. To configure the widget data, click it in the content area and click the **Down arrow icon** > **Edit**. The **Select a dataset** window is displayed.
4. Select the dataset that you want to use to provide the data. To search for the data type, enter the data type name and click the **Search** button. To display a list of all the available data types, click the **Show All** icon.
5. Click **Settings**. In the **Items per page list**, change the number of items per page from **All** to a number. For example, change it to 50.
6. Click **OK** to save the widget.

Names reserved for the UI data provider

The following names are reserved for use by the UI data provider. You cannot use these names in your policies and databases.

The UI data provider uses the comma (,) and ampersand (&) characters on the UI and in the URL for policies that use the DirectSQL policy function. You can use AS instead of ampersand (&) in policies.

The UI data provider uses the UIObjectId field to index key fields. You cannot use the UIObjectId field in any of your policies.

Netcool/Impact uses the AS UIDPROWNUM field in the query that is used for DB2 and Oracle databases. You cannot use UIDPROWNUM as a field name for any of the connected DB2 and Oracle databases.

The topology and tree widgets use the UITreeNodeId field. The UITreeNodeId field is reserved for use by the UI data provider and it contains the following fields that are also reserved:

- UITreeNodeId
- UITreeNodeParent
- UITreeNodeStatus
- UITreeNodeLabel
- UITreeNodeType

General steps for integrating the UI data provider and the console

You can use a variation of the general steps that are listed here to visualize data from the Netcool/Impact UI data provider in the console.

The exact steps differ depending on whether you use a data type or policy to provide the data. However, in general, to integrate the UI data provider and the console, you must complete the following activities:

1. Create the remote connection.
2. Create the information provider.
If you want to visualize data directly from a DB2 table or other data source, you must set up the data model in Netcool/Impact.
If you want to visualize data directly from a policy in Netcool/Impact, you must create the policy. For example, if you want to use a policy to mashup data from two different sources, you must create a policy in Netcool/Impact that summarizes the data.
3. Create a page and widget in the console.

Setting up the remote connection between the UI data provider and the console

Before you can visualize data from the UI data provider in the console, you must configure the remote connection between the UI data provider and the console.

About this task

The connection from Jazz for Service Management (the console) is to the host and port where the GUI Server with the Tivoli Integrated Portal profile is running. The default port for the Tivoli Integrated Portal profile is 16311 for HTTPS and 16310 if you are using a HTTP protocol.

Procedure

1. Open the console.
2. To open the **Connections** window, click **Settings > Connections**. The **Connections** window lists all the available data providers.
3. To create a new remote provider to represent the Netcool/Impact UI data provider, click the **Create new remote provider** icon and complete the following settings:
 - a. Select the protocol from the list. For example, HTTPS.
 - b. Enter the host name. For example, the IP address of the GUI Server.
 - c. Enter the port number. For example, for HTTPS the default value is 16311.
 - d. Enter the user name and password that you used when you installed the Impact Server.
 - e. Select the data provider that you created. To view all the available data providers, click **Search**. After you select the data provider, the **Name** and **Provider ID** fields are automatically populated. If you use multiple servers in a clustered environment, a connection is displayed for each server. For example, if you use Netcool/Impact in a cluster with TBSM, a connection is displayed for both members of the cluster.
4. To create the remote provider connection, click **Ok**.

Creating the data model

Before you can integrate the UI data provider and the console, you must create a data source and data type to provide data.

Before you begin

Before you create a data type, there are a number of specific settings that are required to facilitate the integration with the UI data provider. For more information about these settings, see “Data types and the UI data provider” on page 162.

Procedure

1. Create a data source.
2. Create a data type.

Results

The changes are only visible after the refresh interval. The refresh rate is 5 minutes by default. For more information about how to change this setting, see “UI data provider customization” on page 207.

Creating a page in the console

Before you can create a widget to visualize the data from the UI data provider, you must create a page in the console.

Procedure

1. Open the console.
2. To create a page, click **Settings > New Page**. Alternatively, you can also select **Getting Started > Build a page on the desktop**.
3. Enter the name of the page.
4. Click **OK** to save the page.

Results

The page is created. You can now create a widget to visualize the data.

Creating a widget on a page in the console

To visualize data from Netcool/Impact on a page in the console, you must create a widget.

About this task

Before you create a widget, you must create a page in the console. For more information, see “Creating a page in the console.”

Procedure

1. Open the page that you want to use for this widget in the console.
2. Select a widget. To select a widget, click **All** and drag the widget into the content area.

For example, click **All**, select the table widget, and drag the widget into the content area.

3. Configure the widget data. To configure the widget data, click it in the content area and click the **Down arrow icon > Edit**. The **Select a dataset** window is displayed.
4. Select the dataset that you want to use to provide the data. To search for the data source, data type, or policy that provides the data, enter the data type name and click the **Search** button. If you use a data type from Netcool/Impact to provide data for the widget, you can search for either the data source or data type name. If you use a policy from Netcool/Impact, you can search for the policy name or the output parameter name.

If you configured any specific policy-related actions on a policy to be used with a UI data provider when you create the widget and right-click an action in the widget the policy-related actions are displayed.

The data type is only displayed after the defined refresh interval. The default is 5 minutes. If you use a data type that you just created, you must wait 5 minutes before the data type displays.
5. If you want to use the line or pie chart widget, you must change the number of items per page from **All** to a number. To do so, click **Settings**. In the **Items per page list**, change the number of items per page from **All** to a number.
6. To save the new widget, click **OK**.

Results

You can now use the new widget to visualize the data from the specified data type in the console.

Accessing data from Netcool/Impact policies

You can use the UI data provider to access data from Netcool/Impact policies.

You must create user output parameters for each policy that you want to use with the UI data provider.

In addition, if you use an Impact object, an array of Impact objects, the `DirectSQL` or the `GetByFilter` policy function, you need to be aware of certain special requirements. These special cases are described and, where required, an example is provided.

If your policy retrieves data from a data base where the key field contains quotation marks ("), the console cannot support the widget that is based on the data provided by the policy. This means that you cannot click the row that is based on the key field, use it to send an event to another widget or to provide hover preview information. You need to use a key field that does not contain quotation marks.

Configuring user parameters

To use the UI data provider or OSLC with your Netcool/Impact policies, you must configure user parameters to make the policy results compatible with the UI data provider or available as OSLC resources.

About this task

You can create either policy runtime parameters or policy output parameters. Policy runtime parameters represent the runtime parameters that you define in policies. For example, you can use a policy runtime parameter to pass values from one policy to another in a data mashup.

Policy output parameters represent the parameters that are output by policies. For example, the UI data provider uses policy output parameters to visualize data from policies in the console.

Procedure

1. To open the policy user parameter editor in the policy editor toolbar, click the **Configure User Parameters** icon.
2. To create a policy output parameter, click **New Output Parameter:New**. To create a policy runtime parameter, click **New Runtime Parameter:New**. Mandatory fields are denoted by an asterisk (*). You must enter a unique name in the **Name** field.
3. Define the custom schemas for the output parameters if required.
If you are using the DirectSQL policy function with OSLC, you must define the custom schema for it.
If you are using **DirectSQL**, **Impact Object**, or **Array of Impact Object** with the UI data provider or the chart widget, you must define the custom schema for these values.
For more information, see “Creating custom schema values for output parameters” on page 170
4. To save the changes to the parameters and close the window, click **OK**.

Example

This example demonstrates how to create output parameters for a policy. First, you define a simple policy, like:

```
first_name = "Mark";  
zip_code = 12345;  
Log("Hello " + first_name + " living at " + zip_code);
```

Next, define the output parameters for this policy. In this case, there are two output parameters. You enter the following information:

Table 32. PolicyDT1 output parameter

Field	User entry
Name	Enter a unique name. For example, PolicyDT1.
Policy variable name	first_name
Format	String

Table 33. PolicyDT2 output parameter

Field	User entry
Name	Enter a unique name. For example, PolicyDT2
Policy variable name	zip_code

Table 33. PolicyDT2 output parameter (continued)

Field	User entry
Format	Integer

Accessing Netcool/Impact object variables in a policy

You can use the `NewObject` function to create Netcool/Impact objects in a policy. If you want to access these objects from the UI data provider, you must create a policy output parameter.

Procedure

1. To open the policy user parameter editor, click the **Configure User Parameters** icon in the policy editor toolbar. You can create policy user parameters for runtime and output. Click **New** to open the Create a New Policy Output Parameter window as required.
2. Select **Impact Object** from the **Format** list.
3. Enter the policy object name in the **Policy Variable Name** field.

Example

The following example demonstrates how to make an Impact object variable available to the UI data provider. First, you create the following policy, called `Test_Policy2`:

```
MyObject = NewObject();
MyObject.fname = 'Sam';
MyObject.age = 25;
MyObject.bmi = 24.5;
```

Define the output parameters for the policy as follows:

Table 34. PolicyObject1 output parameter

Field	User entry
Name	Enter a unique name. For example PolicyObject1.
Policy variable name	MyObject
Format	Impact Object

Accessing data types output by the GetByFilter function

If you want to access the results from the `GetByFilter` function, you must create output parameters for the UI data provider.

Procedure

1. To open the policy user parameter editor, click the **Configure User Parameters** icon in the policy editor toolbar. You can create policy user parameters for runtime and output. To open the Create a New Policy Output Parameter window, click **New**.
2. Select data type as the format.
3. Enter the name of the data item to which the output of the `GetByFilter` function is assigned in the **Policy Variable Name** field.
4. Enter the name of the data source in the **Data Source Name** field.
5. Enter the name of the data type in the **Data Type Name** field.

Example

This example demonstrates how to make the output from the `GetByFilter` function available to the Netcool/Impact UI data provider.

You created a data type that is called `ALERTS` that belongs to the `defaultobjectserver` data source. This data type belongs to Netcool/OMNIBus and it points to `alerts.status`. The key field is `Identifier`. The following four rows of data are associated with the key field:

- Event1
- Event2
- Event3
- Event4

Create the following policy, called `Test_Policy3`:

```
MyAlerts = GetByFilter("ALERTS", "Severity > 0", false);
```

Define the output parameters for the policy as follows:

Table 35. *PolicyData1* output parameter

Field	User entry
Name	PolicyData1
Policy variable name	MyAlerts
Format	Datatype
Data source name	defaultobjectserver
Data type name	ALERTS

Select the output parameter as the dataset for the widget that you are using to visualize the data.

1. Open the console and open a page.
2. Drag a widget into the content area.
3. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset** window is displayed.
4. Select **PolicyData1** as the data type that belongs to the `defaultobjectserver` data source.
5. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.

Accessing data types output by the DirectSQL function

If you want to access the results from the `DirectSQL` policy function, you must create output parameters for the UI data provider.

About this task

The comma (,) and ampersand (&) characters are reserved as special characters for the user interface and the URL. You cannot use these characters in policies that are accessed by the `DirectSQL` policy function. You can use `AS` instead of ampersand (&) in policies as required.

For example, consider the following policy:

```
SELECT "My&Test" AS My_Test FROM test_table
```

This policy returns the field name `My_Test` instead of `My&Test`

Procedure

1. To open the policy user parameter editor, click the **Configure User Parameters** icon in the policy editor toolbar. You can create policy user parameters for run time and output. To open the Create a New Policy Output Parameter window, click **New**.
2. Select **DirectSQL / UI Provider Datatype** as the format.
3. Enter a name for the output parameter.
4. Enter the name of the data item to which the output of the DirectSQL function is assigned in the **Policy Variable Name** field.
5. To define the DirectSQL format values, click the **Open the schema definition editor** editor icon. For detailed information about how to create custom schema values, see “Creating custom schema values for output parameters”

Creating custom schema values for output parameters

When you define output parameters that use the **DirectSQL**, **Array of Impact Object**, or **Impact Object** format in the user output parameters editor, you also must specify a name and a format for each field that is contained in the **DirectSQL**, **Array of Impact Object**, or **Impact Object** objects.

About this task

Custom schema definitions are used by Netcool/Impact to visualize data in the console and to pass values to the UI data provider and OSLC. You create the custom schemas and select the format that is based on the values for each field that is contained in the object. For example, you create a policy that contains two fields in an object:

```
01.city="NY"  
01.ZIP=07002
```

You define the following custom schemas values for this policy:

Table 36. Custom schema values for City

Field	Entry
Name	City
Format	String

Table 37. Custom schema values for ZIP


Field	Entry
Name	ZIP
Format	Integer

If you use the DirectSQL policy function with the UI data provider or OSLC, you must define a custom schema value for each DirectSQL value that you use.

If you want to use the chart widget to visualize data from an Impact object or an array of Impact objects with the UI data provider and the console, you define custom schema values for the fields that are contained in the objects. The custom

schemas help to create descriptors for columns in the chart during initialization. However, the custom schemas are not technically required. If you do not define values for either of these formats, the system later rediscovers each Impact object when it creates additional fields such as the key field. UIObjectId, or the field for the tree widget, UITreeNodeId. You do not need to define these values for OSLC.

Procedure

1. In the policy user parameters editor, select **DirectSQL**, **Impact Object**, or **Array of Impact Object** in the **Format** field.
2. The system shows the **Open the Schema Definition Editor** icon  beside the **Schema Definition** field. To open the editor, click the icon.
3. You can edit an existing entry or you can create a new one. To define a new entry, click **New**. Enter a name and select an appropriate format.
To edit an existing entry, click the **Edit** icon beside the entry that you want to edit
4. To mark an entry as a key field, select the check box in the **Key Field** column. You do not have to define the key field for Impact objects or an array of Impact objects. The system uses the UIObjectId as the key field instead.
5. To delete an entry, select the entry and click **Delete**.

Accessing an array of Impact objects with the UI data provider

Before you can use the UI data provider to access an array of Impact objects, you must create an output parameter that represents the array of Impact objects.

About this task

Netcool/Impact uses the field UIObjectId to index the key fields. As a result, UIObjectId is a reserved field name. You must not use UIObjectId as a custom field in any of your policies.

Procedure

1. To define an output parameter for the array of Impact objects, click the **Configure User Parameters** icon in the policy editor toolbar. To open the **Create a New Policy Output Parameter** window, click **New**. Create the output parameter as outlined in the following table:

Table 38. Output parameter for a policy that contains the Array of Impact Objects array

Field	Instructions
Name	Enter a name for the output parameter.
Policy Variable Name	Enter a name that is identical to the name of the array of Netcool/Impact objects in the policy that you want to reference.
Format	Select Array of Impact Objects .

2. After you create the output parameter, you define the custom schema values for the array of Impact objects. For more information, see “Creating custom schema values for output parameters” on page 170.
3. To display all the fields and values that are associated with the array, use the following URL:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER /datasources/<datasourceid>/datasets/  
<outputparametername>/items?properties=all
```

where *<outputparametername>* is the name of the parameter that is defined in the previous step.

Example

For example, consider the following Netcool/Impact objects:

```
MyObject1=NewObject();
MyObject1.firstname="first_name";
MyObject1.lastname="last_name";

MyObject2=NewObject();
MyObject2.city="mycity";
MyObject2.state="mystate";
```

An Impact Policy Language (IPL) policy references the array as follows:

```
MyArrayOfObjects={MyObject1,MyObject2};
```

A JavaScript policy references the array as follows:

```
MyArrayOfObjects=[MyObject1,MyObject2];
```

To map `MyArrayOfObjects` to the output parameters, create the output parameter for the array of objects as follows:

Table 39. Output parameters for MyArrayOfObjects

Field	User entry
Name	MyArrayOfObjects
Policy Variable Name	MyArrayOfObjects
Format	Array of Impact Object

To map the values that are contained in the array, create the custom schema values as follows:

Table 40. first_name custom schema values

Field	User entry
Name	first_name
Format	String

Table 41. last_name custom schema values

Field	User entry
Name	last_name
Format	String

Table 42. mycity custom schema values

Field	User entry
Name	mycity
Format	String

Table 43. mystate custom schema values

Field	User entry
Name	mystate

Table 43. mystate custom schema values (continued)

Field	User entry
Format	String

Use the following URL to view the fields and values for MyArrayOfObjects:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER /datasources/<datasourceid>/datasets/  
MyArrayOf1/items?properties=all
```

UI data provider and the IBM Dashboard Application Services Hub

To create visualizations and mashups of Netcool/Impact data from sources such as Netcool/Impact policies and database tables in the IBM Dashboard Application Services Hub, referred to as the console throughout this section, you can integrate the Netcool/Impact UI data provider with the console.

Filtering data in the console

You can use the console to filter data based on runtime parameters. This data can be derived from Netcool/Impact policies or other data types. To filter data in the console, you configure the widget settings in the console.

About this task

You make these settings in the table widget UI in the console. To make the settings that are described here, open the table widget UI and click **Edit**.

Procedure

- To filter data provided by Netcool/Impact policies, you must select the **executePolicy** check box to include the `executePolicy` Boolean parameter in the policy. The `executePolicy` parameter ensures that the policy runs when the user opens the widget. The system then populates the widget with the required data from the policy.

If you want to enter values for the policy runtime parameters in the console, you can enter these values under **Configure Optional Dataset Parameters**. The system passes the values to the runtime parameters in the policy while the policy is running.

Attention: The runtime parameters must be already defined in the policy. If the runtime parameters are not defined in the policy, the system cannot pass the values for the runtime parameters to the policy. For more information about how to create policy runtime parameters, see “Configuring user parameters” on page 166.

- To filter data from other sources, such as data derived from a database table, users can enter values for the filter parameters in the **Configure Optional Dataset Parameters** section in the console. Netcool/Impact uses the values that are entered here to filter the results that are displayed in the console.

Example: filtering data based on a database table

For example, you want to configure a console widget to display the rows from a database table that contain a value of 192.168.1.119 for the **Device** field. In the console under **Configure Optional Dataset Parameters**, enter 192.168.1.119 in the **Device** field. The widget returns only the data that contains this value in the **Device** field.

Integrating the tree widget with an Impact object or an array of Impact objects

Before you can integrate a policy that contains an Impact object or an array of Impact objects with the tree widget that is available in the console, you must specify certain fields in the policy and create the required custom schema definitions.

Procedure

1. If the object is a parent with an indexed ID, add the `UITreeNodeId` field for the object. If the object is a child, add the `UITreeNodeParent` field for the object. If you do not add these values, the object is not displayed as a tree hierarchy. The following conditions also apply:
 - The first object in the hierarchy cannot be a child, as is the case for all hierarchies.
 - You must specify the parent object before the child.
 - A child object cannot use the same ID for itself and its parent.
 - The parent ID is an indexed ID and it must start with 0. If you want to skip the parent ID, you must do so in this order.
 - The schema of each object must be the same, as is the case for all objects that use the tree widget. In other words, an object can use less schema elements than its parent object but these elements must be defined in the parent object. A child object cannot use additional schema elements that are not defined in the parent object.

The `UITreeNodeId` and `UITreeNodeParent` fields are not displayed in the console

2. Create a policy output parameter for the Impact object or the array of Impact objects. To create a policy output parameter, click the **Configure User Parameters** icon in the policy editor toolbar. To open the **Create a New Policy Output Parameter** window, click **New**. Create the following entries:

Table 44. User output parameters for Impact object or array of Impact objects

Field	Instructions
Name	Enter a name for the output parameter.
Policy Variable Name	Enter a name that is identical to the name of the Impact object or the array of Impact objects that is specified in the policy that you want to reference.
Format	Choose Impact Object or Array of Impact Objects .

3. Create the custom schema values for the values that you want to display as columns in the console. You must specify a custom schema value for the Impact object or the objects that are contained in an array of Impact objects. The schema values that you define can be displayed as columns in the console. You only need to specify custom schema values for the values that you want to display. Values such as `UITreeNodeId` are displayed as properties unless you specify them as custom schema values.

To create a custom schema value in the policy editor, select **Impact object** or **Array of Impact objects** in the **Format** field. To open the editor, click the **Open the Schema Definition Editor** icon. Define the custom schema values as outlined in the following table:

Table 45. Custom schema values for Impact object or array of Impact objects

Field	Instructions
Name	Enter the name of the custom schema value. For example, this could be the name of the Impact object or the name of one of the objects in the array of Impact objects.
Format	Choose the format of the custom schema value. For example, if the parameter is a string, choose String .

Example

The following example demonstrates how to integrate an array of Impact objects and the tree widget.

1. Create a policy that contains an array of Impact objects and the additional fields that are required for the tree widget, `UITreeNodeId` and `UITreeNodeParent`.

```
Log("Array of objects with same fields....");
O1=NewObject();
O1.UITreeNodeId=0;
O1.fname="o1fname";
O1.lname="o1lname";
O1.dob="o1dob";

O2=NewObject();
O2.UITreeNodeId=1;
O2.UITreeNodeParent=0;
O2.fname="o2fname";
O2.lname="o2lname";
O2.dob="o2dob";

O3=NewObject();
O3.UITreeNodeId=2;
O3.UITreeNodeParent=1;
O3.fname="o3fname";
O3.lname="o3lname";
O3.dob="o3dob";

O4=NewObject();
O4.UITreeNodeId=3;
O4.UITreeNodeParent=20;
O4.fname="o4fname";
O4.lname="o4lname";
O4.dob="o4dob";

O5=NewObject();
O5.UITreeNodeId=4;
O5.fname="o5fname";
O5.lname="o5lname";
O5.dob="o5dob";

O6=NewObject();
O6.UITreeNodeParent=4;
O6.fname="o6fname";
O6.lname="o6lname";
O6.dob="o6dob";

O7=NewObject();
O7.UITreeNodeParent=4;
O7.fname="o7fname";
O7.lname="o7lname";
```

```

07.dob="o7odb";

08=NewObject();
08.UITreeNodeParent=4;
08.fname="o8fname";
08.lname="o8lname";
08.dob="o8odb";

09=NewObject();
09.fname="o9fname";
09.lname="o9lname";
09.dob="o9odb";

010=NewObject();
010.fname="NJ";
010.lname="Bayonne";
010.dob="April 1st 2011";

011=NewObject();
011.UITreeNodeParent=11;
011.fname="o11fname";
011.lname="o11lname";
011.dob="o11odb";

012=NewObject();
012.UITreeNodeId=11;
012.UITreeNodeParent=0;
012.fname="o12fname";
012.lname="o12lname";
012.dob="o12odb";

0a=NewObject();
0a.UITreeNodeId=12;
0a.UITreeNodeParent=2;
0a.fname="oafname";
0a.lname="oalname";
0a.dob="oaodb";

0b=NewObject();
0b.UITreeNodeId=13;
0b.UITreeNodeParent=12;
0b.fname="obfname";
0b.lname="oblname";
0b.dob="obodb";

0c=NewObject();
0c.UITreeNodeId=14;
0c.UITreeNodeParent=14;
0c.fname="ocfname";
0c.lname="oclname";
0c.dob="ocodb";

0e=NewObject();
0e.UITreeNodeParent=14;
0e.fname="oefname";
0e.lname="oelname";
0e.dob="obedb";

0s={01,02,03,04,05,06,07,08,09,010,012,011,0a,0b,0c,0e};
log("0s " + 0s);

```

2. In the policy editor, create the following user output parameter:

Table 46. ArrayofNewObject user output parameter

Field	User input
Name	ArrayofNewObject

Table 46. ArrayofNewObject user output parameter (continued)

Field	User input
Policy Variable Name	0s
Format	Array of Impact Objects

3. In the policy editor, create the following custom schema value definitions for the array of Impact objects:

Table 47. fname custom schema definition

Field	User input
Name	fname
Format	String

Table 48. lname custom schema definition

Field	User input
Name	lname
Format	String

Integrating data from a policy with the topology widget

Before you can use the topology widget to visualize data from a Netcool/Impact policy in the console, you need to specify certain fields in the policy.

About this task

The topology widget is intended for use with the tree widget. You use the fields described here with the fields that you specify for the tree widget. For more information, see “Integrating the tree widget with an Impact object or an array of Impact objects” on page 174

Generally, the nodes are connected in a hierarchy. However, this connection is not technically required. If you define a node that is not part of a hierarchy, it is displayed as a stand-alone node that is not part of any other hierarchy.

Procedure

- You must include the following statement in the first object in the policy:

```
ObjectName.UITreeNodeType= <Node Type>;
```

where <Node Type> is either GRAPH or TREE. If you do not specify a value, the default value is TREE.

- You must specify a label for each object. If you do not, the system displays **No label was specified** for the label and tooltip. To specify a label, add the following statement for each object:

```
ObjectName.UITreeNodeLabel=<Tooltip text>
```

where <Tooltip text> is the text that is used for the label and tooltip.

- Define the status for each node. This status is not mandatory. If you do not add this statement, the status is unknown. If you want to display the status for each object, add the following statements for each node.

```
ObjectName.UITreeNodeStatus=<Status>;
```

where *<Status>* is the status. The table lists the supported values and the numbers that represent those values. You can use either the number or the word to represent the status.

Table 49. Tree node status

Status	Number
Critical	5
Major	4
Minor	3
Warning	2
Normal	0
Unknown	

Example

The following examples illustrate how to define a policy that you want to use with the tree and topology widgets. This example policy includes a multi-level topology and a node status that represents severity.

```
Log("Test Topo");
O0=NewObject();
O0.UITreeNodeType="GRAPH";
O0.UITreeNodeId=0;
O0.UITreeNodeLabel="NJ-Bayonne";
O0.UITreeNodeStatus="Warning";
O0.state="NJ";
O0.city="Bayonne";

O1=NewObject();
O1.UITreeNodeId=1;
O1.UITreeNodeStatus="Normal";
O1.UITreeNodeParent=0;
O1.UITreeNodeLabel="NY-Queens";
O1.state="NY";
O1.city="Queens";

O2=NewObject();
O2.UITreeNodeId=2;
O2.UITreeNodeStatus="Critical";
O2.UITreeNodeParent=1;
O2.UITreeNodeLabel="NC-Raleigh";
O2.state="NC";
O2.city="Raleigh";

O3=NewObject();
O3.UITreeNodeId=3;
O3.UITreeNodeParent=0;
O3.UITreeNodeStatus="Warning";
O3.UITreeNodeLabel="CA-Los Angeles";
O3.state="CA";
O3.city="Los Angeles";

O4=NewObject();
O4.UITreeNodeId=4;
O4.UITreeNodeParent=3;
O4.UITreeNodeStatus="Normal";
O4.UITreeNodeLabel="CO-Denver";
O4.state="CO";
O4.city="Denver";

O5=NewObject();
```



```

05.UITreeNodeId=5;
05.UITreeNodeStatus="Critical";
05.UITreeNodeParent=4;
05.UITreeNodeLabel="MA-Main";
05.state="MA";
05.city="Main";

06=NewObject();
06.UITreeNodeId=6;
06.UITreeNodeParent=0;
06.UITreeNodeStatus="Warning";
06.UITreeNodeLabel="NH-New Hampshire";
06.state="NH";
06.city="New Hampshire";

07=NewObject();
07.UITreeNodeId=7;
07.UITreeNodeParent=6;
07.UITreeNodeStatus="Normal";
07.UITreeNodeLabel="TX-Hudson";
07.state="TX";
07.city="Houston";

08=NewObject();
08.UITreeNodeId=8;
08.UITreeNodeParent=7;
08.UITreeNodeStatus="Critical";
08.UITreeNodeLabel="VA-Virgina Beach";
08.state="VA";
08.city="Virigina Beach";

Obs={00,01,02,03,04,05,06,07,08};

```

After you implement the policy you need to create the output parameters and custom schema values. For more information about how to do this, see “Configuring user parameters” on page 166.

Displaying status and percentage in a widget

You can show status and percentage in topology, tree, table, and list widgets by using policies or data types. To show status and percentages in a widget, you must create a script in JavaScript format in the data type. Or if the policy uses the `GetByFilter` function.

About this task

For data types, SQL, SNMP, and internal data types are supported. For policies the `GetByFilter`, `DirectSQL` and `Impact Object`, and `Array Of Impact Objects` are supported.

1. Create the data type.
2. In the data type configuration window, add the script to the **Define Custom Types and Values (JavaScript)** area.
3. Click the **Check Syntax and Preview Sample Result** button to preview the results and to check the syntax of the script.

For `DirectSQL` and `Impact Object`, `Array Of Impact Objects`, the `Status`, and `Percentage` can be specified when you create the schema definition. For policies, you can use `IPL` or `JavaScript` for the `DirectSQL` or `GetByFilter` functions.

The script uses the following syntax for data types and for policies that use the `GetByFilter` function.

```
ImpactUICustomValues.put("FieldName,Type",VariableName);
```

Where Type is either **Percentage** or **Status**. VariableName, can be a variable or hardcoded value. Always cast the variable name to String to avoid any error even if the value is numeric. See the following examples:

```
ImpactUICustomValues.put("MyField,Percentage",""+VariableName);
ImpactUICustomValues.put("MyField,Percentage","120");
ImpactUICustomValues.put("FieldName,Percentage",""+(field1/40));
```

The status field expects the value to be similar to the Topology widget configuration:

Table 50. Status field values

Status	Number
Critical	5
Major	4
Minor	3
Warning	2
Normal	0
Intermediate (Available when the connection to Netcool/Impact uses https.)	1

There is no limit to how many fields you can put in the variable *ImpactUICustomValues*. The variable must be at the very end of the script. Anything before the variable must be in JavaScript and can be anything if the variable *ImpactUICustomValues* is populated correctly.

Example 1:

Assigns the field name from the table to be the status or the percentage and assigns the field value. This example assigns **SHAREUP** and **SHAREDOWN** as the percentages, and **STANDING** as the status.

```
ImpactUICustomValues.put("SHAREUP,Percentage",SHAREUP);
ImpactUICustomValues.put("SHAREDOWN,Percentage",SHAREDOWN);
ImpactUICustomValues.put("PROFIT,Percentage",PROFIT);
ImpactUICustomValues.put("STANDING,Status",STANDING);
```

Example 2:

This example has an extra calculation to determine the value of percentage or status fields. The percentage assumes the maximum value to use is 100. Then, a factor is used to scale the values that are based on the maximum value that is expected by the user. The status and percentage is scaled based on a factor.

```
var status = "Normal";
var down = 0;
var up = 0;
var factor = ( TOTAL / 100);
down = (DOWN / factor);
up = (UP / factor);
var statusFactor = (DOWN / TOTAL) * 100;
if ( statusFactor >= 50 ) {
    status = "Critical";
} else if ( statusFactor >= 30 ) {
    status = "Major";
} else if (statusFactor >= 20) {
```

```

        status = "Minor";
    } else if (statusFactor >= 10 ) {
        status = "Warning";
    } else {
        status = "Normal";
    }
}
ImpactUICustomValues.put("DownPercentage,Percentage",""+down);
ImpactUICustomValues.put("UpPercentage,Percentage",""+up);
ImpactUICustomValues.put("NetworkStatus,Status",""+status);

```

Example 3:

This example uses extra fields that do not exist in the table and used to be the Status and Percentage. The values are the exact values that come from fields that exist in the table. Calculation can be used to assign different values:

```

ImpactUICustomValues.put("CPUPercentUsage,Percentage",CPUUsage);
ImpactUICustomValues.put("RAMPercentUsage,Percentage",RAMUsage);
ImpactUICustomValues.put("DiskPercentUsage,Percentage",DiskUsage);
ImpactUICustomValues.put("NetworkAvailability,Status",NetworkStatus);

```

Tip: The Table or List widget shows duplicate entries or have missing data when you compare the data to the data type data items. Check the data source to ensure that all keys are unique.

Tip: If you use a policy function to create a dynamic filter, you can get a message in the policy log. The messages states that the filter variable is not defined in policy. No eventing occurs between widgets. Check that you are not using any special characters in the custom value in the data type for example, `ImpactUICustomValues.put("CPU%,Percentage",""+value)`. The widgets do not support special characters in field names.

Tip: If a data type field type is incorrectly defined, for example the field is defined as an integer, but contains float values the widget fails to load. The widget shows a message similar to this example:

Failed to load

To resolve the issue, edit the data type field and select the correct data type float.

Visualizing data from the UI data provider in the console

You can use the Netcool/Impact UI data provider to visualize data in the IBM Dashboard Application Services Hub, referred to as the console throughout this section.

You can visualize data from Netcool/Impact in the console. You can use data types or Netcool/Impact policies to provide this data. You can also use Netcool/Impact policies to create mashups of data from multiple sources.

The example scenarios that are provided are intended to provide examples that help you when you are trying to visualize your own data in the console.

Before you can implement any of the examples below, you must set up the remote connection between Netcool/Impact and the console. For more information, see “Setting up the remote connection between the UI data provider and the console” on page 164.

Note: If you use the line or pie chart widget to visualize data from a DB2 data type, you must change the number of items per page from **All** to a number. For example, change it to 50.

Example scenario overview

Read the following example scenarios to get an overview of the possible ways to integrate the UI data provider and the console.

For more scenarios and examples visit the Netcool/Impact developerWorks wiki *Scenarios and examples* page available from the following URL:

<https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/Tivoli%20Netcool%20Impact/page/Scenarios%20and%20examples>

Visualizing data from a DB2 database table in a line chart

You can use the console to visualize data that is retrieved directly from a data type in Netcool/Impact.

About this task

This example uses a line chart to visualize data. You can use the same process to visualize the data in a bar, column, or line chart.

Procedure

1. Create a DB2 data source.
 - a. Enter **NewDataSource** in the **Data Source Name** field.
 - b. Enter the user name and password for the database.
 - c. Complete the other fields as required.
 - d. Save the data source.
2. Create a data type for the DB2 data source.
 - a. Enter **NewUIDPDT** as the name and complete the required fields.
 - b. To ensure that the data type is compatible with the UI data provider, select the **UI data provider: enabled** check box.
 - c. Select the key fields for the data type.
 - d. Save the data type.
3. Create a page in the console.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter **Page for DB2** in the **Page Name** field.
 - d. Save the page.
4. Create a widget in the console.
 - a. Open the **Page for DB2** page that you created.
 - b. Drag the **Line Chart** widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **NewUIDPDT** data type that belongs to the **NewDataSource** data source. The data type is only displayed after the defined refresh interval. The default is 5 minutes.

- e. The **Visualization Settings** UI is displayed. Enter the values that you want to use for the y axis. You can select multiple lines. You can also select a text to display as a tooltip. Click **Ok**.
- f. To ensure that the console can display all the items, change the number of items that are allowed per page from all to a number. Click **Settings**. In the **Items per page list** list, change the number of items per page from **All** to a number.
- g. To save the widget, click the **Save and exit** button on the toolbar.

Results

When you display the widget, the data is retrieved directly from the data type in Netcool/Impact and displayed in a line chart.

Visualizing data from a Netcool/Impact policy in a pie chart

You can use the pie chart widget in the console to visualize data from the UI data provider.

Procedure

1. Create a policy to provide the data for the pie chart widget. The policy must group all the items from a database table into a single Impact object.
For example, define the following policy, called `Policyforpiechart`, that gathers the rows in the database table into a single Impact object:


```
obj = NewObject ();
obj.name = 'Internet Banking';
obj.availabilityduringoperationalhours=99.9;
obj.availabilityduringnonoperationalhours=95;
```
2. Create the user output parameters for the policy.
 - a. In the policy editor, click the **Configure User Parameters** icon to create the output parameter.


Table 51. Output parameters for `MyNewObject`

Field	User entry
Name	MyNewObject
Policy Variable Name	Obj
Format	Impact Object

You must enter the name of the Impact object exactly as it is defined in the policy in the **Policy Variable Name** field.

- b. You must create the custom schema values for the fields in the object. In this example, the Impact object contains two fields that are integers.

After you select **Impact Object** in the **Format** field, the system displays the

Open the Schema Definition Editor icon  beside the **Schema Definition** field. To open the editor, click the icon.

You define the following custom schema definitions for the policy.

Table 52. Custom schema for operational and non-operational hours

Name	Format
availabilityduringoperationalhours	Integer
availabilityduringnonoperationalhours	Integer

3. Create a page.

- a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter Page for Pie Chart in the **Page Name** field.
 - d. Save the page.
4. Create a pie chart widget.
 - a. Open the **Page for Pie Chart** page that you created.
 - b. Drag the **Pie chart** widget into the content area.
 - c. To configure the widget data, click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **MyNewObject** datatype that belongs to the **Policyforpiechart** datasources. The dataset represents the user output parameter that you defined previously. The dataset is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box.
 - f. Select the values that you want to use for the pie chart. In this example, select **obj.availabilityduringoperationalhours** and **obj.availabilityduringnonoperationalhours**. To save your selection, click **Ok**.
 - g. To ensure that the console can display all the items, change the number of items allowed per page from all to a number. Click **Settings**. In the **Items per page list** list, change the number of items per page from **All** to a number.
 - h. To save the widget, click the **Save and exit** button on the toolbar.

Results

The data from the UI data provider is displayed as a pie chart in the console.

Visualizing data mashups from two web services in a table

You can use Netcool/Impact policies to create mashups of data from different sources such as web services. You can also use the console and the UI data provider to visualize the data from these mashups.

About this task

The following example uses a policy that is created in Netcool/Impact to retrieve data from two different web services. The policy uses an array to group the results. After you create the policy, you can use a table widget to visualize the data mashup.

Procedure

1. In the policy editor, create a policy that is called **TestArrayOfObjectsWebService**. For example, create the following policy. This policy is based on the WSDL at <http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL>. Create the policy as follows:
 - a. Define the package that was defined when the WSDL file was compiled in Netcool/Impact.


```
WSSetDefaultPKGName('weather');
```
 - b. Specify the parameters.


```
GetCityWeatherByZIPDocument=WSNewObject("com.cdyne.ws.weatherws.GetCityWeatherByZIPDocument");
_GetCityWeatherByZIP=WSNewSubObject(GetCityWeatherByZIPDocument,
```

```
"GetCityWeatherByZIP");

_ZIP = '07002';
_GetCityWeatherByZIP['_ZIP'] = _ZIP;
```

```
WSParams = {GetCityWeatherByZIPDocument};
```

- c. Specify the web service name, end point, and method.

```
WSService = 'Weather';
WSEndPoint = 'http://wsf.cdyne.com/WeatherWS/Weather.asmx';
WSMethod = 'GetCityWeatherByZIP';
```

- d. Use the GetbyXpath policy function to get the value for the element that you want.

```
nsMapping= NewObject();
nsMapping.tns = "http://wsf.cdyne.com/WeatherWS/";
nsMapping.xsd="http://www.w3.org/2001/XMLSchema" ;
nsMapping.soap="http://www.w3.org/2003/05/soap-envelope" ;
nsMapping.xsi="http://www.w3.org/2001/XMLSchema-instance";

log("About to invoke Web Service call GetCityWeatherByZIP .....");

WSInvokeDLResult = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams);
log("Web Service call GetCityWeatherByZIP return result: " +WSInvokeDLResult);

Result1=GetByXPath(""+WSInvokeDLResult, nsMapping, XPathExpr);

Object1=NewObject();
Object1.City=Result1.Result.City[0];
Object1.State=Result1.Result.State[0];
Object1.Temperature=Result1.Result.Temperature[0];
```

- e. Start the WebService call.

```
log("About to invoke Web Service call GetCityWeatherByZIP .....");

WSInvokeDLResult = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams);
log("Web Service call GetCityWeatherByZIP return result: " +WSInvokeDLResult);
```

- f. Define another call.

```
_ZIP = '23455';
_GetCityWeatherByZIP['_ZIP'] = _ZIP;

WSParams = {GetCityWeatherByZIPDocument};
log("About to invoke Web Service call GetCityWeatherByZIP .....");

WSInvokeDLResult = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams);
log("Web Service call GetCityWeatherByZIP return result: " +WSInvokeDLResult);
```

```
//Retrieve the element values and assign them to an object
XPathExpr = "//tns:State/text() |//tns:City/text() | //tns:Temperature/text()";
```

- g. Define and assign values to the Impact Objects.

```
xPathExpr = "//tns:State/text() |//tns:City/text() | //tns:Temperature/text()";
Result2=GetByXPath(""+WSInvokeDLResult, nsMapping, XPathExpr);

Object2=NewObject();
Object2.City=Result2.Result.City[0];
Object2.State=Result2.Result.State[0];
Object2.Temperature=Result2.Result.Temperature[0];

CustomObjs= {Object1,Object2};

log(CustomObjs);
```

2. Define the user output parameters for the array of objects.

You must create the following user output parameters for the array that is contained in the policy that you created. In the policy editor, click the **Configure User Parameters** icon to create the output parameters for the array of objects.

Table 53. Output parameters for MyArrayObj1

Field	User entry
Name	MyArrayOfCustomObjects
Policy Variable Name	CustomObjs
Format	Array of Impact Object

You must enter the exact name of the array as it is in the policy in the **Policy Variable Name** field.

3. Create a page.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter Page for Array of Objects in the **Page Name** field.
 - d. To save the page, click **Ok**.
4. Create a table widget.
 - a. Open the **Page for Array of Objects** page that you created.
 - b. Drag the **Table** widget into the content area.
 - c. To configure the widget data, click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **MyArrayOfCustomObjects** data type that belongs to the **TestArrayOfObjectsWebService** data source. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed. The system displays all the available columns by default. You can change the displayed columns in the **Visualization Settings** section of the UI. You can also select the row selection and row selection type options.
 - f. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
 - g. To save the widget, click the **Save and exit** button on the toolbar.

Results

When you open the table widget, the data from two different sources is displayed in a table.

Visualizing data mashups with an array of Impact objects

You can use Netcool/Impact policies to create mashups of data from the DirectSQL policy function and another sources. You can use the console and the UI data provider to visualize the data from these mashups in a table.

Procedure

1. In the policy editor, create a policy that is called MultipleObjectsPolicy. The policy uses the DirectSQL policy function to retrieve the data from the defaultobjectserver data source. The policy also retrieves data from one other source.

Create the following policy called MultipleObjectsPolicy.

```
Log("Test Policy with Multiple different objects..");
NodesVarJS=DirectSQL('defaultobjectserver',"SELECT Node,Identifier,
Severity from alerts.status", null);
Log("ClassOf() " + ClassOf(NodesVarJS));
Obj1=NewObject();
Obj1.fname="MyFirstName";
```



```

Obj1.lastName="MyLastName";
Obj1.city="MyCity";
MyObjsAll={Obj1};
i=0;
while(i < length(NodesVarJS) ) {
    O = newObject();
    O.Node=NodesVarJS[i].Node;
    O.Identifier=NodesVarJS[i].Identifier;
    O.Severity=NodesVarJS[i].Severity;
    MyObjsAll = MyObjsAll + {O};
    i = i +1;
}

Log("MyObjs is " + MyObjsAll);

```

2. Define the user output parameters for the array of objects in the policy.
In the policy editor, click the **Configure User Parameters** icon to create the output parameters for the array of objects.

Table 54. Output parameters for MyObjsAll

Field	User entry
Name	MyArrayOfObjects
Policy Variable Name	MyObjsAll
Format	Array of Impact Object

You must enter the exact name of the array as it is in the policy in the **Policy Variable Name** field.

3. Create a page.
 - Open the console.
 - To create a page, click **Settings > New Page**.
 - Enter Page for Table in the **Page Name** field.
 - Save the page.
4. Create a table widget.
 - a. Open the **Page for Table** page that you created.
 - b. Drag the **Table** widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **MyArrayOfObjects** data type that belongs to the **MultipleObjectsPolicy** data source. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed. The system displays all the available columns by default. You can change the displayed columns in the **Visualization Settings** section of the UI. You can also select the row selection and row selection type options.
 - f. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
 - g. To save the widget, click the **Save and exit** button on the toolbar.

Results

When you display the table widget, the data from the DirectSQL policy function and the other source is displayed in a table.

Visualizing data output by the GetByFilter policy function in a list

You can use the list widget to visualize data from a Netcool/Impact policy that contains the GetByFilter policy function in the console.

Procedure

1. In the Netcool/Impact policy editor, create a Netcool/Impact policy.
Create a policy that is called IPLGetByFilterPolicy that includes the GetByFilter policy function. You want to visualize the data that is output by the policy function. In this example, the filter is defined statically within the policy. In a real world situation, you might want to pass the values dynamically from the UI data provider to the policy function.

```
Log ("Executing IPL Impact Policy");  
filter = "SERVICEREQUESTIDENTIFIER = 1";  
GetbyFilter ('dataTypeforDemoUIDP', filter, false);
```

2. Define the output parameter for the data items. This parameter is made available to the UI data provider and are displayed as a data type in the console.

In the policy editor, click the **Configure User Parameters** icon and the **Policy output parameter:New** push button to create the output parameters for the data items.

Table 55. User output parameters for data items

Field	Entry
Name	DataFromPolicy
Policy Variable Name	DataItems
Format	Datatype
Data Source Name	localDB2UIDPTest
Data Type Name	dataTypeforUIDPdemo

3. Create a page in the console.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter Page for List in the **Page Name** field.
 - d. To save the page, click **Ok**.
4. Create a list widget.
 - a. Open the **Page for List** page that you created.
 - b. Drag the **List** widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **DataFromPolicy** datatype that belongs to the **IPLGetByFilterPolicy** datasources. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed. You must select values for the label, status, description and timestamp. You can also configure a number of optional settings such as the size of the page.
 - f. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
 - g. To save the widget, click the **Save and exit** button on the toolbar.

Results

When you display the widget, the policy is run. The information that is contained in the policy results is added to the widget and is displayed as a list.

Visualizing data output by the DirectSQL policy function in an analog gauge

You can use the analog gauge widget to visualize data from a Netcool/Impact policy that contains the DirectSQL policy function in the console.

Procedure

1. In the policy editor in Netcool/Impact, create a policy that uses the DirectSQL policy function.

Create the following policy that includes the DirectSQL policy function called TestDirectSQL:

```
Log("TestDirectSQL");
query= "select SUM(VALUE) as sumvalue, HISTORYRESOURCENAME, METRICNAME ,
( HISTORYRESOURCENAME || METRICNAME ) as key from
TBSMHISTORY.HISTORY_VIEW_RESOURCE_METRIC VALUE;
DirectSQL ('directSQLSample'.query.false);
```

This policy accesses data from a database table and sums a particular column in the table to create a sum value. The policy also groups a number of columns.

2. Define the output parameters for the policy.

To define the output parameters for the policy, click the **Configure User Parameters** icon.

Table 56. Output parameters for IPLDirectSQL policy

Field	Entry
Name	IPLDirectSQL
Policy Variable Name	DataItems
Format	DirectSQL

You must also create new custom schema values to represent the values that are contained in the fields of the DirectSQL policy function. After you select **DirectSQL** in the **Format** field, the system displays the **Open the Schema Definition Editor** icon. To create a value, click the icon. You must enter a name for each new value and select a format. For this example, create the following custom schema values:

Table 57. Custom schema values for IPLDirectSQL output parameter

Name	Value
HISTORYRESOURCENAME	String
METRICNAME	String
sumvalue	Float

3. Create a page.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter Page for Analog Gauge in the **Page Name** field.
 - d. To save the page, click **Ok**.
4. Create an analogue gauge widget
 - a. Open the **Page for Analog Gauge** page that you created.

- b. Drag the **Analog Gauge** widget into the content area.
- c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
- d. Select the dataset. Select the **IPLDirectSQL** datatype that belongs to the **TestDirectSQL** datasource. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
- e. The **Visualization Settings** UI is displayed. Select the value that you want to display in the gauge in the **Value** field. In this example, you enter SUMVALUE in the field. You can also select a number of other optional values for the gauge such as minimum value, maximum value and unit of measure.
- f. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
- g. To save the widget, click the **Save and exit** button on the toolbar.

Results

When you display the widget, the policy is run and the information that is contained in the policy results is displayed as a gauge.

Visualizing data with the tree and topology widgets

You can use the tree and topology widgets with the UI data provider to visualize hierarchical and topological data from a Netcool/Impact policy in the console.

About this task

You use the tree and topology widget to visualize a hierarchy with topological data in the console. This example demonstrates how to do so for a specific policy. For more information about the requirements for using these widgets, see “Integrating the tree widget with an Impact object or an array of Impact objects” on page 174 and “Integrating data from a policy with the topology widget” on page 177.

Procedure

1. In the policy editor in Netcool/Impact, create a policy that is called TestTreeTopoPolicy.

The policy uses the ArrayofImpactObjects policy function to retrieve information about addresses. This information is hierarchical. Entries are ordered by number, city, and country. You also want to add topological information about the status of an entry to the policy.

```
MyObject1=NewObject();
MyObject1.country="United States";
MyObject1.city="New York";
```

```
MyObject2=NewObject();
MyObject2.country="United States";
MyObject2.city="Philadelphia";
```

```
MyObject3=NewObject();
MyObject3.country="England";
MyObject3.city="London";
```

```
MyArrayOfObjects={MyObject1,MyObject2,MyObject3};
```

2. Next, you must make the policy compatible with the tree widget. To make the policy compatible with the tree widget, you must add the UITreeNodeId and UITreeNodeParent parameters to the policy.

```
MyObject1=NewObject();
MyObject1.UITreeNodeId=0;
MyObject1.country="United States";
MyObject1.city="New York";
```

```
MyObject2=NewObject();
MyObject2.UITreeNodeId=1;
MyObject2.UITreeNodeParent=0;
MyObject2.country="United States";
MyObject2.city="Philadelphia";
```

```
MyObject3=NewObject();
MyObject3.UITreeNodeId=2;
MyObject3.UITreeNodeParent=1;
MyObject3.country="England";
MyObject3.city="London";
```

```
MyArrayOfObjects={MyObject1,MyObject2,MyObject3}
```

3. Next, you must make the policy compatible with the topology widget. To make the policy compatible with the tree widget, you must add the `UITreeNodeType`, `UITreeNodeLabel`, and `UITreeNodeStatus` fields to the policy.

```
MyObject1=NewObject();
MyObject1.UITreeNodeId=0;
MyObject1.country="United States";
MyObject1.city="New York";
UITreeNodeType="GRAPH";
UITreeNodeLabel="NY";
UITreeNodeStatus="Major";
```

```
MyObject2=NewObject();
MyObject2.UITreeNodeId=1;
MyObject2.UITreeNodeParent=0;
MyObject2.country="United States";
MyObject2.city="Philadelphia";
UITreeNodeLabel="PA";
UITreeNodeStatus="Minor";
```

```
MyObject3=NewObject();
MyObject3.UITreeNodeId=2;
MyObject3.UITreeNodeParent=1;
MyObject3.country="England";
MyObject3.city="London";
UITreeNodeLabel="LN";
UITreeNodeStatus="Warning";
```

```
MyArrayOfObjects={MyObject1,MyObject2,MyObject3};
```

4. Define the user output parameters for the array of objects in the policy.

Table 58. User output parameters for MyObjArray

Field	Entry
Name	MyObjArray
Policy Variable Name	MyArrayOfObjects
Format	Array of Impact Object

5. Create a page in the console.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter Page for Tree and Topology in the **Page Name** field.
 - d. To save the page, click **Ok**.
6. Create a tree and a topology widget in the console.

- a. Open the **Page for Tree and Topology** page that you created.
- b. Drag the **Tree** widget into the content area.
- c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset** window is displayed.
- d. Select the dataset. Select the **MyObjArray** data type that belongs to the **TestTreeTopoPolicy** data source. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
- e. The **Visualization Settings** window is displayed. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
- f. To save the widget, click the **Save and exit** button on the toolbar
- g. Drag the **Topology** widget into the content area.
- h. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset** window is displayed.
- i. Select the dataset. Select the **MyObjArray** data type that belongs to the **TestTreeTopoPolicy** data source. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
- j. The **Visualization Settings** window is displayed. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
- k. To save the widget, click the **Save and exit** button on the toolbar.

Results

The data from the UI data provider is displayed in a hierarchy in the console alongside the status.

Filtering data output by a policy in the console

If you visualize data from a policy that contains runtime parameters, you can use the runtime parameters to filter the values that are displayed in the console.

Procedure

1. Create a policy that contains runtime parameters.
In this example, create a policy that is called **TestKey** that contains the **GetByKey** policy function. The runtime parameter is the **Key** parameter.

```

DataType = "Node";
MaxNum = 1;
MyCustomers = GetByKey(DataType, Key, MaxNum);

```
2. Create the **Key** user runtime parameter. In the policy editor, click the **Configure User Parameters** icon to create the user runtime parameter.

Table 59. Node user runtime parameter

Field	Entry
Name	Node
Policy Variable Name	Key
Format	String

3. Create a page
 - Open the console.
 - To create a page, click **Settings > New Page**.
 - Enter **Page for Bar Chart** in the **Page Name** field.

- Save the page.
- 4. Create a bar chart widget.
 - a. Open the **Page for Bar Chart** page that you created.
 - b. Drag the **Bar Chart** widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **DataFromPolicy** datatype that belongs to the **TestKey** data source. The dataset information is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed.
 - f. The runtime parameters that are available for the policy are listed under **Configure Optional Dataset**. You can enter the values that you want to filter for here. In this example, the system displays a field that is called **Key**. If you enter a value here, for example, R12345, only the data from the rows that contain the key field value R12345 is displayed. In this way, you can filter the values for the runtime parameters in the console.
 - g. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
 - h. To save the widget, click the **Save and exit** button on the toolbar.

Passing parameter values from a widget to a policy

You can use widgets to pass values as runtime parameters to policies in Netcool/Impact.

About this task

In this example, you want to be able to send an email that contains contextual information from a table widget. The information is passed as a runtime parameter value from the widget in the console to the policy in Netcool/Impact.

This example uses the wire widget. You may experience problems with dragging and dropping the widget. This is a known issue with the Tivoli Integrated Portal. For more information, see <http://www-01.ibm.com/support/docview.wss?uid=swg21626092>.

Procedure

1. Create a policy. For example, create the following policy that is called **DB2Emailpolicy** that retrieves values for some rows in a DB2 table and sends these values to an email address:


```
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);
```
2. Create a page in the console.
 - a. Open the console.
 - b. Click **Settings > Page**.
 - c. Enter Pageforbutton in the **Page Name** field.
 - d. To save the page, click **Ok**.
3. Create a table widget that visualizes data from the DB2 database.

- a. Open the **Pageforbutton** page.
- b. Drag the **Table** widget into the content area.
- c. To configure the widget data, click the down arrow icon and click **Edit**.
- d. Select the dataset. Use the search box or the **Show All** button to find the dataset that represents the DB2 database table.
- e. To save the widget, click the **Save** button.
4. Create a button widget. Enter the name of the button and specify the parameter that you would like to display in the console UI.
 - a. Drag the button widget that you created into the content area.
 - b. Enter a name. In this example, enter **Emailbutton**.
 - c. Select the dataset. In this example, Use the search box or the **Show All** button to find the dataset that represents the policy that you created and select it.
 - d. To ensure that the policy is run when the user clicks the button, select the **executePolicy** check box.
 - e. To save the button widget, click **Save**.
5. Create a wire.
 - a. Click the **Show Wire** icon and click the **New Wire** button.
 - b. Select the table widget as the source event for the new wire. Select **Table > NodeClickOn**.
 - c. Select the target for the wire. Select the button widget that you created.
 - d. Select **None** for the transformation.
 - e. To save the wire, click **Ok** and click the **Save** button on the toolbar on the page.

Results

After you complete this task, the **Send** button is displayed on the console. When a user clicks the **Send** button, the information that is contained in the row in the table is sent as an email to the address specified by the policy.

Passing parameter values from a table to a gauge

This example demonstrates how you can use Netcool/Impact policies to pass variables as runtime parameters from a widget to a policy and on to another widget.

About this task

In this example, users want to select rows in a table and display the status for the row in a gauge widget. You create 2 Netcool/Impact policies to facilitate this. One policy is the publisher policy and provides data in an output parameter to the second policy, the subscriber policy. The subscriber policy receives data from the publisher policy in a policy runtime parameter and it outputs the results as an output parameter. The data contained in the output parameter is then visualized as a gauge in the console.

The publisher policy retrieves the **SiteStatus** data from the **ObjectServer**. The subscriber policy retrieves related data from the **DB2** database.

Procedure

1. Create the publisher policy.

The following policy is called PolicyEventingPublisher and uses the DirectSQL policy function to retrieve data from the defaultobjectserver data source. You need to create a new integer field called SiteStatus in the ObjectServer if you have not done so already.

```
Log("Policies Eventing From OS...");
DataFromOS=DirectSQL('defaultobjectserver',
"SELECT SiteStatus,Node,Identifier from alerts.status",false);
```

Create the output parameter so that the widget can visualize the data.

Table 60. Output parameter for DatafromOS

Field	Entry
Name	DatafromOS
Policy Variable Name	DatafromOS
Format	DirectSQL / UI provider datatype

Create the custom schema values for the fields that you want to display in the console. You need to create 3 custom schema values. You also need to select the **Key Field** checkbox for the SiteStatus value.

Table 61. Custom schema value for SiteStatus

Field	Entry
Name	SiteStatus
Format	Integer

Table 62. Custom schema value for Node

Field	Entry
Name	Node
Format	String

Table 63. Custom schema value for Identifier

Field	Entry
Name	Identifier
Format	String

2. Create the subscriber policy.

The following policy is called PolicyEventingSubscriber and uses the GetByFilter function to retrieve the value for SiteStatus that is output by the publisher policy. The publisher policy retrieves the SiteStatus data from the ObjectServer. The subscriber policy retrieves related data from the DB2 database.

```
Log("Demo Policies Eventing From DB2...");
Filter="SiteStatus="+ SiteStatus;
DataFromDB2=GetByFilter('MachineInfo',Filter,false);
Log(DataFromDB2);
```

Create the policy runtime parameter.

Table 64. SiteStatus runtime parameter

Field	Entry
Name	SiteStatus
Policy Variable Name	SiteStatus

Table 64. SiteStatus runtime parameter (continued)

Field	Entry
Format	DirectSQL / UI provider datatype

Create the policy output parameter.

Table 65. DatafromDB2 output parameter

Field	Entry
Name	DatafromDB2
Policy Variable Name	DatafromDB2
Format	Datatype
Data Source Name	DB2Source
Data Type Name	MachineInfo

3. Create a page in the console.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter PageforMachineInfo in the **Page Name** field.
 - d. Save the page.
4. Create a table widget.
 - a. Open the **PageforMachineInfo** page that you created.
 - b. Drag the **Table** widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the **DatafromOS** datatype that belongs to the **PolicyEventingPublisher** datasource. The datatype is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed. The system displays all the available columns by default. You can change the displayed columns in the **Visualization Settings** section of the UI. You can also select the row selection and row selection type options.
 - f. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.
 - g. To save the widget, click the **Save and exit** button on the toolbar.
5. Create a gauge widget.
 - a. Open the **PageforMachineInfo** page that you created.
 - b. Drag the **Analog Gauge** widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
 - d. Select the dataset. Select the datatype **DatafromDB2** that belongs to the **PolicyEventingSubscriber** datasource. The datatype is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The **Visualization Settings** UI is displayed. Select the value that you want to display in the gauge in the **Value** field. In this example, you select **SiteStatus** from the list. You can also select a number of other optional values for the gauge such as minimum value, maximum value and unit of measure.
 - f. To ensure that the policy runs when the widget is displayed, select the **executePolicy** check box. Click **Ok**.

- g. To save the widget, click the **Save and exit** button on the toolbar.

Results

When a user selects a row in the table widget, the SiteStatus is displayed in the gauge widget.

Visualizing a data mashup from two IBM Tivoli Monitoring sources

You can use Netcool/Impact to visualize data from two different sources in IBM Tivoli Monitoring.

Before you begin

Find the configuration details for each UI data provider that you want to use. For example, if you want to connect to an IBM Tivoli Monitoring system, you must retrieve the following information for the Tivoli Enterprise Monitoring Server UI data provider:

- User
- Password
- Port
- Base URL

About this task

Tip: You can also use this procedure to obtain data from Tivoli Monitoring 6.3 for event management purposes, the only difference is that you omit the step that describes how to create the user output parameters for the policies.

Procedure

1. Use the UI data provider DSA to create two data sources. Each data source connects to a different IBM Tivoli Monitoring system. For example, create the following data sources:
 - Data source 1 retrieves data from ITM1
 - Data source 2 retrieves data from ITM2
 - a. Create the data source that retrieves data from ITM1:
 - 1) Enter ITM_DS1 in the **Data Source Name** field.
 - 2) Enter the user name and password for the database.
 - 3) Complete the other fields as required.
 - 4) Save the data source.
 - b. Create the data source that retrieves data from ITM2:
 - 1) Enter ITM_DS2 in the **Data Source Name** field.
 - 2) Enter the user name and password for the database.
 - 3) Complete the other fields as required.
 - 4) Save the data source.

For more information about how to create a data source for the UI data provider DSA, see the section about creating a UI data provider data source in the Netcool/Impact DSA Guide.

2. Create two data types for each data source that you created in the previous step. Later, you combine the data from the two data types that belong to the

same data source into a single object. Then, you combine the data from the two objects so that the data from the different systems is merged. For example, create the following data types:

- Datatype1A - select Tivoli Enterprise Monitoring Agent
 - Datatype1B - select Tivoli Enterprise Monitoring Agent
 - Datatype2A - select Tivoli Enterprise Monitoring Agent
 - Datatype2B - select Tivoli Enterprise Monitoring Agent
- a. Create the data types as follows, changing the name for each data type:
- 1) Enter Datatype1A as the name and complete the required fields.
 - 2) To enable the data type, select the **Enabled** check box.
 - 3) Select the key fields for the data type.
 - 4) Save the data type.

For more information about how to create a data type for the UI data provider DSA, see the section about creating a UI data provider data type in the Netcool/Impact DSA Guide.

3. To combine the data from the different sources, create a policy in Netcool/Impact that uses the `GetByFilter` function. For this example, you must create the following arrays to combine the data from the different sources:
- Array1A = `GetByFilter()`
 - Array1B = `GetByFilter()`
 - Array2A = `GetByFilter()`
 - Array2B = `GetByFilter()`

For example, the following policy uses the `GetByFilter` function to combine the data from the ITM_DS1 data source into a single object.

- a. The output parameter of the policy is `cpuLinuxITM={}`;
- b. Datatype1A retrieves data from a Tivoli Enterprise Monitoring Agent and it also retrieves the IP address data for each node:

```
ipaddress01="";
DataType="datatype1A";
Filter="&param_SourceToken=paramValue";

iparray=GetByFilter(DataType, Filter, false);
count=0;
while(count<Length(iparray)){
    if( (iparray[count].IPVERSION != "IPv6")&&(iparray[count].
IPADDRESS!="127.0.0.1")){
        ipaddress01= iparray[count].IPADDRESS;
    }
    count = count +1;
}
```

- c. Datatype1B retrieves data from a Tivoli Enterprise Monitoring Agent and it also provides processor usage data for each node. The policy creates an array of metrics for each monitored node. It also enhances this information with the IP address:

```
DataType="datatype1B";
Filter="&param_SourceToken=paramValue&sort=BUSYCPU";
MyFilteredItems = GetByFilter( DataType, Filter, false );

index = 0;
if(Num > index){
    while(index<Num){
        cpu=NewObject();
        cpu.TIMESTAMP= MyFilteredItems[index].TIMESTAMP;
        cpu.ORIGINNODE= MyFilteredItems[index].ORIGINNODE;
```

```

        cpu.BUSYCPU= MyFilteredItems[index].BUSYCPU;
        cpu.IPADDRESS=ipaddress01;
        cpuLinuxITM = cpuLinuxITM+{cpu};
        index=index+1;
    }
}

```

Log(" Finished collecting cpu usage from Metrics Agent :" + DataType);

For more information about using the GetByFilter function with the UI data provider, see the topic about accessing data types output by the GetByFilter Function in the Netcool/Impact Solutions Guide.

4. Create the user output parameters for the policies. In this example, cpuLinuxITM is the output parameter that is defined in the policy. You must create an output parameter for cpuLinuxITM as outlined in the table. To create a user output parameter, open the policy editor and click the **Configure User Parameters** icon and the **Policy output parameter:New** button.

Table 66. ITM1 output parameter

Field	Entry
Name	ITM1
Policy Variable Name	cpuLinuxITM
Format	Array of Impact Objects

This parameter ensures that the policy is exposed as part of the Netcool/Impact UI data provider.

For more information about how to configure user parameters, see the topic about how to configure user parameters in the Netcool/Impact Solutions Guide.

5. Create a data source and data type that are based on the policy. In this example, create the data source as follows:
 - a. Select **ITM_mashup_policy** from the list in the **Data Source Name** field.
 - b. Enter the user name and password for the database.
 - c. Select the Netcool/Impact UI data provider, **Impact_NCICLUSTER**, as the provider.
 - d. Complete the other fields as required.
 - e. Save the data source.

Create the data type as follows:

- a. Enter ITM_mashup_dt as the name and complete the required fields.
 - b. To ensure that the data type is compatible with the UI data provider, select the **UI data provider: enabled** check box.
 - c. Select the key fields for the data type.
 - d. Save the data type.
6. To confirm that the policy returns the correct data when it runs, right click the data type and select **View Data Items**. Enter &executePolicy=true in the filter and refresh.
7. Create a page in the console.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter Page for ITM mashup in the **Page Name** field.
 - d. Save the page.
8. Create a table widget that visualizes data from the policy's data type.

- a. Open the **Page for ITM mashup** page that you created.
- b. Drag the **Table** widget into the content area.
- c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The **Select a dataset window** is displayed.
- d. Select the dataset. Select the **ITM_mashup_dt** data type that belongs to the **ITM_mashup_policy** data source. The data type is only displayed after the defined refresh interval. The default is 5 minutes.
- e. The **Visualization Settings UI** is displayed. Enter the values that you want to use. You can select multiple lines. You can also select a text to display as a tooltip. Click **Ok**.
- f. To save the widget, click the **Save and exit** button on the toolbar.

Results

After you create the table widget, the same data that was displayed in step 6 in Netcool/Impact is displayed in the console.

Visualizing data from the Netcool/Impact self service dashboards

Netcool/Impact self service dashboard widgets are designed to enable users or administrators to create dashboards. The self service dashboards are able to accept user input through a customizable Input Form widget and can drive user actions through a Button widget. Both of these widgets interact with Netcool/Impact through the Netcool Rest API Interface.

Netcool/Impact 6.1.1.4 has a UI Data provider which makes Netcool/Impact data available for consumption by dashboard widgets in the IBM Dashboard Applications Services Hub. The UI Data provider works well for dashboards that are intended for read-only usage but cannot interact dynamically with Netcool/Impact policies. For example, you can create a dashboard with a table widget which displays all the trouble tickets that are managed by Netcool/Impact. However you cannot interact with the trouble tickets to create a ticket.

When the Netcool/Impact Self Service dashboard widgets are installed on the console, you have the option of adding to the existing visualizations. You can update the data through the execution of Netcool/Impact policies and to take certain actions on a data set.

Installing the Netcool/Impact Self Service Dashboard widgets

To create custom dashboards for Netcool/Impact to view in Jazz™ for Service Management, you can add Netcool/Impact specific widgets to enhance the capabilities of the dashboards you create.

Before you begin

Before you install the Netcool/Impact **Impact_SSD_Dashlet.war** file on the Dashboard Application Services Hub Server. You must have the following environment.

- A server with Netcool/Impact 6.1.1.4 installed.
- A server with IBM Dashboard Applications Services Hub installed and configured with a data connection to the Netcool/Impact 6.1.1.4 server. For information about setting up a connection between the Impact server and Jazz for Service Management, see “Setting up the remote connection between the UI data provider and the console” on page 164.

For more information about Jazz for Service Management, see the Jazz for Service Management infocenter available from the following URL: http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/index.jsp?topic=%2Fcom.ibm.psc.doc_1.1.0%2Fpsc_ic-homepage.html.

Procedure

1. Log on to the Impact server.
2. Navigate to the add-ons directory, %IMPACT%/add-ons/ssd.
3. Copy the **Impact_SSD_Dashlet.war** file over to the Dashboard Application Services Hub Server. Note the location where you download the file to. For example, C:\build\Impact_SSD_Dashlet.war.
4. On the Dashboard Application Services Hub Server, run the wsadmin tool by using one of the following commands:
 - UNIX: %INSTALL%/JazzSM/profile/bin/wsadmin.sh
 - Windows: %INSTALL%/JazzSM/profile/bin/wsadmin.bat

Where %INSTALL% is the installed location of Jazz for Service Management.

5. Run the following command all on one line to install the **Impact_SSD_Dashlet.war** file.

```
$AdminApp update isc modulefile
{-operation addupdate -contents "<ImpactSSDWar>"
-custom paavalidation=true
-contenturi Impact_SSD_Dashlet.war
-usedefaultbindings
-contextroot /Impact_SSD_Dashlet
-MapWebModToVH {{.* .* default_host}}}
```

Where <ImpactSSDWar> is the location of the copied war file. For example, C:\build\Impact_SSD_Dashlet.war.

6. If the wsadmin command succeeds without any errors, use the following command to save the changes:

```
$AdminConfig save
```

7. Use one of the following commands to restart the Dashboard Application Services Hub Server:

- UNIX:

```
%INSTALL%/JazzSM/profile/bin/stopServer.sh server1
%INSTALL%/JazzSM/profile/bin/startServer.sh server1
```
- Windows:

```
%INSTALL%/JazzSM/profile/bin/stopServer.bat server1
%INSTALL%/JazzSM/profile/bin/startServer.bat server1
```

Uninstalling the Netcool/Impact Self Service Dashboard widgets

How to uninstall the self service dashboard widgets feature from Jazz for Service Management.

Before you begin

You must remove any dependencies on the Netcool/Impact self service dashboard widgets from any existing pages or portlets in Jazz for Service Management. Then, delete any instances of the Netcool/Impact dashboard widgets from the page or portlets in Jazz for Service Management. For information about how to delete pages or portlets see the *IBM® Dashboard Application Services Hub online help*.

Procedure

1. On the Dashboard Application Services Hub Server, use one of the following commands to run the wsadmin tool:
 - UNIX: `%INSTALL%/JazzSM/profile/bin/wsadmin.sh`
 - Windows: `%INSTALL%/JazzSM/profile/bin/wsadmin.bat`
2. Run the following command to uninstall the **Impact_SSD_Dashlet.war** file.

```
$AdminApp update isc modulefile  
{-operation delete -contenturi Impact_SSD_Dashlet.war }
```
3. If the wsadmin command succeeds without errors, run the following command to save the changes:

```
$AdminConfig save
```
4. Use one of the following commands to restart the Dashboard Application Services Hub Server.
 - UNIX:


```
%INSTALL%/JazzSM/profile/bin/stopServer.sh server1  
%INSTALL%/JazzSM/profile/bin/startServer.sh server1
```
 - Windows:

```
%INSTALL%/JazzSM/profile/bin/stopServer.bat server1  
%INSTALL%/JazzSM/profile/bin/startServer.bat server1
```

Editing an Input Form widget

The Input Form widget is a form control which can run Netcool/Impact policies with user-defined runtime parameters. The Input Form widget dynamically generates a form with a set of input fields which correspond to the runtime parameters of the policy.


When you submit the form, the associated policy is run with runtime parameters from the form input fields.

1. In the title bar, click the **Edit options** icon  and select **Edit**.
2. Choose the Netcool/Impact policy that you want to run. You can search for the policy by completing the search field with either a full or partial name and clicking **Search**. You can also view the complete list of available data sets by clicking the **Show All** button in the center of the results panel.
3. Select that data set, and click the right arrow to show the **Visualization Settings** page. In the **Visualization Settings** page, you can configure the button title and Netcool/Impact policy parameters.
4. In the **Required Settings** section, select the **executePolicy** option.
5. Optional. Select **Optional Settings**, add the name of the form to the **Title** field. The title is also used for the label of the form submission button.
6. Optional. **Configure Optional Dataset Parameters:** You can also set default values for any runtime parameters that are attached to the Netcool/Impact policy. The Input Form widget populates the runtime parameter form with any default values you set here.
7. Click **OK** to implement the changes, or **Cancel** to discard the changes.
8. On the dashboard, click the button on the widget to run the policy. The runtime parameters are passed to the policy as policy runtime parameters. The results are displayed in the widget.

Tip: In the Input Form, you can manually change the values in the form fields, click the button, and run the policy again and show the results in the dashboard.

Editing a Button widget

The Button widget can be used in an Operator View to run a policy. You can edit the Button widget to use the runtime parameters that are set by the policy that is attached to the button.

1. In the title bar, click the **Edit options** icon  and select **Edit**.
2. Choose the Netcool/Impact policy that you want to run. You can search for the policy by completing the search field with either a full or partial name and clicking **Search**. You can also view the complete list of available data sets by clicking the **Show All** button in the center of the results panel.
3. Select that data set, and click the right arrow to show the **Visualization Settings** page. In the **Visualization Settings** page, you can configure the button title and Netcool/Impact policy parameters.
4. In the **Required Settings** section, select the **executePolicy** option.
5. Optional. Select **Optional Settings**, add the name of the button to the **Title** field. The title is also used for the label of the form submission button.
6. Optional. **Configure Optional Dataset Parameters:** You can also set default values for any runtime parameters that are attached to the Netcool/Impact policy.
7. Click **OK** to implement the changes, or **Cancel** to discard the changes.
8. On the dashboard, click the button on the widget to run the policy. The runtime parameters are passed to the policy as policy runtime parameters. The results are displayed in the widget.

Tip: To change the values of the runtime parameters, you must edit the Button widget and change the parameters in the **Visualization Settings** page before you run the policy again from the dashboard.

Configuring the Button widget to receive data from other widgets

The Button widget can receive data from other widgets. For example, in the console you can create a wire between the Button widget and a table widget. Wires are connections between widgets to share information and for opening pages in context. When you click a row in the Table widget, the Button widget processes the data. When you click the button on the Button widget, that data from the table row is processed. The data is then sent on to the policy as runtime parameters.

About this task

This example uses a Button widget and a table widget. You can use the same process with a Button widget and any other widget.

Procedure

1. In Netcool/Impact, create a DB2 data source.
 - a. Enter NewDataSource in the **Data Source Name** field.
 - b. Enter the user name and password for the database.
 - c. Complete the other fields as required.
 - d. Save the data source.
2. Create a data type for the DB2 data source.
 - a. Enter NewDataType as the name and complete the required fields.
 - b. To ensure that the data type is compatible with the UI data provider, select the **UI data provider: enabled** check box.
 - c. Select the key fields for the data type.

- d. Save the data type.
3. Create the target policy.
 - a. In the Netcool/Impact policy editor, create a Netcool/Impact policy.
 - b. Define the following policy:


```
Log ("Executing IPL Impact Policy");
filter = "SERVICEREQUESTIDENTIFIER = " + inputParamID;
GetbyFilter ('NewDataType', filter, false);
```
 - c. Save the policy, name it PolicyForButtonWidget.
4. Create the runtime parameter for the policy.
 - a. In the policy editor, click the **Configure User Parameters** icon to create the runtime parameter.
 - b. Name the runtime parameter inputParamID.
 - c. Specify Long for the **Format** field.
 - d. Click **OK**.
 - e. Save the policy.
5. Create a page in the console.
 - a. Open the console.
 - b. To create a page, click **Settings > New Page**.
 - c. Enter a name for the page in the **Page Name** field.
 - d. Save the page.
6. Create a Button widget in the console.
 - a. Open the new page that you created.
 - b. Open the **Impact** widget folder, drag the Button widget into the content area.
 - c. To configure the widget data, click the down arrow icon and click **Edit**. The Select a data setwindow is displayed.
 - d. Select the data set. Select the **PolicyForButtonWidget** policy that you created earlier.
 - e. The Visualization Settings UI is displayed. Click **OK**.
 - f. To save the button widget, click the **Save** button on the page toolbar.
7. Create a Table widget in the console.
 - a. Open the new page that you created.
 - b. Drag the Table widget into the content area.
 - c. To configure the widget data, click it. Click the down arrow icon and click **Edit**. The Select a dataset window is displayed.
 - d. Select the data set. Select the **NewDataType** data type that belongs to the **NewDataSource** data source. The data type is only displayed after the defined refresh interval. The default is 5 minutes.
 - e. The Visualization Settings UI is displayed. Click **OK**.
 - f. To save the Table widget, click **Save** on the page toolbar.
8. Create a Wire between the Button and Table widgets.
 - a. To open the Wires wizard, click the **Show Wires** button on the page toolbar.
 - b. Click the **New Wire** button.
 - c. On the Select Source Event for New Wire page, identify the Table widget in the list of **Available source events** and select the **NodeClickedOn** event.
 - d. Click **OK**.

- e. On the Select Target for New Wire page, select the **Button** widget from the list of **Available targets**.
- f. Click **OK** on the next two pages to create a wire between the Button and Table widgets.

Results

When you click a row in the Table widget, a NodeClickedOn event is generated. The Button widget processes the event by extracting the data for the clicked table row. When you click the button on the Button widget, it runs the policy that is configured in the widget. The data passes from the table row to the policy as policy runtime parameters. The GetByFilter function runs and uses the runtime parameter that is provided by the table row.

Reference topics

You can use custom URLs and the UI data provider to access data directly. You can also customize the UI data provider and enable large data model support.

Large data model support for the UI data provider

You can use the UI data provider with large data models based on the supported databases.

Large data model support is used to facilitate the integration of the UI data provider with database tables that use filtering and paging to limit the number of rows.

The following databases and associated data types are supported and large data model integration is enabled by default:

- DB2
- Derby
- HSQLDB
- Informix
- MySQL
- MS-SQLServer
- Oracle
- PostgreSQL

For information about how to enable and disable large data models see “Disabling and enabling large data models” on page 206.

Restrictions

- Sybase databases and associated data types are not supported. You can limit the data by using the input filters in the data type or in the widget.

Important: If you access a Netcool/OMNIBus or Sybase data type which has a huge number of rows, for example more than 10,000 rows you can potentially run out of memory. Out of memory issues can occur because all the rows of data that are fetched from the database use the available heap memory that is allocated by the Java virtual machine. If you plan to access large amounts of data for a Netcool/OMNIBus or Sybase data type, consider increasing the heap memory settings for the TIPProfile and ImpactProfile from their default values.

For information about changing heap memory settings, see the *Memory status monitoring* section of the *Administration Guide*.

- If you want to integrate the UI data provider with large data models, you must not use UIDPROWNUM as a field name in the Oracle or DB2 database. The AS UIDPROWNUM field is added to the query for Oracle and DB2 databases. As a result, this field is reserved for use by the query that is associated with these types of databases.
- The MS-SQLServer database uses the first field of the data type in the query. There are no reserved fields for this type of database.
- If pagination is enabled in Tivoli Integrated Portal or in the URL, Netcool/Impact does not store this information in the memory. Instead, Netcool/Impact retrieves the rows directly from the database to avoid any adverse effects on performance and memory.

Disabling and enabling large data models

You can enable and disable the UI data provider so that it is not compatible with large data models.

About this task

Large data model integration is enabled by default for the following databases:

- DB2
- Derby
- HSQLDB
- Informix®
- MySQL
- MS-SQLServer
- Oracle
- PostgreSQL

Procedure

1. To disable the integration of the UI data provider and large data models, change `impact.uidataprowner.largetablemodel=true` to `impact.uidataprowner.largetablemodel=false` in the `server.props` file.

Tip: If this parameter does not exist, you can add it to the `server.props` file.

2. Restart the GUI Server.

Enabling and disabling the large data model for the Objectserver:

Large data model integration is disabled by default for the ObjectServer and is not supported unless you use Netcool/OMNIBus version 7.4.0. fix pack 1. This fix pack has functions that can be used to perform paging in the ObjectServer.

Before you begin

Before you begin this task, you must stop the GUI Server.

On UNIX operating systems, enter the following command in the command prompt:

```
$IMPACT_HOME/bin/ewasGUIStartStop.sh stop  
[-username adminuser -password adminpassword]
```

On Windows operating systems, you use the Services Extension in the Microsoft Management Console. In the Services Extension window right-click **Tivoli Integrated Portal**, and select **Properties**. In the **Properties** dialog box, click **Stop**, and then click **OK**.

About this task

For information about Netcool/OMNIbus version 7.4.0. fix pack 1, see http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIbus.doc_7.4.0/omnibus/wip/welcome.htm.

Procedure

1. To enable the integration of the UI data provider and large data models for the ObjectServer, change the property `impact.uidataprovider.largetablemodel.objectserver=false` to `impact.uidataprovider.largetablemodel.objectserver=true` in the `server.props` file.

If the property does not exist, you can create it.

2. Restart the GUI Server.
 - On UNIX systems, enter the following command at the command prompt:

```
$IMPACT_HOME/bin/ewasGUIStartStop.sh start  
[-username adminuser -password adminpassword]
```
 - On Windows systems, you use the Services Extension in the Microsoft Management Console. In the Services Extension window right-click **Tivoli Integrated Portal**, and select **Properties**. In the **Properties** dialog box, click **Start**, and then click **OK**.

Tip: You can also use the `startServer.sh` script in the `$TIP_HOME/profiles/TIPProfile/bin` directory.

UI data provider customization

After you enable the UI data provider, you can customize it by changing the refresh rate, initializing all SQL data items, and enabling multiple Netcool/Impact clusters that access the same Tivoli Integrated Portal provider.

Refresh rate

You can configure how often the UI data provider is refreshed. This interval is set to 5 minutes. To change this setting, add the following statement to the `server.props` file that is in the `IMPACT_HOME/etc/` folder:

```
impact.uidataprovider.refreshrate=<refresh_rate_in_milliseconds>
```

For example, add the following statement to change the refresh interval to 3 minutes:

```
impact.uidataprovider.refreshrate=180000
```

Initialization of SQL data items

You can configure Netcool/Impact so that the SQL data items are initialized during startup by default. To do so, add the following to the `server.props` file:

```
impact.uidataprovider.sql.initializenodes=true
```

Restriction: This setting can have an adverse affect on performance and memory usage. This restriction depends on the amount of data that is held by the data type. This setting is disabled by default for this reason.

Enable multiple servers

Your deployment may include multiple UI data provider servers in a server cluster that access the same Tivoli Integrated Portal provider. To integrate the UI data provider with this type of deployment, you must configure the navigational model load so that it regularly refreshes the data from each UI data provider server. To do so, add the following statement to the `server.props` file:

```
impact.uidataprovider.refreshclusters=true
```

Character encoding

By default, Netcool/Impact uses UTF-8 character encoding to parse parameter values and to send these values to the UI data provider. You change this setting if, for example, you want to use Chinese characters alongside the UI data provider. To change this setting, shut down your Tivoli Integrated Portal server and add the following statement to the `server.props` file that is in the `IMPACT_HOME/etc` folder:

```
impact.uidataprovider.encoding=<charset>
```

Start your Tivoli Integrated Portal server. Netcool/Impact uses the encoding that is defined in the `charset` variable to parse parameter values.

Disabling the UI data provider

By default, the UI data provider is enabled in the Tivoli Integrated Portal profile. To disable the UI data provider, add the following statement to the `server.props` file in the `IMPACT_HOME/etc` folder. You must shut down the GUI Server before you add the statement.

```
impact.uidataprovider.enable=false
```

Note: In a split installation, add the statement to the `server.props` file in the Tivoli Integrated Portal server.

To complete the change, restart the GUI Server.

Translating date filters for connected databases

Netcool/Impact must translate filter values from the console into a format that is compatible with the queries used for the various databases. The translation is required because the console uses milliseconds as the generic format to send dates. This translation is controlled by the `impact.uidataprovider.dateformat` property in the `server.props` file in the `IMPACT_HOME/etc` folder. The default pattern is `yyyy-MM-dd HH:mm:ss.SSS`. For example, if you filter for January 1st 2012, Netcool/Impact translates the filter value into `2012-01-01 00:00:00.000`.

To change the default pattern, change the `impact.uidataprovider.dateformat` property in the `server.props` file in the `IMPACT_HOME/etc` folder.

Connection Timeout

You can configure how long the UI data provider waits for a successful connection. This property is set in milliseconds and the default is set to 60 seconds. To change this setting, add the following statement to the `server.props` file that is in the `IMPACT_HOME/etc/` folder:

```
impact.uidataprovider.gethttp.conntimeout=<connection_timeout_in_miliseconds>
```

For example, add the following statement to change the connection timeout to 3 minutes:

```
impact.uidataprovider.gethttp.conntimeout=180000
```

Response Timeout

You can configure how long the UI data provider waits for a successful response. This property is set in milliseconds and the default is set to 100 seconds. To change this setting, add the following statement to the `server.props` file that is in the `IMPACT_HOME/etc/` folder:

```
impact.uidataprovider.gethttp.sotimeout=<response_timeout_in_miliseconds>
```

For example, add the following statement to change the response timeout to 3 minutes:

```
impact.uidataprovider.gethttp.sotimeout=180000
```

Accessing the Netcool/Impact UI data provider

You can use URL to access the UI data provider data provided by Netcool/Impact.

Procedure

Use the following URL to access the Netcool/ImpactUI data provider

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
providername
```

hostname is the machine where the Tivoli Integrated Portal is running.
port is the https port of the Tivoli Integrated Portal, the default value is 16311

Note:

The Netcool/ImpactUI data provider registers the name `Impact_NCICLUSTER` by default. If you registered another cluster name during installation, the UI data provider registers this name as `Impact_<clustername>`.

Example

For example, you can use the following URL to access the UI data provider:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER
```

Accessing data sources from a UI data provider

You can use a URL to access the data sources that Netcool/Impact provides when it is functioning as a UI data provider.

Procedure

Use the following URL to access the data sources:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/<datasourcename>
```

Example

If you configure the defaultobjectserver data source in Netcool/Impact to point to an Netcool/OMNIBus installation, you use the following URL to access it:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/defaultobjectserver
```

Accessing data sets from a UI data provider

You use a URL to access data sets provided by a UI data provider. The data sets provided by the UI data provider can be based on the SQL, Internal, or SNMP data types.

Procedure

Use the following URL to access data sets provided by the UI data provider:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/<datasourceId>/datasets
```

This URL returns the data sets that belong to a data source.

Use the following URL to access a specific data set:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/<datasourceId>/datasets/<datasetId>
```

Use the following URL to access the rows in the tables:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/<datasourceId>/datasets/<datasetId>/items
```

Note: If the key field in the table is a unique identifier, this URL returns all the rows. If the table contains key fields that are not unique identifiers, these rows are not returned.

Example

You use the following URL to access data sets from the defaultobjectserver data source:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/defaultobjectserver/datasets
```

You create the following data types for the defaultobjectserver data source:

- The ALERTS data type points to alerts.status
- The JOURNALS data type points to alerts.journals

You use the following URL to access the ALERTS data set:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/defaultobjectserver/datasets/ALERTS
```

You use the following URL to access item ID for the ALERTS data set:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/defaultobjectserver/datasets/ALERTS/items
```


When you create a data type in Netcool/Impact, you designate at least one field as the key field. In this example, you chose Identifier as the key field for the ALERTS data set. You use the following URL to access data where the identifier is Impact123:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/
datasources/defaultobjectserver/datasets/ALERTS/items/Impact123
```

You use the following URL to access the Serial and Severity properties for this event:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/
datasources/defaultobjectserver/datasets/ALERTS/items/
Impact123?properties=Serial,Severity
```

You use the following URL to access the data for all columns for the row where the identifier is Impact123:

```
https://example.com:16311/ibm/tivoli/rest/providers/Impact_NCICLUSTER/
datasources/defaultobjectserver/datasets/ALERTS/items/Impact123?properties=all
```

Known issues with JavaScript and the UI data provider

How to resolve known issues with user output parameters in a JavaScript policy and the UI data provider.

When you expose a user output parameter inside a JavaScript policy, the user output parameter data when queried from the UI data provider might be blank. To resolve this issue, certain objects inside the JavaScript policy must be deleted, including JavaScript functions and Java objects, typically at the end of the function or policy.

For example, the following JavaScript function must be deleted before the user output parameters are successfully returned from the UI data provider.

```
function myFunction() {
//.....
}
delete myFunction;
```

Additionally, any Java objects that are created inside the JavaScript policy must be deleted as well. For example,

```
var myString = NewJavaObject("java.lang.String", ["myString"]);
//.....
delete myString;
```

If the Java object contains the information that is to be used in a user output parameter. The value of the Java object must be stored by using a Netcool/Impact policy function to convert the Java object to the correct variable type.

```
var myString = NewJavaObject("java.lang.String", ["myString"]);
//.....
var outputString = String(myString);
delete myString;
```

If the variable is returned from an invocation of a function or call, you must delete that object as well.

Running policies and accessing output parameters

You can use a URL to run a policy and to make the output parameters of that policy such as variables, objects, or variables output by the `GetByFilter` function available to the UI data provider.

Procedure

To run a policy and make the output parameters available to the UI data provider, add `executePolicy=true` to the following URL:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/Impact_NCICLUSTER/  
datasources/IMPACT_POLICY_<polycyname>/datasets/  
<polycyname>_policy_variables/items?executePolicy=true
```

Example

You can use a URL to run a policy and to make the output parameters available to the UI data provider. You create a policy called `Test_Policy`.

You add `executePolicy=true` to the following URL to run the `Test_Policy` policy and make the output parameters available to the UI data provider:

```
https://example.com:16311/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER/datasources/IMPACT_POLICY_Test_Policy/  
datasets/Test_Policy_policy_variables/items?executePolicy=true
```

UI data provider URLs

Use the following URLs to access Netcool/Impact data that has been made available to the UI data provider.

Access the UI data provider:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/<providername>
```

Access a data source:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER/datasources/<datasourcename>
```

Access all the data types that belong to a particular data source:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER/datasources/<datasourceId>/datasets
```

Access a specific data set:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER/datasources/<datasourceId>/datasets/<datasetId>
```

Access the rows in a data base table

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER/datasources/<datasourceId>/datasets/<datasetId>/  
items
```

Run a policy and make the output parameters available to the UI data provider:

```
https://<hostname>:<port>/ibm/tivoli/rest/providers/  
Impact_NCICLUSTER/datasources/IMPACT_POLICY_<polycyname>/datasets/  
<polycyname>_policy_variables/items?executePolicy=true
```

Chapter 15. Working with OSLC for Netcool/Impact

You can use Open Services for Lifecycle Collaboration (OSLC) for Netcool/Impact to integrate Netcool/Impact with other OSLC providers and clients. Netcool/Impact also functions as a client of OSLC data. You can use these capabilities to integrate Netcool/Impact with compatible products and data.

Netcool/Impact 6.1.1.4 contains an implementation of the Open Services for Lifecycle Collaboration (OSLC) Core Specification version 2.0. For more information about OSLC, see the OSLC Core Specification (<http://open-services.net/bin/view/Main/OslcCoreSpecification>).

Netcool/Impact does not support delegated UI dialogs or creation factories. Netcool/Impact supports only the RDF/XML representation of OSLC and the following aspects of the OSLC Core Specification v2:

- OSLC Service Provider
- OSLC Query Capability
- OSLC Resource Shape
- OSLC Resource

Usage Scenarios

Netcool/Impact is able to act as an OSLC provider and an OSLC client. You can use Netcool/Impact as a generic OSLC adapter for other OSLC and non-OSLC service providers.

Response Formats

Netcool/Impact uses the RDF/XML format for all OSLC responses, as required by the OSLC Core Specification v2.

Important: When viewed in some web browsers, such as Mozilla Firefox, the raw RDF/XML is automatically translated into the abbreviated RDF/XML format, which omits blank nodes. For more information, see the RDF/XML Syntax Specification (<http://www.w3.org/TR/REC-rdf-syntax/#section-Syntax-blank-nodes>)

The raw RDF/XML and the abbreviated version are semantically identical. You can use Internet Explorer, Mozilla Firefox, the Netcool/Impact GetHTTP function, or the Linux `curl` utility to retrieve the raw XML/RDF.

Note: If you are working with the Netcool/Impact GUI in the Mozilla Firefox browser and you simultaneously open a second instance to view an OSLC URL, the system logs you out of the first instance. To prevent this problem, you must create a second profile in Mozilla Firefox for viewing OSLC URLs. For more information about how to do so, see the help section about profiles on the Mozilla website (<http://support.mozilla.org/en-US/kb/profile-manager-create-and-remove-firefox-profiles>).

Jazz for Service Management

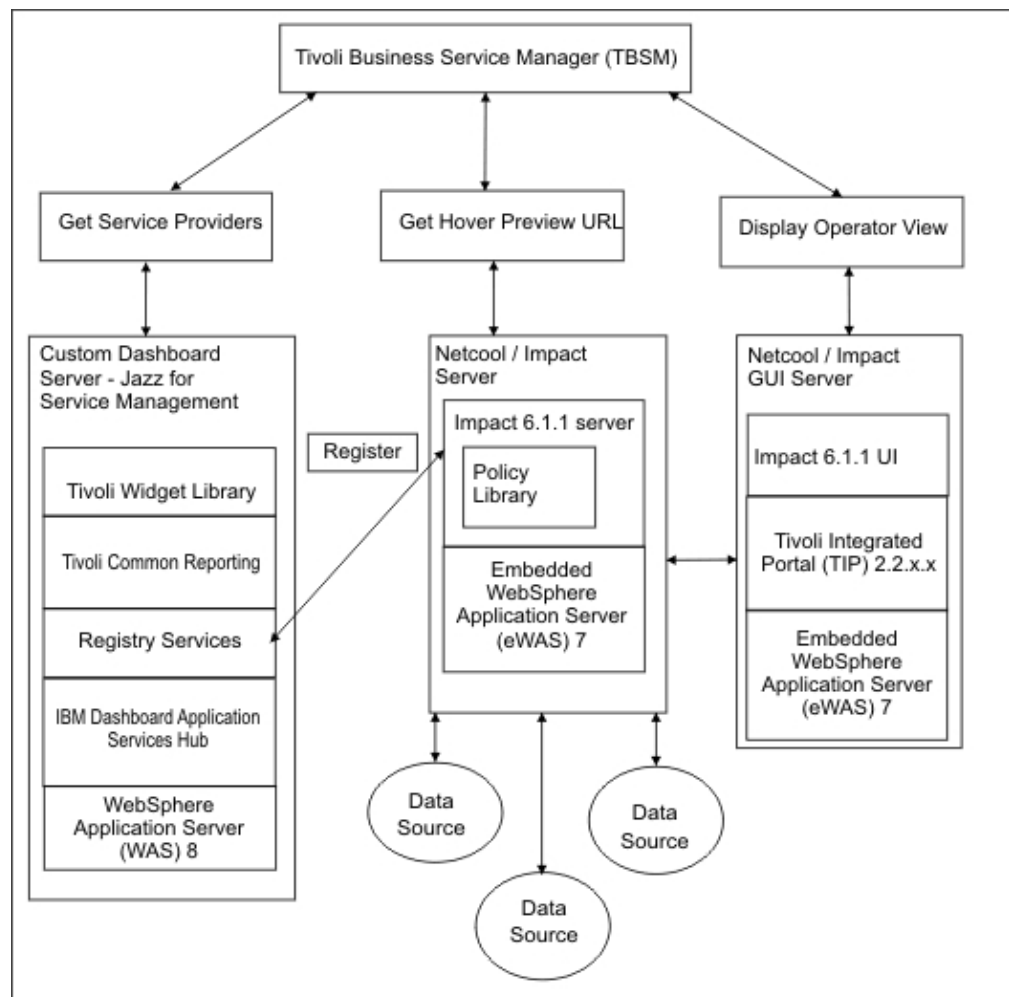
OSLC requires Netcool/Impact 6.1.1 or higher. OSLC also requires Jazz for Service Management, which is bundled with Netcool/Impact 6.1.1. You use the installer provided with Jazz for Service Management to install it separately.

Before you can use OSLC, you must install the Registry Service component of Jazz for Service Management. For more information, see http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.psc.doc_1.1.0/psc_ic-homepage.html

Introducing OSLC

Before you use OSLC for Netcool/Impact, read this information about the specifics of this implementation.

The following graphic outlines an example of a typical system architecture for OSLC:



The installation that is illustrated in the graphic shows how Netcool/Impact uses the registry services from Jazz for Service Management to provide hover preview support to TBSM. TBSM retrieves the list of service providers for a resource from the server where the Registry Services component of Jazz for Service Management is installed. TBSM connects to specified service provider in the backend of Netcool/Impact where the service provider was created. The backend is connected

to the data sources and policies that provide data and generate the hover preview window information, including the URL used to retrieve the hover preview content. The URL can be the frontend of Netcool/Impact which uses the operator views to render the actual content inside the hover preview window.

OSLC resources and identifiers

OSLC for Netcool/Impact follows the OSLC specifications with regards to OSLC resources and identifiers. However, there are some important things to consider before you start working on OSLC for Netcool/Impact.

Following the OSLC specification, the URIs generated by Netcool/Impact in OSLC documents are opaque. The only valid URIs are those URIs that are discovered through an OSLC service provider registry or from a previous OSLC document.

For example, if the URI for a resource that is called **Person** is **http://<server>/person** it cannot be assumed that **http://<server>/dog** is the correct URI for a resource called **Dog**. This document provides URIs as examples but these do not imply functioning URIs on any particular system.

All Netcool/Impact URIs use the **http://** or **https://** scheme.

As URIs are opaque, it follows that the **http://<server>/** resource and the **https://<server>/** resource are two different resources.

Although all URIs can use the **http** or **https** scheme, not all URIs can be resolved as an HTTP resource.

Where possible, the term URI is used to indicate identifiers, which may or may not resolve to a particular document, and URLs to refer to resources which resolve to a document.

You can use HTTP or HTTPS authentication for security. If you use HTTP basic authentication, the security credentials are available on the network as clear text. HTTPS is the preferred method as the security credentials are not available as clear text.

OSLC roles

You use the `bsmAdministrator` and `impactOSLCDataProviderUser` roles to regulate access to the OSLC service provider.

The `bsmAdministrator` role is assigned to your Netcool/Impact administrator, who is, in most cases, the Tivoli Integrated Portal administrator.

To add users to roles, use the `$IMPACT_HOME/bin/jython/mapRole.py` script. To add users or groups to the `impactOSLCDataProviderUser` role, use the following command:

```
wsadmin -lang jython -f mapRole.py -A NCI -r impactOSLCDataProviderUser  
-g "group1|group2|group3" -u "user1|user2|user3"
```

Note: You must use a single script to add the user to the role and to add the role to the group. If you use separate scripts, the first mapping is overwritten by the second.

If you use file based authentication for your users and groups, the users and groups that are assigned to the OSLC role are not replicated to the Impact Server

automatically. You must add these roles manually. To add the users and groups to the Impact Server, use the WebSphere Application Server administration console that is available at the following URL:

`https://<Impact_server_host>:9086/ibm/console/logon.jsp`

If you use the Object server or Lightweight Directory Access Protocol (LDAP) to authenticate your users and groups, the users and groups that are assigned to the OSLC role are replicated to the Impact Server automatically

If you change the current user registry, you must restart the Netcool/Impact profile WebSphere server.

Example

You use the following command to add the `oslcuser` user to the `impactOSLCDataProviderUser` role for a Linux operating system:

```
/opt/IBM/tivoli/tipv2/profiles/ImpactProfile/bin/wsadmin.sh -lang jython
-f /opt/IBM/tivoli/impact/bin/jython/mapRole.py
-A NCI -r impactOSLCDataProviderUser -u "tipadmin|oslcuser"
```

Working with data types and OSLC

You can use the following information to integrate the Netcool/Impact OSLC provider and data types.

You cannot use a display name that contains special characters with OSLC. You must enter a display name that does not contain special characters. To edit the display name:

1. Open Netcool/Impact and select **System Configuration > Event Automation > Data Model**.
2. Click the data source that the data type belongs to.
3. Select the row that contains the display name that uses special characters and click the **Edit Current Row** icon.
4. Replace the special characters in the display name and save your changes.

Accessing Netcool/Impact data types as OSLC resources

To allow Netcool/Impact to access a data type as an OSLC resource, you must add the data type to the `NCI_oslc.props` file in `<IMPACT_HOME>/etc/`.

About this task

In this scenario, the OSLC resources that are returned by a query are Netcool/Impact data items, which are rows from the underlying database.

Procedure

1. Add a property for each Netcool/Impact data type that you want to make available as OSLC resources to the `NCI_oslc.props` file in `<IMPACT_HOME>/etc/`. **NCI** is the default name of the Impact Server. You add the property in the following format:

```
oslc.data.<pathcomponent>=<datatype>
```

where `<pathcomponent>` is the path component of the URI that you want to use, and `<datatype>` is the name of the data type you want to use.

For example, if you add the `oslc.data.staff=Employees` to the properties file, you can use the following URL to access the Employees data type:
`http://example.com:9080/NCI_NCICLUSTER_oslc/data/staff`

where NCI is the default Impact Server name and NCICLUSTER is the Netcool/Impact cluster name.

2. Restart the Impact Server.

Example

The following example shows how to create a data type for a DB2 table and how to add the information to the **NCI_oslc.props** file.

In this example, the DB2 table has information for a table called **People**:

db2 => describe table People

Column name	Data type schema	Data type name	Column Length	Scale	Nulls
ID	SYSIBM	INTEGER	4	0	Yes
FIRST_NAME	SYSIBM	VARCHAR	255	0	Yes
LAST_NAME	SYSIBM	VARCHAR	255	0	Yes
COMPANY	SYSIBM	VARCHAR	255	0	Yes
BIRTHDAY	SYSIBM	DATE	4	0	Yes

1. Create a Netcool/Impact data type which represents the information in the DB2 table and add the information as fields to the data type:
 - a. In the navigation tree, expand **System Configuration > Event Automation > Data Model**, to open the **Data Model** tab.
 - b. Select the data source for which you want to create a data type, right-click the data source and click **New Data Type**.
 - c. In the **Data Type Name** field, give the data type a name, for example **Employees**.
 - d. Select the **Data Source Name** from the list menu, in this example DB2.
 - e. Select the **Enabled** check box to activate the data type so that it is available for use in policies.
 - f. Select the **Base Table** name from the list menu.
 - g. Click **Refresh** to add the fields from the DB2 example table to the data type.
 - h. Select at least one **Key Field**. Key fields are fields whose value or combination of values can be used to identify unique data items in a data type.
 - i. Click **Save**.
2. Specify the data type in the **NCI_oslc.props** file, for example `oslc.staff=Employees`.
3. Restart the Impact Server.

Retrieving OSLC resources that represent Netcool/Impact data items

You use OSLC resource collections to represent Netcool/Impact data items. You use a URL to retrieve these resource collections.

Before you begin

Use only data types that conform with standard database best practices. The key fields must be unique, non-NULL, and they must not change over time. If the key

values change, the OSLC URI also changes.

Procedure

Use a URL like the following one to retrieve the OSLC resource collections that represent the data items:

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/data/<datatype>
```

where <datatype> is defined in the NCI_oslc.props file.

Results

Netcool/Impact maps the rows in the database to OSLC resources and it maps columns to OSLC resource properties. The URL for each data item uses the key values in the form of HTTP matrix parameters to uniquely identify the data item. The key values are defined in the Netcool/Impact data type configuration. For example, a data item with multiple keys would result in a URI like this one:

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/data/people/item;  
<key1=value1>;<key2=value2>
```

Each non-NULL value in the database is represented as an RDF triple that consists of the data item, the value, and the property that is derived from the column name. NULL values are represented in OSLC by the absence of the property that is derived from the column name.

Example

For example, you can use the following URL to access the employee data type that is configured to use the people path component:

```
http://example.com:9080/NCICLUSTER_NCI_oslc/data/people/
```

The URL returns a collection of OSLC resources that are based on the rows from the database table. The following example shows the results for two data items that belong to the employee data type:

```
<?xml version="1.0"?>  
<rdf:RDF  
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
  xmlns:dcterms="http://purl.org/dc/terms/"  
  xmlns:people="http://jazz.net/ns/ism/event/impact#data/people/"  
  xmlns:impact="http://jazz.net/ns/ism/event/impact#/"  
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
  xmlns:oslc="http://open-services.net/ns/core#">  
  <oslc:ResponseInfo rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/people">  
    <rdfs:member>  
      <people:people rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc  
/data/people/item;ID=2">  
        <people:ID>2</people:ID>  
        <people:FIRST_NAME>George</people:FIRST_NAME>  
        <people:LAST_NAME>Friend</people:LAST_NAME>  
      </people:people>  
    </rdfs:member>  
    <rdfs:member>  
      <people:people rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc  
/data/people/item;ID=1">  
        <people:FIRST_NAME>Michael</people:FIRST_NAME>  
        <people:LAST_NAME>Ryan</people:LAST_NAME>  
        <people:ID>1</people:ID>  
      </people:people>  
    </rdfs:member>  
  </oslc:ResponseInfo>  
</rdf:RDF>
```



```

        </rdfs:member>
        <oslc:totalCount>2</oslc:totalCount>
    </oslc:ResponseInfo>
</rdf:RDF>

```

Displaying results for unique key identifier

Each resource that is returned is assigned a unique key that identifies the resource in the results and has certain information associated with it. You can use a URL to display the information associated with a specific identifier.

Example

You use the following URL to display the information associated with a particular resource key, in this case 1010:

```

http://example.com:9080/NCICLUSTER_NCI_oslc/policy/example/
myGetFilter/item;ID=1010

```

This URL returns the following results:

```

<rdf:RDF>
<examplePolicy:myGetFilter rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/
policy/example/myGetFilter/item;ID=1010">
    <myGetFilter:NAME>Brian Doe</myGetFilter:NAME>
    <myGetFilter:STARTED>1980-08-11</myGetFilter:STARTED>
    <myGetFilter:MANAGER>1001</myGetFilter:MANAGER>
    <myGetFilter:ID>1010</myGetFilter:ID>
    <myGetFilter:DEPT>Documentation</myGetFilter:DEPT>
</examplePolicy:myGetFilter>
</rdf:RDF>

```

OSLC resource shapes for data types

The OSLC resource shape represents the structure of the SQL schema that the Netcool/Impact data type is using. Netcool/Impact automatically produces an OSLC resource shape for the specified data types, extracting the data from the underlying database table.

Table 67. Mapping to OSLC Resource Shape properties

OSLC Resource Shape parameters	Maps to Netcool/Impact
dcterms:title	Data type name
oslc:describes	<a href="http://jazz.net/ns/ism/events/impact/data/<pathcomponent>">http://jazz.net/ns/ism/events/impact/data/<pathcomponent>

Table 68. OSLC properties generated by the Netcool/Impact data type parameters

OSLC property	Netcool/Impact data type
oslc:readOnly	Always 'true'
oslc:valueType	For more information, see the Table 69 on page 220.
dcterms:title	Column display name
oslc:propertyDefinition	<a href="http://jazz.net/ns/ism/events/impact/data/<pathcomponent>#<columnname>">http://jazz.net/ns/ism/events/impact/data/<pathcomponent>#<columnname>
oslc:occurs	oslc:ZeroOrOne
oslc:name	Column name
dcterms:description	Column description

Table 69. OSLC value type mapping:

Netcool/Impact column types	OSLC value types
String	http://www.w3.org/2001/XMLSchema#string
Integer, Long	http://www.w3.org/2001/XMLSchema#integer
Date, Timestamp	http://www.w3.org/2001/XMLSchema#dateTime
Float	http://www.w3.org/2001/XMLSchema#float
Double	http://www.w3.org/2001/XMLSchema#double
Boolean	http://www.w3.org/2001/XMLSchema#boolean
Anything else	http://www.w3.org/2001/XMLSchema#string

Viewing the OSLC resource shape for the data type

The OSLC resource shape for a data type is displayed in the `oslc:ResourceShape` property.

Example

The following example contains the OSLC resource shape for the **Employees** data type that was created for a DB2 table that is called People in the abbreviated RDF format.

The resource URI is as follows:

`http://<host>:9080/NCICLUSTER_NCI_oslc/data/resourceShapes/staff`

The URI returns the following RDF:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:oslc="http://open-services.net/ns/core#">
  <oslc:ResourceShape rdf:about=
    "http://<host>:9080/NCICLUSTER_NCI_oslc/data/resourceShapes/staff">
    <dcterms:title>Employees</dcterms:title>
    <oslc:property>
      <oslc:Property>
        <oslc:readOnly>true</oslc:readOnly>
        <oslc:valueType rdf:resource=
          "http://www.w3.org/2001/XMLSchema#string"/>
        <dcterms:title>LAST_NAME</dcterms:title>

        <oslc:propertyDefinition rdf:resource=
          "http://jazz.net/ns/ism/events/impact/data/staff/LAST_NAME"/>
        <oslc:occurs rdf:resource=
          "http://open-services.net/ns/core#Exactly-one"/>
        <oslc:name>LAST_NAME</oslc:name>
        <dcterms:description>LAST_NAME</dcterms:description>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>

        <oslc:readOnly>true</oslc:readOnly>
        <oslc:valueType rdf:resource=
          "http://www.w3.org/2001/XMLSchema#integer"/>
        <dcterms:title>ID</dcterms:title>
        <oslc:propertyDefinition rdf:resource=
```

```

"http://jazz.net/ns/ism/events/impact/data/staff/ID"/>
  <oslc:occurs rdf:resource=
"http://open-services.net/ns/core#Exactly-one"/>
  <oslc:name>ID</oslc:name>
  <dcterms:description>ID</dcterms:description>

</oslc:Property>
</oslc:property>
<oslc:property>
  <oslc:Property>
    <oslc:readOnly>true</oslc:readOnly>
    <oslc:valueType rdf:resource=
"http://www.w3.org/2001/XMLSchema#string"/>
    <dcterms:title>FIRST_NAME</dcterms:title>
    <oslc:propertyDefinition rdf:resource=
"http://jazz.net/ns/ism/events/impact/data/staff/FIRST_NAME"/>

    <oslc:occurs rdf:resource=
"http://open-services.net/ns/core#Exactly-one"/>
    <oslc:name>FIRST_NAME</oslc:name>
    <dcterms:description>FIRST_NAME</dcterms:description>
  </oslc:Property>
</oslc:property>
<oslc:property>
  <oslc:Property>
    <oslc:readOnly>true</oslc:readOnly>

    <oslc:valueType rdf:resource=
"http://www.w3.org/2001/XMLSchema#string"/>
    <dcterms:title>COMPANY</dcterms:title>
    <oslc:propertyDefinition rdf:resource=
"http://jazz.net/ns/ism/events/impact/data/staff/COMPANY"/>
    <oslc:occurs rdf:resource=
"http://open-services.net/ns/core#Exactly-one"/>
    <oslc:name>COMPANY</oslc:name>
    <dcterms:description>COMPANY</dcterms:description>
  </oslc:Property>

</oslc:property>
<oslc:property>
  <oslc:Property>
    <oslc:readOnly>true</oslc:readOnly>
    <oslc:valueType rdf:resource=
"http://www.w3.org/2001/XMLSchema#dateTime"/>
    <dcterms:title>BIRTHDAY</dcterms:title>
    <oslc:propertyDefinition rdf:resource=
"http://jazz.net/ns/ism/events/impact/data/staff/BIRTHDAY"/>
    <oslc:occurs rdf:resource=
"http://open-services.net/ns/core#Exactly-one"/>

    <oslc:name>BIRTHDAY</oslc:name>
    <dcterms:description>BIRTHDAY</dcterms:description>
  </oslc:Property>
</oslc:property>
<oslc:describes rdf:resource=
"http://jazz.net/ns/ism/events/impact/data/staff"/>
</oslc:ResourceShape>
</rdf:RDF>

```

Configuring custom URIs for data types and user output parameters

Netcool/Impact can act as a proxy to non-OSLC systems. To facilitate this function, you can use Netcool/Impact to represent data from a database as OSLC resources.

About this task

You use customized URIs to represent the data type columns. You need to add these customized URIs to the OSLC configuration file to facilitate the mapping.

Restriction:

All the namespace URIs that you specify must include http. You cannot use https.

Procedure

Add the following statement to the `NCI_oslc.props` file to specify a particular Type URI for a data type:

```
oslc.data.<path>.uri=<uri>
```

Optionally, you can add the following statement to specify a column name:

```
oslc.data.<path>.<columnname>.uri=<uri>
```

You can also add the following statement to specify a particular prefix for a namespace:

```
oslc.data.<path>.namespaces.<prefix>=<uri>
```

If you do not specify a prefix, the RDF that is returned automatically shows the generated prefix for the namespace.

Example

The following code example demonstrates how an employee table can be represented in a friend of a friend (FOAF) specification by adding the following statements to the `NCI_oslc.props` file:

```
oslc.data.staff=Employees
oslc.data.staff.uri=http://xmlns.com/foaf/0.1/Person
oslc.data.staff.NAME.uri=http://xmlns.com/foaf/0.1/name
oslc.data.staff.BIRTHDAY.uri=http://xmlns.com/foaf/0.1/birthday
oslc.data.staff.PHOTO.uri=http://xmlns.com/foaf/0.1/img
oslc.data.staff.STAFFPAGE.uri=http://xmlns.com/foaf/0.1/homepage
oslc.data.staff.EMAIL.uri=http://xmlns.com/foaf/0.1/mbox
```

When the user queries the OSLC resource `http://example.com:9080/NCICLUSTER_NCI_oslc/data/staff/jdoe`, the following RDF is returned.

Note: The example RDF is an approximation. Also, as the user did not specify the prefix and the namespace, the RDF automatically shows the generated prefix for the namespace. In this example, the namespace is `j.0` and the prefix is `http://xmlns.com/foaf/0.1/`.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://xmlns.com/foaf/0.1/"
  xmlns:impact="http://jazz.net/ns/ism/events/impact#"
  xmlns:oslc="http://open-services.net/ns/core#">
<j.0:Person rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/data/staff/jdoe"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <j.0:name>John Doe</foaf:name>
  <j.0:homepage rdf:resource="http://example.com" />
  <j.0:mbox rdf:resource="john.doe@example.com" />
```

```

<j.0:img rdf:resource="http://example.com/images/jdoe.jpg"/>
<j.0:birthday>19770801</foaf:birthday>
</foaf:Person>
</rdf:RDF>

```

The following code example demonstrates how to specify a particular prefix for a namespace. First, you specify the prefix and the namespace:

```
oslc.data.staff.namespaces.foaf=http://xmlns.com/foaf/0.1/
```

When the user queries the OSLC resource `http://example.com:9080/NCICLUSTER_NCI_oslc/data/staff/jdoe`, the following RDF is returned.

Note: The example RDF is an approximation.

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:impact="http://jazz.net/ns/ism/events/impact#/"
  xmlns:oslc="http://open-services.net/ns/core#">
<foaf:Person rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/data/staff/jdoe"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:name>John Doe</foaf:name>
  <foaf:homepage rdf:resource="http://example.com" />
  <foaf:mbox rdf:resource="john.doe@example.com" />
  <foaf:img rdf:resource="/images/jdoe.jpg" />
  <foaf:birthday>19770801</foaf:birthday>
</foaf:Person>
</rdf:RDF>

```

Working with the OSLC service provider

To allow other applications that are OSLC consumers to use OSLC data from Netcool/Impact, you must create the OSLC service provider, register the service provider with the Registry Service provided by Jazz for Service Management, and register the OSLC resources with the OSLC service provider.

To allow other applications that are OSLC consumers to use OSLC data from Netcool/Impact, you must complete the following tasks:

1. Create the OSLC service provider. See “Creating OSLC service providers in Netcool/Impact” on page 224
2. Register the OSLC service provider with the Registry Service provided by Jazz for Service Management. See “Registering OSLC service providers with Netcool/Impact” on page 226
3. Register the OSLC resources with the OSLC service provider. See “Registering OSLC resources” on page 227

The Registry Service is an integration service that is part of the Jazz for Service Management product. The Registry Service contains two directories, the Provider registry and the Resource registry. As part of the implementation of OSLC for Netcool/Impact, you must register the OSLC service provider and resources with the Resource registry.

For more information about the Registry Service and Jazz for Service Management, see http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.psc.doc_1.1.0/psc_ic-homepage.html.

If you are registering the OSLC resources with the Registry Service provided by Jazz for Service Management, you need use resources and RDF models that match the specifications that are defined in the common resource type vocabulary (CRTV). For more information, see the section about the common resource type vocabulary in the Registry Services guide (https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/W8b1151be2b42_4819_998e_f7de7db7bfa2/page/Milestone%20documentation).

The examples in this documentation use the namespace `crtv`. To integrate the Netcool/Impact OSLC service provider with the Registry Service provided by Jazz for Service Management, you must use the `crtv` namespace. If you do not want to integrate the OSLC service provider with the Registry Service provided by Jazz for Service Management, you must change the namespace. For more information about how to define a custom namespace, see “Configuring custom URIs for data types and user output parameters” on page 221.

Creating OSLC service providers in Netcool/Impact

Before you can use Netcool/Impact to register OSLC resources with the registry service provided by Jazz for Service Management, you must create an OSLC service provider in Netcool/Impact. To create an OSLC service provider, update the `NCI_oslc.props` configuration file.

About this task

The service provider definition is based on the OSLC resource collection that it is associated with. An OSLC resource collection can share a single provider or it can use multiple providers.

While you can use RDF policy functions to manually create the service provider, generally you use Netcool/Impact to generate a service provider automatically.

Procedure

1. To define a service provider, add the following statement to the `NCI_oslc.props` configuration file:

```
oslc.<type>.<path>.provider=<provider_name>
oslc.provider.<provider_name>.title=<title>
oslc.provider.<provider_name>.description=<description>
```

For example:

```
oslc.data.computer=RESERVATION
oslc.data.computer.provider=provider01
...
oslc.provider.provider01.title=Customer-x Product-y OSLC Service Provider
oslc.provider.provider01.description=Customer-x Product-y OSLC Service Provider
```

2. OSLC resources can share an OSLC service or they can use different OSLC services. This is controlled by the specified domain name. To specify a domain and a title for a resource, add the following statement to the `NCI_oslc.props` configuration file:

```
oslc.<type>.<path>.provider.domain=<domain_URI>
oslc.<type>.<path>.provider.title=<title>
```

For example:

```

oslc.data.computer=RESERVATION
...
oslc.data.computer.provider=provider01
oslc.data.computer.provider.domain=http://domainx/
oslc.data.computer.provider.title=Computer Title

```

If you specify the same service provider and domain name for two OSLC resources, both resources share a single OSLC service. If two resources use the same service provider but have different domains, the resources use different OSLC services. If no domain is specified, then the system uses the default Netcool/Impact namespace URI for this path.

3. Use this URL to view the service providers:

https://<server>:9081/NCICLUSTER_NCI_oslc/provider

The results are returned as an RDF. For example:

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:oslc="http://open-services.net/ns/core#"
  <rdf:Description rdf:about="https://<server>:9081/NCICLUSTER_NCI_oslc/
provider">
  <rdfs:member>
    <oslc:ServiceProvider rdf:about="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/provider/provider02">
      <oslc:service>
        <oslc:Service>
          <oslc:queryCapability>
            <oslc:QueryCapability>
              <dcterms:title>Query Capability - http://policy.js/xmlns/
directSQL</dcterms:title>
              <oslc:resourceType rdf:resource="http://policy.js/xmlns/
directSQL"/>
              <oslc:resourceShape rdf:resource="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/policy/resourceShapes/testBry/myDirectSQL1"/>
              <oslc:queryBase rdf:resource="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/policy/testBry/myDirectSQL1"/>
            </oslc:QueryCapability>
          </oslc:queryCapability>
          <oslc:domain rdf:resource="http://domainy"/>
        </oslc:Service>
      </oslc:service>
    </oslc:ServiceProvider>
  </rdfs:member>
  <rdfs:member>
    <oslc:ServiceProvider rdf:about="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/provider/provider01">
      <dcterms:title>Customer-x Product-y OSLC Service Provider
</dcterms:title>
      <dcterms:description>Customer-x Product-y OSLC Service Provider
</dcterms:description>
      <oslc:service>
        <oslc:Service>
          <oslc:queryCapability>
            <oslc:QueryCapability>
              <dcterms:title>Query Capability - http://jazz.net/ns/ism/events/
impact/data/managers</dcterms:title>
              <oslc:resourceType rdf:resource="http://jazz.net/ns/ism/
event/impact/data/managers"/>
              <oslc:resourceShape rdf:resource="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/data/resourceShapes/managers"/>
              <oslc:queryBase rdf:resource="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/data/managers"/>
            </oslc:QueryCapability>
          </oslc:queryCapability>
        </oslc:Service>
      </oslc:service>
    </oslc:ServiceProvider>
  </rdfs:member>

```

```

        </oslc:queryCapability>
        <oslc:queryCapability>
        <oslc:QueryCapability>
        <dcterms:title>Managers Title</dcterms:title>
        <oslc:resourceType rdf:resource="http://open-services.net/ns/
crtv#ComputerSystem"/>
        <oslc:resourceShape rdf:resource="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/data/resourceShapes/computer"/>
        <oslc:queryBase rdf:resource="https://<server ip>:9081/
NCICLUSTER_NCI_oslc/data/computer"/>
        </oslc:QueryCapability>
        </oslc:queryCapability>
        <oslc:domain rdf:resource="http://domainx"/>
        </oslc:Service>
        </oslc:service>
        </oslc:ServiceProvider>
        </rdfs:member>
        </rdf:Description>
    </rdf:RDF>

```

Registering OSLC service providers with Netcool/Impact

To specify the registry server information in the `NCI_oslc.props` file, add the OSLC registry server property to the `NCI_oslc.props` file.

Before you begin

Before you can specify the registry server information in the `NCI_oslc.props` file, you must create a service provider. See “Creating OSLC service providers in Netcool/Impact” on page 224.

Procedure

1. Specify the registry server, user name, and password. If the registry server does not require a user name and password, you do not need to specify them.

To specify the registry server, add the following statement to the `NCI_oslc.props` file:

```
impact.oslc.registry.server=<RegistryserverproviderregistryURL>
```

where `<RegistryserverproviderregistryURL>` is registry server provider's registry URL.

To specify a registry server user, add the following statement to the `NCI_oslc.props` file:

```
impact.oslc.registry.username=<OSLCproviderregistryserver
username>
```

To specify the registry server password, add the following statement to the `NCI_oslc.props` file:

```
impact.oslc.registry.password=<OSLCproviderregistryserver
password>
```

where `<OSLCproviderregistryserverpassword>` is the password for the OSLC provider registry server in encrypted form.

To locate the encrypted form of the password, run the `nci_crypt` program in the `impact/bin` directory. For example:

```
nci_crypt password
{aes}DE865CEE122E844A2823266AB339E91D
```

In this example, the password parameter uses the entire string, `{aes}DE865CEE122E844A2823266AB339E91D`, as the password.

- Restart Netcool/Impact to register the service providers. After the restart, Netcool/Impact registers the service providers in the service registry. If the service provider has been registered successfully, the resources that belong to the service provider contain a new property, `oslc:serviceProvider`. The `oslc:serviceProvider` property is displayed when you navigate to the URI that contains the resources associated with the provider.

```
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:RESERVATION="http://jazz.net/ns/ism/events/impact/data/computer/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:crtv="http://open-services.net/ns/crtv#"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:impact="http://jazz.net/ns/ism/events/impact/">
  <crtv:ComputerSystem rdf:about="http://<impact-server>:9080/NCICLUSTER_NCI_oslc/
data/computer/item;ID=4">
    <crtv:serialNumber>IBM00003SN</crtv:serialNumber>
    <crtv:model>IBM Model01</crtv:model>
    <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
    <oslc:serviceProvider rdf:resource="http://<registry-server>:9080/oslc/
providers/6015"/>
    <RESERVATION:RESERVED_DATE>2012-07-16</RESERVATION:RESERVED_DATE>
    <RESERVATION:RESERVED_BY>Michael Morton</RESERVATION:RESERVED_BY>
    <RESERVATION:RELEASE_DATE>2013-03-06</RESERVATION:RELEASE_DATE>
    <RESERVATION:ID>4</RESERVATION:ID>
  </crtv:ComputerSystem>
</rdf:RDF>
```

- Register the resource with the registry server. Netcool/Impact does not automatically register resources. See “Registering OSLC resources.”

Registering OSLC resources

Netcool/Impact does not automatically register OSLC resources with the services registry.

About this task

If you are registering the OSLC resources with the Registry Service provided by Jazz for Service Management, you need use resources and RDF models that match the specifications of the common resource type vocabulary (CRTV). For more information, see the section about the common resource type vocabulary in the Registry Services guide (http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.psc.doc_1.1.0/psc_ic-homepage.html).

If you want to view the resource record for OSLC resources that you register with the Registry Service provided by Jazz for Service Management, you must include the `crtv` namespace in the URL.

Procedure

To register OSLC resources, you can use one of the following two methods:

- Use the `RDFRegister` policy function in a policy to register the resource. For more information, see “`RDFRegister`” on page 232.
- Use the `GetHTTP` policy function to perform a HTTP POST function on the resource or list of resource members. You must define the `Method` parameter as `POST`. You can also use the OSLC query syntax to limit the properties that are registered as part of the resource.

Results

After you run the policy that contains the policy function, Netcool/Impact tries to register the resource or list of resource members included in the policy function.

Netcool/Impact also returns the response status, location header, and body text from the registry server to the client. The location header displays the location of the resources registration record for each resource that was registered. The body content that is contained in the response specifies the location of each registration record for each resource that was registered.

If a single resource is registered successfully, the system displays a 201 status code (Created) message. If multiple resources are registered successfully, the system displays a 200 status code (OK) message.

When you register multiple resources, Netcool/Impact also returns the following headers and the response body text from the registry server to the client:

- **NextPage:** If a next page of resources exists, the header contains the location URI of the next set of resources. If no next page exists, the response does not contain this header.
- **TotalCount:** The total number of resources across all pages. This header is returned when you register multiple resource.

The successful registration of an OSLC resource results in two records. A registration record is created in the resource registry. A resource record is also created and this record is available through the resource URI.

To view the registration records for the resource registry that is used by the Registry Service, add `/rr/registration/collection` to the URI. For example:
`http://example.com:9080/oslc/rr/registration/collection`

To view the registered resources for a service provider, such as the Registry Service, add `/rr/collection` to the Registry Service URL. For example:
`http://example.com:9080/oslc/rr/collection?oslc.select=*`

If the same resource is registered in two different instances because they belong to two different service providers, two registration records are created but only a single resource record is created and it is available through a single resource URI.

If you are integrating OSLC with the Registry Service and the OSLC resources are not displayed in this collection, check that the resources used match the modeling guidelines and use the common resource type vocabulary (CRTV). Also check that the resource URL contains the `crtv` namespace.

Single resource example

For example, consider the resource that is located at the following URL:

`http://<Impactserver>:9080/NCICLUSTER_NCI_oslc/data/computer/item;ID=4`

This returns the following RDF:

```
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:RESERVATION="http://jazz.net/ns/ism/event/impact/
data/computer/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  >
```

```

    xmlns:crtv="http://open-services.net/ns/crtv#"
    xmlns:oslc="http://open-services.net/ns/core#"
    xmlns:impact="http://jazz.net/ns/ism/event/impact#"
    <crtv:ComputerSystem rdf:about="http://<Impactserver>:9080/
NCICLUSTER_NCI_oslc/data/computer/item;ID=4">
    <crtv:serialNumber>IBM00003SN</crtv:serialNumber>
    <crtv:model>IBM Model01</crtv:model>
    <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
    <oslc:serviceProvider rdf:resource="http://
<registryserver>:9080/oslc/providers/6015"/>
    <RESERVATION:RESERVED_DATE>2012-07-16</RESERVATION:RESERVED_DATE>
    <RESERVATION:RESERVED_BY>Michael Morton</RESERVATION:RESERVED_BY>
    <RESERVATION:RELEASE_DATE>2013-03-06</RESERVATION:RELEASE_DATE>
    <RESERVATION:ID>4</RESERVATION:ID>
  </crtv:ComputerSystem>
</rdf:RDF>

```

Use the query syntax in the URL to limit the properties to crtvm:serialNumber, crtvm:model, crtvm:manufacturer, and oslc:serviceProvider:

```

http://<Impactserver>:9080/NCICLUSTER_NCI_oslc/data/computer/
item;ID=4?oslc.properties=crtvm:serialNumber,oslc:serviceProvider,
crtvm:manufacturer,crtvm:model

```

This URL returns the following RDF:

```

<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:RESERVATION="http://jazz.net/ns/ism/event/impact/data/
computer/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:crtv="http://open-services.net/ns/crtv#"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"
  <crtv:ComputerSystem rdf:about="http://<Impactserver>:9080/
NCICLUSTER_NCI_oslc/data/computer/item;ID=4">
    <crtv:serialNumber>IBM00003SN</crtv:serialNumber>
    <crtv:model>IBM Model01</crtv:model>
    <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
    <oslc:serviceProvider rdf:resource="http://<registryserver>:9080/
oslc/providers/6015"/>
  </crtv:ComputerSystem>
</rdf:RDF>

```

Use the following policy to perform a POST function on the URI of the resource. The POST function registers the resource with the resource registry associated with the serviceProvider property that is defined in the resource.

```

Log("SCR_RegisterSystems: Entering policy");
HTTPHost="impactserver";
HTTPPort=9080;
Protocol="http";
Path="/NCICLUSTER_NCI_oslc/data/computer/item;ID=4?oslc.properties
=crtvm:serialNumber,
oslc:serviceProvider,crtvm:manufacturer,crtvm:model";
ChannelKey="tom";
//Method="GET"; //Retrieves the Systems
Method="POST"; //Registers the Systems
AuthHandlerActionTreeName="";
FilesToSend=newobject();
HeadersToSend=newobject();
HttpProperties=newobject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="password";

x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName,

```

```

null, FilesToSend, HeadersToSend, HttpProperties);
Log(CurrentContext());
Log("SCR_RegisterSystems: HTTP Response: " + x);

```

After the policy runs and the resource is registered, the location of the registration record on the Registry Services server is detailed in the **Location** header.

Registering multiple resources

You can also use Netcool/Impact to register multiple resources in the resource registry.

Example

The following URL contains a set of resource members that are to be registered:

`http://<Impactserver>:9080/NCICLUSTER_NCI_oslc/data/computer/`

This URL returns the following RDF:

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:RESERVATION="http://jazz.net/ns/ism/event/impact/data/computer/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:crtv="http://open-services.net/ns/crtv#"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:impact="http://jazz.net/ns/ism/event/impact#">
  <rdf:Description rdf:about="http://<impact-server>:9080/
NCICLUSTER_NCI_oslc/data/computer/">
    <rdfs:member>
      <crtv:ComputerSystem rdf:about="http://<Impactserver>:
9080/NCICLUSTER_NCI_oslc/data/computer/item;ID=4">
        <crtv:serialNumber>IBM00003SN</crtv:serialNumber>
        <crtv:model>IBM Model01</crtv:model>
        <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
        <oslc:serviceProvider rdf:resource="http://<registry-server>:9080/
oslc/providers/6015"/>
        <RESERVATION:RESERVED_DATE>2012-07-16</RESERVATION:RESERVED_DATE>
        <RESERVATION:RESERVED_BY>Michael Morton</RESERVATION:RESERVED_BY>
        <RESERVATION:RELEASE_DATE>2013-03-06</RESERVATION:RELEASE_DATE>
        <RESERVATION:ID>4</RESERVATION:ID>
      </crtv:ComputerSystem>
    </rdfs:member>
    <rdfs:member>
      <crtv:ComputerSystem rdf:about="http://<Impactserver>:
:9080/NCICLUSTER_NCI_oslc/data/computer/item;ID=3">
        <crtv:serialNumber>IBM00002SN</crtv:serialNumber>
        <crtv:model>IBM Model01</crtv:model>
        <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
        <oslc:serviceProvider rdf:resource="http://
<registryserver>:9080/oslc/providers/6015"/>
        <RESERVATION:RESERVED_DATE>2011-02-20</RESERVATION:RESERVED_DATE>
        <RESERVATION:RESERVED_BY>Sandra Burton</RESERVATION:RESERVED_BY>
        <RESERVATION:RELEASE_DATE>2013-01-30</RESERVATION:RELEASE_DATE>
        <RESERVATION:ID>3</RESERVATION:ID>
      </crtv:ComputerSystem>
    </rdfs:member>
    <rdfs:member>
      <crtv:ComputerSystem rdf:about="http://<impact-server>:9080/
NCICLUSTER_NCI_oslc/data/computer/item;ID=0">
        <crtv:serialNumber>IBM00001SN</crtv:serialNumber>
        <crtv:model>IBM Model01</crtv:model>
        <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
        <oslc:serviceProvider rdf:resource="http://
<registryserver>:9080/oslc/providers/6015"/>
        <RESERVATION:RESERVED_DATE>2012-08-11</RESERVATION:RESERVED_DATE>
        <RESERVATION:RESERVED_BY>John Lewis</RESERVATION:RESERVED_BY>
        <RESERVATION:RELEASE_DATE>2013-04-12</RESERVATION:RELEASE_DATE>
        <RESERVATION:ID>0</RESERVATION:ID>
      </crtv:ComputerSystem>
    </rdfs:member>
  </rdf:Description>
</rdf:RDF>

```

```

        </crtv:ComputerSystem>
    </rdfs:member>
</rdf:Description>
<oslc:ResponseInfo rdf:about="http://<impact-server>9080/
NCICLUSTER_NCI_oslc/data/computer/?oslc.paging=true&oslc.pageSize=100">
    <oslc:totalCount>3</oslc:totalCount>
</oslc:ResponseInfo>
</rdf:RDF>

```

As this list contains a list of resource members, you can use the `oslc.select` query parameter to limit the properties of each resource member:

```

http://<Impactserver>:9080/NCICLUSTER_NCI_oslc/data/
computer?oslc.select=crtv:serialNumber,crtv:manufacturer,crtv:model,
oslc:serviceProvider

```

The URL returns the following RDF:

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:RESERVATION="http://jazz.net/ns/ism/event/impact/data/computer/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:crtv="http://open-services.net/ns/crtv#"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:impact="http://jazz.net/ns/ism/event/impact#">
  <oslc:ResponseInfo rdf:about="http://<Impactserver>:9080/
NCICLUSTER_NCI_oslc/data/computer?oslc.select=crtv:serialNumber,
crtv:manufacturer,crtv:model,oslc:serviceProvider&
oslc.paging=true&oslc.pageSize=100">
    <oslc:totalCount>3</oslc:totalCount>
  </oslc:ResponseInfo>
  <rdf:Description rdf:about="http://<Impactserver>:9080/
NCICLUSTER_NCI_oslc/data/computer?oslc.select=crtv:serialNumber,
crtv:manufacturer,crtv:model,oslc:serviceProvider">
    <rdfs:member>
      <crtv:ComputerSystem rdf:about="http://<Impactserver>:
9080/NCICLUSTER_NCI_oslc/data/computer/item;ID=4">
        <crtv:serialNumber>IBM00003SN</crtv:serialNumber>
        <crtv:model>IBM Model01</crtv:model>
        <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
        <oslc:serviceProvider rdf:resource="http://<registry-server>:
9080/oslc/providers/6015"/>
      </crtv:ComputerSystem>
    </rdfs:member>
    <rdfs:member>
      <crtv:ComputerSystem rdf:about="http://<Impactserver>:
9080/NCICLUSTER_NCI_oslc/data/computer/item;ID=3">
        <crtv:serialNumber>IBM00002SN</crtv:serialNumber>
        <crtv:model>IBM Model01</crtv:model>
        <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
        <oslc:serviceProvider rdf:resource="http://<registryserver>:
9080/oslc/providers/6015"/>
      </crtv:ComputerSystem>
    </rdfs:member>
    <rdfs:member>
      <crtv:ComputerSystem rdf:about="http://<Impactserver>:
9080/NCICLUSTER_NCI_oslc/data/computer/item;ID=0">
        <crtv:serialNumber>IBM00001SN</crtv:serialNumber>
        <crtv:model>IBM Model01</crtv:model>
        <crtv:manufacturer>IBM Manufacturer01</crtv:manufacturer>
        <oslc:serviceProvider rdf:resource="http://<registry-server>:
9080/oslc/providers/6015"/>
      </crtv:ComputerSystem>
    </rdfs:member>
  </rdf:Description>
</rdf:RDF>

```

Use the following policy to perform a POST function on the URI of the resources. The POST function registers the resources with the resource registry associated with the serviceProvider property that is defined in the resource.

```
Log("SCR_RegisterSystems: Entering policy");
HTTPHost="impactserver";
HTTPPort=9080;
Protocol="http";
Path="/NCICLUSTER_NCI_oslc/data/computer?oslc.paging
=true&oslc.pageSize=100";
ChannelKey="tom";
//Method="GET";
//Retreives the Systems Method="POST";
//Registers the Systems AuthHandlerActionTreeName="";
FilesToSend=newobject();
HeadersToSend=newobject();
HttpProperties=newobject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="password";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName, null,
FilesToSend, HeadersToSend, HttpProperties); Log(CurrentContext());
Log("SCR_RegisterSystems: HTTP Response: " + x);
```

If the resources are registered successfully, the system displays a message to confirm. Netcool/Impact also returns the header, body text, and other information that is contained in the response from the registry server to the client. The header and body text specify the location of each registration record for each resource that was registered.

RDFRegister

You can use the RDFRegister function to help you to register service providers or OSLC resources with the registry server.

Before you can register a service provider or resource, you must use the other RDF policy functions to build an RDF model that meets the OSLC and Registry Services requirements.

After you build the RDF model, use the RDFRegister function to register the RDF with the resource registry contained in the Registry Services integration service.

If the service provider or OSLC resource is registered successfully, the RDFRegister function returns the resource location of the registration record. The following variables and their return values are also returned to provide more information:

- ResultCode contains the result code for the response.
- HeadersReceived contains the headers received in the response.
- HeadersSent contains the headers sent in the response.
- ResponseBody contains the response body text.

If the query parameters are set in the URL and you use the RDFRegister policy function to register a service provider, you must manually add the location of the service provider to the policy. For example:

```
RDFStatement(newModel, manu[0].subject, "http://open-services.net/ns/
core#serviceProvider", serviceProviderURL, true);
```

If you use the query string inside the path, you must also ensure that the FormParameters parameter is set to null. For example:

```
FormParameters=null;
```

Finally, you must ensure that the policy contains pagination information. For example:

```
Path="/NCICLUSTER_NCI_oslc/data/mysql1?oslc.paging=true&oslc.pageSize=100";
```

If unsuccessful, the return value of the resource location registration record is null. Error code information is returned in the `ErrorReason` and `ResultCode` variables.

Syntax

The `RDFRegister` function has the following syntax:

```
[String =] RDFRegister(URI, Username, Password, Model)
```

where `Username` can be a null or void string to specify that no authentication is required.

Parameters

The `RDFRegister` function has the following parameters:

Table 70. RDFRegister function parameters

Parameter	Type	Description
URI	String	Registry Services server creation factory URI
Username	String	User name for the Registry Services server
Password	String	Password for the Registry Services server
Model	Model	Model that contains the RDF

Example

The following example manually registers a service provider and a set of resources that have been exposed by the OSLC server provider in Netcool/Impact.

The Registry Services server information is as follows:

```
RegistryServerProviderCFUri="http://<registry_services_server>:  
9080/oslc/pr/collection";  
RegistryServerResourceCFUri="http://<registry_services_server>:  
9080/oslc/rr/registration/collection";  
RegistryServerUsername="system";  
RegistryServerPassword="manager";
```

The Netcool/Impact server information is as follows:

```
HTTPHost="<impact_server>";  
HTTPPort=9080;  
Protocol="http";  
Path1="/NCICLUSTER_NCI_oslc/provider/provider01";  
Path2="/NCICLUSTER_NCI_oslc/data/computer";  
ChannelKey="";  
Method="GET";  
AuthHandlerActionTreeName="";  
FormParameters=NewObject();  
FilesToSend=NewObject();  
HeadersToSend=NewObject();  
HttpProperties = NewObject();
```

```

HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
HttpProperties.AuthenticationScheme="basic";

```

Get the service provider RDF from Netcool/Impact:

```

serviceProviderResponse=GetHTTP(HTTPHost,HTTPPort, Protocol, Path1,
ChannelKey,Method, AuthHandlerActionTreeName, null, FilesToSend,
HeadersToSend,HttpProperties);

```

Create an RDF model that is based on the service provider response:

```

serviceProviderModel=RDFParse(serviceProviderResponse)

```

Register the service provider in the provider registry:

```

serviceProviderURL = RDFRegister(RegistryServerProviderCFUri,
RegistryServerUsername, RegistryServerPassword,serviceProviderModel);
log("Provider Registry-Service Provider URL: " + serviceProviderURL);

```

Get all the computer system resources from Netcool/Impact:

```

allResources=GetHTTP(HTTPHost,HTTPPort, Protocol, Path2, ChannelKey,
Method,AuthHandlerActionTreeName, null, FilesToSend, HeadersToSend,
HttpProperties);

```

Create an RDF model that is based on the resource response:

```

allResourceModel=RDFParse(allResources);

```

Register each computer system and a set of properties with the resource registry:

```

statements=RDFSelect(allResourceModel, null, "http://jazz.net/ns/ism/
events/impact/data/computer/ID", null);
size=Length(statements);
count=0;
while(count<size) {
Path3=statements[count].subject;
//Get the individual computer system resource
resourceResponse=GetHTTP(HTTPHost,HTTPPort, Protocol, Path3, ChannelKey,
Method,AuthHandlerActionTreeName, null, FilesToSend, HeadersToSend,
HttpProperties);
resourceModel=RDFParse(resourceResponse);

```

Create a model that contains the properties and data that you want to register:

```

newModel=RDFModel();
manu=RDFSelect(resourceModel, null, "http://open-services.net/ns/
crtv#manufacturer",null);
model=RDFSelect(resourceModel, null, "http://open-services.net/ns/
crtv#model", null);
serial=RDFSelect(resourceModel, null, "http://open-services.net/ns/
crtv#serialNumber", null);
RDFModelUpdateNS(newModel, "crtv", "http://open-services.net/ns/crtv#");
RDFModelUpdateNS(newModel, "oslc","http://open-services.net/ns/core#");
RDFStatement(newModel, manu[0].subject, "http://www.w3.org/1999/02/
22-rdf-syntax-ns#type",
"http://open-services.net/ns/crtv#ComputerSystem", true);
RDFStatement(newModel, manu[0].subject, manu[0].predicate, manu[0].object,
RDFNodeIsResource(manu[0].object));
RDFStatement(newModel, manu[0].subject, model[0].predicate, model[0].object,
RDFNodeIsResource(manu[0].object));
RDFStatement(newModel, manu[0].subject, serial[0].predicate,
serial[0].object, RDFNodeIsResource(manu[0].object));

```

Update the model with the service provider location:

```

RDFStatement(newModel, manu[0].subject, "http://open-services.net/ns/
core#serviceProvider", serviceProviderURL, true);

```


Register the resource in the resource registry:

```
resourceURL = RDFRegister(RegistryServerResourceCFUri,  
RegistryServerUsername, RegistryServerPassword, newModel);  
log("Resource Registry-Resource URL: " +resourceURL);  
  
count=count+1;  
}
```

RDFUnRegister

To remove the registration record of a service provider or resource from the registry server, use the RDFUnRegister function to supply the location of the registration record, the Registry Services server username and password, and the registration record that you want to remove.

Before you can remove the registration record of a service provider, you must remove all the registration records for the associated OSLC resources.

If successful, the RDFUnRegister function returns the message code 204 and the value true. The following variables and their return values are also returned to provide additional information:

- ResultCode contains the result code for the response.
- HeadersReceived contains the headers received in the response.
- HeadersSent contains the headers sent in the response.
- ResponseBody contains the response body text.

If unsuccessful, the return value of the resource location registration record is false. Error code information is returned in the ErrorReason and ResultCode variables.

Syntax

The RDFUnRegister function has the following parameters:

```
[ String =] RDFUnRegister(URI, Username , Password)
```

where Username can be a null or void string to specify that no authentication is required.

Parameters

Table 71. RDFUnRegister function parameters

Parameter	Type	Description
URI	String	Location that contains the registration record for the resource or service provider
Username	String	User name for the Registry Services server
Password	String	Password for the Registry Services server

Example of how to remove the registration of a service provider

The following example demonstrates how to remove the registration of the service provider.

The service provider location is:

```
http://<registryserver>:9080/oslc/providers/6577
```

Use the `RDFUnRegister` function to remove the registration. For example:

```
//Registry server information
ServiceProviderUri="http://<registryserver>:9080/oslc/
providers/6577";
RegistryServerUsername="system";
RegistryServerPassword="manager";
result = RDFUnRegister(ServiceProviderUri, RegistryServerUsername,
RegistryServerPassword);
```

Example of how to remove the registration of an OSLC resource

The following example demonstrates how to use the policy function to remove the registration of an OSLC resource.

```
registrationURL = "http://nc004075.romelab.it.ibm.com:16310/oslc/registration/
1351071987349";
providerURL = "http://nc004075.romelab.it.ibm.com:16310/oslc/providers/
1351071987343";
RegistryServerUsername="smadmin";
RegistryServerPassword="tbsm01bm";

returnString = RDFUnRegister (registrationURL, RegistryServerUsername,
RegistryServerPassword);
```

Working with Netcool/Impact policies and OSLC

You can integrate the Netcool/Impact OSLC provider and Netcool/Impact policies.

Accessing output user parameters as OSLC resources

To use the Netcool/Impact OSLC provider to run Netcool/Impact policies and access the results, you must edit the `NCI_oslc.props` file that is in the `IMPACT_HOME/etc` directory.

About this task

Netcool/Impact returns two types of RDF objects, literals and resources. RDF literals contain an actual value. RDF resources are returned as URLs that you can access to find more information about an object.

Procedure

1. To access Netcool/Impact policy results, edit the `NCI_oslc.props` file that is in the `IMPACT_HOME/etc` directory, where `NCI` is the name of your Impact Server. Add the following statement for each policy that you want to access:
`oslc.policy.<pathcomponent>=<policyname>`
2. Restart the Impact Server.

Example

For example, you add the following to the `NCI_oslc.props` file to access the `SNMPTableTest` policy:

```
oslc.policy.tabletest=SNMPTableTest
```

Use the following URL to run the policy and return the results:

```
http://example.com:9080/NCI_NCICLUSTER_oslc/policy/tabletest
```

where NCI is the Impact Server name and NCICLSUTER is the Netcool/Impact cluster name.

When you access this URL, the policy runs and the output user parameters are available as RDF resources.

OSLC and variables output by policy results

Simple variables, such as string, integer, double, float, Boolean, and date/timestamp are made available as RDF literals. More complex variables such as impact objects, arrays, and function results are displayed as RDF resources with an RDF link that contains internal details of the variable.

This example shows the user output parameters from the Example policy. The results of this policy contain both RDF literals and resources. The following URL triggers the policy execution and makes the results available as OSLC resources:

http://example.com:9080/NCICLUSTER_NCI_oslc/policy/example/

The information is returned as:

```
<rdf:RDF
  <examplePolicy:example rdf:about="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example">
    <example:myArrayStr rdf:resource="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example/myArrayStr"/>
    <example:myObject rdf:resource="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example/myObject"/>
    <example:myObjArray rdf:resource="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example/myObjArray"/>
    <example:myGetFilter rdf:resource="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example/myGetFilter"/>
    <example:MyAlerts rdf:resource="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example/MyAlerts"/>
    <example:myString>Brian</example:myString>
    <example:myDouble>22.5</example:myDouble>
    <example:myFloat>100.55</example:myFloat>
    <example:myInteger>32</example:myInteger>
    <example:myBoolean>true</example:myBoolean>
  </examplePolicy:example>
</rdf:RDF>
```

The more complex variables in this example, such as myObject, myArrayStr, myObjArray, and myGetFilter, are displayed as resource links. The other variables are simple variables, such as myString, myDouble, myFloat, myInteger, and myBoolean, that are displayed as literals alongside their values.

You use the following URL to access the resource URL used for Netcool/Impact objects, represented by the myObject variable:

http://example.com:9080/NCICLUSTER_NCI_oslc/policy/example/myObject

The resource URL returns the results as:

```
<rdf:RDF>
  <examplePolicy:myObject rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/
policy/example/myObject">
    <myObject:bmi>24.5</myObject:bmi>
    <myObject:lname>Doe</myObject:lname>
    <myObject:fname>John</myObject:fname>
    <myObject:age>25</myObject:age>
    <oslc:totalCount>1</oslc:totalCount>
    <rdf:type rdf:resource="http://open-services.net/ns/core#ResponseInfo"/>
  </examplePolicy:myObject>
</rdf:RDF>
```

Accessing arrays of variables from policy results

To access an array of variables that are contained in a policy result in an OSLC context, you use a URL that contains the variable name.

Before you begin

The default array prefix is `oslc_pos`. To change this setting, add the following definition to the `NCI_oslc.props` file:

```
oslc.policy.<pathcomponent>.arrayprefix=<prefix>.
```

For example, add the following definition to the `NCI_oslc.props` file to change the prefix to `pos` for the example path component:

```
oslc.properties.example.arrayprefix=pos
```

Procedure

To access the resource URL used for an array of objects, represented in this example by the `myArrayStr` variable, use the following URL:

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/policy/  
example/myArrayStr
```

Results

This URL returns the following results:

```
<rdf:RDF>  
  <examplePolicy:myArrayStr rdf:about="http://<server>:<port>/  
NCICLUSTER_NCI_oslc/policy/example/myArrayStr">  
    <myArrayStr:oslc_pos_2>Hi</myArrayStr:oslc_pos_2>  
    <myArrayStr:oslc_pos_1>Hey</myArrayStr:oslc_pos_1>  
    <myArrayStr:oslc_pos_0>Hello</myArrayStr:oslc_pos_0>  
    <oslc:totalCount>1</oslc:totalCount>  
    <rdf:type rdf:resource="http://open-services.net/ns/core#ResponseInfo"/>  
  </examplePolicy:myArrayStr>  
</rdf:RDF>
```

If an array variable contains multiple Netcool/Impact objects, then a resource that contains a link to multiple resources is created. Each of these resources contains a link to the actual Netcool/Impact object in the array.

Example

Use the following URL to access the array of variables that is represented by the `myObjArray` variable:

```
http://example.com:9080/NCICLUSTER_NCI_oslc/policy/example/myObjArray/
```

As this array contains multiple objects, the URL returns the following results:

```
<rdf:RDF>  
  <oslc:ResponseInfo rdf:about="http://example.com:9080/  
NCICLUSTER_NCI_oslc/policy/example/myObjArray/">  
    <rdfs:member>  
      <examplePolicy:myObjArray rdf:about="http://example.com:9080/  
NCICLUSTER_NCI_oslc/policy/example/myObjArray">  
        <myObjArray:oslc_pos_2 rdf:resource="http://example.com:9080/  
NCICLUSTER_NCI_oslc/policy/example/myObjArray/oslc_pos_2"/>  
        <myObjArray:oslc_pos_1 rdf:resource="http://example.com:9080/  
NCICLUSTER_NCI_oslc/policy/example/myObjArray/oslc_pos_1"/>  
        <myObjArray:oslc_pos_0 rdf:resource="http://example.com:9080/  
NCICLUSTER_NCI_oslc/policy/example/myObjArray/oslc_pos_0"/>
```

```

        </examplePolicy:myObjArray>
    </rdfs:member>
    <oslc:totalCount>1</oslc:totalCount>
</oslc:ResponseInfo>
</rdf:RDF>

```

If you access the URL associated with one of the Netcool/Impact objects, the following results are returned:

```

<rdf:RDF>
  <myObjArray:oslc_pos_1 rdf:about="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/example/myObjArray/oslc_pos_1">
    <oslc_pos_1:fname>Garrett</oslc_pos_1:fname>
    <oslc_pos_1:bmi>33.1</oslc_pos_1:bmi>
    <oslc_pos_1:age>30</oslc_pos_1:age>
  </myObjArray:oslc_pos_1>
</rdf:RDF>

```

Displaying the resource shapes for policy results

Resource shapes are available for any OSLC resource produced by the Netcool/Impact OSLC provider. The resource shape defines the set of OSLC properties for a specific operation.

Procedure

To display the resource shape for any OSLC object, add `resourceShapes` to the URL.

Example

For example, you use the following URL to display the resource shape definition for the specified resource:

```

http://example.com:9080/NCICLUSTER_NCI_oslc/policy/resourceShapes/
example/myGetFilter/item;ID=1010

```

This URL returns the following results which include the resource shape definition:

```

<rdf:RDF
  <oslc:ResourceShape rdf:about="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/resourceShapes/example/myGetFilter">
    <dcterms:title>examplePolicy</dcterms:title>
    <oslc:property>
      <oslc:Property rdf:about="http://xmlns.com/foaf/0.1/givenName">
        <oslc:readOnly>true</oslc:readOnly>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <dcterms:title>NAME</dcterms:title>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myGetFilter/NAME"/>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
        <oslc:name>NAME</oslc:name>
        <dcterms:description>NAME</dcterms:description>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myGetFilter/STARTED">
        <oslc:readOnly>true</oslc:readOnly>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
        <dcterms:title>STARTED</dcterms:title>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myGetFilter/STARTED"/>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
        <oslc:name>STARTED</oslc:name>
        <dcterms:description>STARTED</dcterms:description>
      </oslc:Property>
    </oslc:property>

```

```

        </oslc:Property>
    </oslc:property>
    <oslc:property>
        <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myGetFilter/MANAGER">
            <oslc:readOnly>true</oslc:readOnly>
            <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
            <dcterms:title>MANAGER</dcterms:title>
            <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myGetFilter/MANAGER"/>
            <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
            <oslc:name>MANAGER</oslc:name>
            <dcterms:description>MANAGER</dcterms:description>
        </oslc:Property>
    </oslc:property>
    <oslc:property>
        <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myGetFilter/ID">
            <oslc:readOnly>true</oslc:readOnly>
            <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
            <dcterms:title>ID</dcterms:title>
            <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myGetFilter/ID"/>
            <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
            <oslc:name>ID</oslc:name>
            <dcterms:description>ID</dcterms:description>
        </oslc:Property>
    </oslc:property>
    <oslc:property>
        <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myGetFilter/DEPT">
            <oslc:readOnly>true</oslc:readOnly>
            <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
            <dcterms:title>DEPT</dcterms:title>
            <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myGetFilter/DEPT"/>
            <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
            <oslc:name>DEPT</oslc:name>
            <dcterms:description>DEPT</dcterms:description>
        </oslc:Property>
    </oslc:property>
    <oslc:property>
        <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myGetFilter/CEASED">
            <oslc:readOnly>true</oslc:readOnly>
            <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
            <dcterms:title>CEASED</dcterms:title>
            <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myGetFilter/CEASED"/>
            <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
            <oslc:name>CEASED</oslc:name>
            <dcterms:description>CEASED</dcterms:description>
        </oslc:Property>
    </oslc:property>
    <oslc:describes rdf:resource="http://xmlns.com/foaf/0.1/Group"/>
</oslc:ResourceShape>
</rdf:RDF>

```

OSLC and UI data provider compatible variables for policy results

The Netcool/Impact OSLC provider and the UI data provider can remotely run a policy and make the results available as an OSLC or UI data provider compatible resource with property values that contain the user output parameters.

The following variable types are supported:

- String
- Integer
- Double
- Float
- Boolean
- Date/Timestamp
- Impact Object
- Long (represented as an Integer in OSLC)

Variable arrays for each of these variables are also supported. These arrays must of the same variable type.

The variables returned by the `GetByFilter` and `DirectSQL` functions are also supported.

Configuring user parameters

To use the UI data provider or OSLC with your Netcool/Impact policies, you must configure user parameters to make the policy results compatible with the UI data provider or available as OSLC resources.

About this task

You can create either policy runtime parameters or policy output parameters. Policy runtime parameters represent the runtime parameters that you define in policies. For example, you can use a policy runtime parameter to pass values from one policy to another in a data mashup.

Policy output parameters represent the parameters that are output by policies. For example, the UI data provider uses policy output parameters to visualize data from policies in the console.

Procedure

1. To open the policy user parameter editor in the policy editor toolbar, click the **Configure User Parameters** icon.
2. To create a policy output parameter, click **New Output Parameter:New**. To create a policy runtime parameter, click **New Runtime Parameter:New**. Mandatory fields are denoted by an asterisk (*). You must enter a unique name in the **Name** field.
3. Define the custom schemas for the output parameters if required.
If you are using the `DirectSQL` policy function with OSLC, you must define the custom schema for it.
If you are using **DirectSQL**, **Impact Object**, or **Array of Impact Object** with the UI data provider or the chart widget, you must define the custom schema for these values.
For more information, see “Creating custom schema values for output parameters” on page 170
4. To save the changes to the parameters and close the window, click **OK**.

Example

This example demonstrates how to create output parameters for a policy. First, you define a simple policy, like:

```

first_name = "Mark";
zip_code = 12345;
Log("Hello " + first_name + " living at " + zip_code);

```

Next, define the output parameters for this policy. In this case, there are two output parameters. You enter the following information:

Table 72. PolicyDT1 output parameter

Field	User entry
Name	Enter a unique name. For example, PolicyDT1.
Policy variable name	first_name
Format	String

Table 73. PolicyDT2 output parameter

Field	User entry
Name	Enter a unique name. For example, PolicyDT2
Policy variable name	zip_code
Format	Integer

Accessing data types output by the GetByFilter function

If you want to access the results from the GetByFilter function, you need to create output parameters for the OSLC provider.

Procedure

1. To open the policy user parameter editor, click the **Configure User Parameters** icon in the policy editor toolbar. You can create policy user parameters for runtime and output. To open the Create a New Policy Output Parameter window, click **New**.
2. Select data type as the format.
3. Enter the name of the data item to which the output of the GetByFilter function is assigned in the **Policy Variable Name** field.
4. Enter the name of the data source in the **Data Source Name** field.
5. Enter the name of the data type in the **Data Type Name** field.

Example

This example demonstrates how to make the output from the GetByFilter function available to the Netcool/Impact OSLC provider.

You create a data type called ALERTS that belongs to the defaultobjectserver data source. This data type belongs to Netcool/OMNIBus and it points to alerts.status. The key field is Identifier. The following four rows of data are associated with the key field:

- Event1
- Event2
- Event3
- Event4

You create the following policy, called Test_Policy3:


```
MyAlerts = GetByFilter("ALERTS", "Severity > 0", false);
```

Next, you define the output parameters for the policy as follows:

Table 74. PolicyData1 output parameter

Field	User entry
Name	PolicyData1
Policy Variable Name	MyAlerts
Format	Datatype
Data Source Name	defaultobjectserver
Data Type Name	ALERTS

Accessing variables output by the DirectSQL function

To access variables output by the DirectSQL policy function, you must create DirectSQL output parameters and format values.

About this task

If a variable is output by the DirectSQL policy function, Netcool/Impact creates an RDF resource. This resource contains multiple properties for each defined output parameter.

Only the following simple variables are supported:

- String
- Double
- Integer
- Long
- Date/Timestamp
- Boolean

If the column names contain special characters, you must add a statement that lists these special characters to the `NCI_server.props` file. For more information, see the topic about using special characters in column names in the section about troubleshooting general Netcool/Impact issues in the Troubleshooting Guide.

If the policies that you use to provide data to OSLC contain special characters, you must escape these special characters. For more information, see the topic about using special characters in OSLC and UI data provider policies in the section about troubleshooting OSLC in the Troubleshooting Guide.

Procedure

To access variables output by the DirectSQL policy function, create a DirectSQL output parameter and define the DirectSQL values for this parameter. For a detailed description of these steps, see “Creating custom schema values for output parameters” on page 170.

Example

This example demonstrates how to access variables output by the DirectSQL policy function. You define the following policy which uses the DirectSQL function:

```
MyAlerts=DirectSQL('Omnibus','select min(Serial) as min_serial,
max(Serial) as max_serial,count(<) as num_events from alerts.status', false);
```

Next, define the DirectSQL output parameter as outlined in the table. You do not need to enter a data source or data type name.

Table 75. DirectSQL output parameter

Field	User Entry
Name	DirectSQL_OP1
Policy Variable Name	DirectSQL_1
Format	DirectSQL

To create the DirectSQL format values, click the **DirectSQL** editor icon. Define the format values as follows:

Table 76. DirectSQL format values

Name	Format	Key
min_serial	Double	True
max_serial	Float	True
num_events	Integer	True

Use the following URI to run the policy and return the results:

```
http://example.com:9080/NCICLUSTER_NCI_oslc/
policy/examplePolicy/MyAlerts
```

The results are:

```
<rdf:RDF
  <oslc:ResponseInfo rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/
policy/examplePolicy/MyAlerts">
  <rdfs:member>
    <examplePolicy:MyAlerts rdf:about="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/examplePolicy/MyAlerts/
item;min_serial=133;num_events=12;max_serial=521">
      <MyAlerts:num_events>12</MyAlerts:num_events>
      <MyAlerts:min_serial>133</MyAlerts:min_serial>
      <MyAlerts:max_serial>521</MyAlerts:max_serial>
    </examplePolicy:MyAlerts>
  </rdfs:member>
  <oslc:totalCount>1</oslc:totalCount>
</oslc:ResponseInfo>
</rdf:RDF>
```

The results also contain the resource shape:

```
<rdf:RDF
  <oslc:ResourceShape rdf:about="http://example.com:9080/
NCICLUSTER_NCI_oslc/policy/resourceShapes/example/MyAlerts">
  <dcterms:title>examplePolicy</dcterms:title>
  <oslc:property>
    <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myDirectSQL/num_events">
      <oslc:readOnly>true</oslc:readOnly>
      <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
      <dcterms:title>num_events</dcterms:title>
      <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myDirectSQL/num_events"/>
      <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
      <oslc:name>num_events</oslc:name>
```

```

        <dcterms:description>num_events</dcterms:description>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myDirectSQL/min_serial">
        <oslc:readOnly>true</oslc:readOnly>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
        <dcterms:title>min_serial</dcterms:title>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myDirectSQL/min_serial"/>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
        <oslc:name>min_serial</oslc:name>
        <dcterms:description>min_serial</dcterms:description>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property rdf:about="http://jazz.net/ns/ism/events/impact/policy/
example/myDirectSQL/max_serial">
        <oslc:readOnly>true</oslc:readOnly>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
        <dcterms:title>max_serial</dcterms:title>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/events/
impact/policy/example/myDirectSQL/max_serial"/>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
        <oslc:name>max_serial</oslc:name>
        <dcterms:description>max_serial</dcterms:description>
      </oslc:Property>
    </oslc:property>
    <oslc:describes rdf:resource="http://jazz.net/ns/ism/events/impact/
policy/example/">
  </oslc:ResourceShape>
</rdf:RDF>

```

Creating custom schema values for output parameters:

When you define output parameters that use the **DirectSQL**, **Array of Impact Object**, or **Impact Object** format in the user output parameters editor, you also must specify a name and a format for each field that is contained in the **DirectSQL**, **Array of Impact Object**, or **Impact Object** objects.

About this task

Custom schema definitions are used by Netcool/Impact to visualize data in the console and to pass values to the UI data provider and OSLC. You create the custom schemas and select the format that is based on the values for each field that is contained in the object. For example, you create a policy that contains two fields in an object:

```

01.city="NY"
01.ZIP=07002

```

You define the following custom schemas values for this policy:

Table 77. Custom schema values for City

Field	Entry
Name	City
Format	String


Table 78. Custom schema values for ZIP

Field	Entry
Name	ZIP
Format	Integer

If you use the DirectSQL policy function with the UI data provider or OSLC, you must define a custom schema value for each DirectSQL value that you use.

If you want to use the chart widget to visualize data from an Impact object or an array of Impact objects with the UI data provider and the console, you define custom schema values for the fields that are contained in the objects. The custom schemas help to create descriptors for columns in the chart during initialization. However, the custom schemas are not technically required. If you do not define values for either of these formats, the system later rediscovers each Impact object when it creates additional fields such as the key field. UIObjectId, or the field for the tree widget, UITreeNodeId. You do not need to define these values for OSLC.

Procedure

1. In the policy user parameters editor, select **DirectSQL**, **Impact Object**, or **Array of Impact Object** in the **Format** field.
2. The system shows the **Open the Schema Definition Editor** icon  beside the **Schema Definition** field. To open the editor, click the icon.
3. You can edit an existing entry or you can create a new one. To define a new entry, click **New**. Enter a name and select an appropriate format.
To edit an existing entry, click the **Edit** icon beside the entry that you want to edit
4. To mark an entry as a key field, select the check box in the **Key Field** column. You do not have to define the key field for Impact objects or an array of Impact objects. The system uses the UIObjectId as the key field instead.
5. To delete an entry, select the entry and click **Delete**.

Configuring custom URIs for policy results and variables

You can assign custom URIs to policy and user output parameters or variables to create a custom mapping. You can use this mapping to represent a resource in any domain.

About this task

Netcool/Impact supports only the one-to-one mapping of user output parameters to OSLC properties.

Procedure

1. To add a custom URI to a policy resource, add the following definition to the NCI_oslc.prop file:
`oslc.policy.<pathcomponent>.uri=<uri>`
2. To add a custom URI to a variable, specify the variable and the path component. As there are multiple layers of variables, you must specify each variable until you reach the one that you want:

```
oslc.policy.<pathcomponent>.<variablename>.uri=<uri>
oslc.policy.<pathcomponent>.<variablename>.<variablename>
...<variablename>.uri=uri
```

Example

The following example demonstrates how the example policy can be represented in a Friend of a Friend (FOAF) specification. You start by adding statements to the NCI_oslc.props file:

```
oslc.policy.example=examplePolicy
oslc.policy.example.namespaces.foaf=http://xmlns.com/foaf/0.1/
oslc.policy.example.uri=http://xmlns.com/foaf/0.1/Person
oslc.policy.example.myGetFilter.NAME.uri=http://xmlns.com/foaf/0.1/givenName
```

You use this URL to query the OSLC resource:

```
http://example.com:9080/NCICLUSTER_NCI_oslc/policy/example/myGetFilter
```

This URL returns the RDF:

Note: This example is an approximation for exemplary purposes.

```
<rdf:RDF
  <oslc:ResponseInfo rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/policy/
example/myGetFilter">
  <rdfs:member>
    <j.0:Person rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/policy/
example/myGetFilter/item;ID=1012">
      <j.0:givenName>Kevin Doe</j.0:givenName>
      <myGetFilter:STARTED>1976-07-06</myGetFilter:STARTED>
      <myGetFilter:MANAGER>1001</myGetFilter:MANAGER>
      <myGetFilter:ID>1012</myGetFilter:ID>
      <myGetFilter:DEPT>Documentation</myGetFilter:DEPT>
    </j.0:Person>
  </rdfs:member>
  <rdfs:member>
    <j.0:Person rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/policy/
example/myGetFilter/item;ID=1010">
      <j.0:givenName>Brian Doe</j.0:givenName>
      <myGetFilter:STARTED>1980-08-11</myGetFilter:STARTED>
      <myGetFilter:MANAGER>1001</myGetFilter:MANAGER>
      <myGetFilter:ID>1010</myGetFilter:ID>
      <myGetFilter:DEPT>Documentation</myGetFilter:DEPT>
    </j.0:Person>
  </rdfs:member>
  <oslc:totalCount>2</oslc:totalCount>
</oslc:ResponseInfo>
</rdf:RDF>
```

The user has not defined the prefix and namespace in this example. In this case, the RDF shows the automatically generated prefix for the namespace j.0 for <http://xmlns.com/foaf/0.1/>.

You specify a prefix for a namespace in this format:

```
oslc.policy.<path>.namespaces.<prefix>=<uri>
```

For this example, you add this statement:

```
oslc.policy.example.namespaces.foaf=http://xmlns.com/foaf/0.1/
```

When you use the following URL to query the OSLC resource, the RDF is produced with the prefix specified by you:

```
http://example.com:9080/NCICLUSTER_NCI_oslc/policy/example/myGetFilter
```

This URL returns as:

```
<rdf:RDF
  <oslc:ResponseInfo rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/policy/
example/myGetFilter">
    <rdfs:member>
      <foaf:Person rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/policy/
example/myGetFilter/item;ID=1012">
        <foaf:givenName>Kevin Doe</foaf:givenName>
        <myGetFilter:STARTED>1976-07-06</myGetFilter:STARTED>
        <myGetFilter:MANAGER>1001</myGetFilter:MANAGER>
        <myGetFilter:ID>1012</myGetFilter:ID>
        <myGetFilter:DEPT>Documentation</myGetFilter:DEPT>
      </foaf:Person>
    </rdfs:member>
    <rdfs:member>
      <foaf:Person rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/policy/
example/myGetFilter/item;ID=1010">
        <foaf:givenName>Brian Doe</foaf:givenName>
        <myGetFilter:STARTED>1980-08-11</myGetFilter:STARTED>
        <myGetFilter:MANAGER>1001</myGetFilter:MANAGER>
        <myGetFilter:ID>1010</myGetFilter:ID>
        <myGetFilter:DEPT>Documentation</myGetFilter:DEPT>
      </foaf:Person>
    </rdfs:member>
    <oslc:totalCount>2</oslc:totalCount>
  </oslc:ResponseInfo>
</rdf:RDF>
```

Passing argument values to a policy

You can use URL query strings to pass argument values in the form of a string to a policy. You can access these values by creating a user output parameter for each of the arguments.

Procedure

Use the following URL to pass argument values to a policy:

```
http://<host>:<port>/NCI_NCI_CLUSTER_oslc/policy/
<path name>?arg1=<value>&arg2=<value>
```

Restriction: Unusually long URLs can cause issues. This depends on the browser and the Websphere application server settings. To avoid these issues, limit the size of the values of the variables that are passed through the query string.

Results

After you access this URL, the policy runs. If you do not define any output user parameters, the user parameters are available as properties within an OSLC resource.

Example

For example, you use the following URL to pass the variable arg1 with the string value table1 to the policy defined in the tableset path:

```
http://example.com:9080/NCI_NCI_CLUSTER_oslc/policy/tableset?arg1=table1
```

Configuring hover previews for OSLC resources

You can enable hover previews for OSLC resources. You can configure a title and other aspects of the hover preview like the size of the window.

About this task

For more information about using hover previews for OSLC, see Open Services for Lifecycle Collaboration Core Specification Version 2.0 UI Preview (<http://open-services.net/bin/view/Main/OslcCoreUiPreview>)

Procedure

To configure hover previews, add a set of properties for each OSLC resource to the `NCI_oslc.props` file. You can use any combination of these properties. Some of these properties only display if the document or icon parameter exists in the OSLC resource. For a detailed description of these properties, see “Hover preview properties for OSLC resources” on page 250.

Results

Each set of properties that you define for an OSLC resource generates a compact XML representation of the hover preview. This compact XML is used to help generate the content for the hover preview window in other applications. Each set of properties can contain variables. When the XML is generated, the variables are replaced with property values from the OSLC resource in the following format:

`<$<prefixnamespace>:<propertyname>`

For example:

```
$RESERVATION:HOSTNAME  
$RESERVATION:ID
```

To view all the possible variables for an OSLC resource, use the OSLC resource URI to view the full XML representation.

If a resource does not exist or an error occurs, the system displays an error code 400 and a message that explains the issue.

If the resource does not support hover previews, for example if the resource contains `rdfs:member` lists, the system displays a 406 Not Acceptable error code.

If no hover preview parameters are defined for the resource, the system displays a compact XML that contains no parameters other than those parameters about the URI.

If you design a third-party hover preview consumer like TBSM, you can add `application/x-oslc-compact+xml` to the URL as an HTTP Accept header to display the hover preview compact XML document in the response.

Example

The following example demonstrates how to configure the hover preview for an OSLC resource that is based on a database table called `RESERVATION`. The following hover preview settings are defined in the `NCI_oslc.props` file:

```
oslc.data.computer=RESERVATION  
oslc.data.computer.uri=http://open-services.net/ns/crtv#ComputerSystem  
oslc.data.computer.MODEL.uri=http://open-services.net/ns/crtv#model  
oslc.data.computer.MANUFACTURER.uri=http://open-services.net/ns/crtv#manufacturer  
oslc.data.computer.SERIALNUMBER.uri=http://open-services.net/ns/crtv#serialNumber  
oslc.data.computer.namespaces.crtv=http://open-services.net/ns/crtv#  
oslc.data.computer.provider=provider01  
oslc.data.computer.provider.domain=http://domainx/
```

```

oslc.data.computer.preview.title=Computer Reservation System - $RESERVATION:HOSTNAME
oslc.data.computer.preview.shortTitle=Reservation
oslc.data.computer.preview.largePreview.document=https://<impactserver>:
16311/opview/displays/NCICLUSTER-Reservations.html?id=$RESERVATION:ID
oslc.data.computer.preview.largePreview.hintWidth=31.250em
oslc.data.computer.preview.largePreview.hintHeight=21.875em

```

Next, derive the hover preview content. In this example, use a Netcool/Impact operator. In this case, two variables are generated in the compact XML:

```

$RESERVATION:HOSTNAME
$RESERVATION:ID

```

These variables are converted into the property values based on data from the OSLC resource:

```

$RESERVATION:HOSTNAME = mycomputer.ibm.com
$RESERVATION:ID = 4

```

When you use an HTTP GET method on the resource URL with the application/x-osl原因-compact+xml HTTP Accept header, the following RDF is returned:

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:oslc="http://open-services.net/ns/core#"
  <oslc:Compact rdf:about="http://example.com:9080/NCICLUSTER_NCI_oslc/data/
computer/item;ID=4">
  <dcterms:title>Computer Reservation System - mycomputer.ibm.com</dcterms:title>
  <oslc:shortTitle>Reservation</oslc:shortTitle>
  <oslc:largePreview>
    <oslc:Preview>
      <oslc:hintWidth>31.250em</oslc:hintWidth>
      <oslc:hintHeight>21.875em</oslc:hintHeight>
      <oslc:document rdf:resource="https://<impact-server>:16311/opview/displays/
NCICLUSTER-Reservations.html?id=4"/>
    </oslc:Preview>
  </oslc:largePreview>
</oslc:Compact>
</rdf:RDF>

```

Hover preview properties for OSLC resources

To configure hover previews, add the parameters that are listed in the tables to the NCI_oslc.props file. Some of these parameters only display if the document or icon parameter exists in the OSLC resource

Table 79. Hover preview parameters

Statement	Description
oslc.<type>.<path>.preview.title = <longtitle>	where <longtitle> specifies the long title string that is used for the hover preview.
oslc.<type>.<path>.preview.shortTitle = <shorttitle>	where <shorttitle> specifies the short title string that is used for the hover preview. In an example implementation with TBSM, this statement supplies the tab name for the hover preview.
oslc.<type>.<path>.preview.icon = <URIof16x16image>	where <URIof16x16image> specifies the URI for a 16x16 image.

Table 79. Hover preview parameters (continued)

Statement	Description
oslc.<type>.<path>.preview.largePreview. document = <PreviewdocumentURI>	where <PreviewdocumentURI> specifies the URI used for the HTML preview document.
oslc.<type>.<path>.preview.smallPreview. document = <PreviewdocumentURI>	where <PreviewdocumentURI> specifies the URI for the HTML preview document. In an example implementation with TBSM, this statement specifies the content rendered inside the hover preview.

Table 80. Hover preview parameters for icon parameter

Statement	Description
oslc.<type>. <path>.preview. iconTitle = <IconTitle>	where <IconTitle> specifies the title that is used for the icon.
oslc.<type>.<path>.preview.iconAltLabel = <AlternativeLabel>	where <AlternativeiconTitle> specifies an alternative label for the icon.

Table 81. Hover preview parameters for document parameter

Statement	Description
oslc.<type>.<path>.preview.largePreview. hintWidth = <Previewwindowwidth>	where <Previewwindowwidth> specifies the width of the preview window. For example, 31.250 em.
oslc.<type>.<path>.preview.largePreview. hintHeight = <Previewwindowheight>	where <Previewwindowheight> specifies the height of the preview window. For example, 21.785 em.
oslc.<type>.<path>.preview.largePreview. initialHeight = <Previewwindowinitialheight>	where <Previewwindowinitialheight> specifies the height of the preview window when it first displays. For example, 21.785 em.
oslc.<type>.<path>.preview.smallPreview. hintWidth = <Previewwindowwidth>	where <Previewwindowwidth> specifies the width of the small preview window. For example, 31.250 em. In an example implementation with TBSM, this statement specifies the width of the hover preview window.
oslc.<type>.<path>.preview.smallPreview. hintHeight = <Previewwindowheight>	where <Previewwindowheight> specifies the height of the small preview window. For example, 21.785 em. In an example implementation with TBSM, this statement specifies the height of the hover preview window.
oslc.<type>.<path>.preview.smallPreview. initialHeight = <Previewwindowinitialheight>	where <Previewwindowinitialheight> specifies the height of the small preview window when it first displays. For example, 21.785 em.

where <type> is the OSLC resource type. It can be either a data or policy.

<path> is the OSLC resource path.

Example scenario: using OSLC with Netcool/Impact policies

Read this example scenario to get an overview of how you can use Netcool/Impact policies to create OSLC service providers and resources, and register the provider and resources with the Registry Services component of Jazz for Service Management.

Before you begin

Before you can use OSLC, you must install the Registry Services component of Jazz for Service Management. For more information, see http://pic.dhe.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.psc.doc_1.1.0/psc_ic-homepage.html.

About this task

The Registry Services component of Jazz for Service Management includes two registries, the resource registry and the provider registry. This example first demonstrates how to create a service provider and register it with the provider registry. The second step creates an OSLC resource and registers it with the resource registry.

Procedure

1. Create a Netcool/Impact policy that creates a service provider and registers it in the provider registry that is part of the Registry Services component of Jazz for Service Management:

- a. Define the server information for the server where the Registry Services component of Jazz for Service Management is installed:

```
RegistryServerProviderCFUri="http://<registry_server>:9080/oslc/pr/collection";
RegistryServerUsername="<user>";
RegistryServerPassword="<password>";
```

- b. Define the service provider information:

```
dsTitle = "Customer-x Product-y OSLC Service Provider";
dsDescription = "Customer-x Product-y OSLC Service Provider";
provider = "http://<URL>/<myProvider>";
domain = "http://<Domain>";
```

- c. Use the RDFModel policy function to create the service provider RDF model:

```
serviceProviderModel = RDFModel();
```

- d. Update the namespace definitions:

```
RDFModelUpdateNS(serviceProviderModel, "oslc", "http://open-services.net/ns/core#");
RDFModelUpdateNS(serviceProviderModel, "dcterms", "http://purl.org/dc/terms/");
```

- e. Create the RDF statements and add them to the model:

```
RDFStatement(serviceProviderModel, provider, "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
"http://open-services.net/ns/core#ServiceProvider", true);
RDFStatement(serviceProviderModel, provider, "http://purl.org/dc/terms/title", dsTitle, false);
RDFStatement(serviceProviderModel, provider, "http://purl.org/dc/terms/description",
dsDescription, false);
serviceStmt=RDFStatement(serviceProviderModel, null, "http://www.w3.org/1999/
02/22-rdf-syntax-ns#type", "http://open-services.net/ns/core#Service", true);
RDFStatement(serviceProviderModel, serviceStmt.getSubject, "http://open-services.net/ns/
core#domain", domain, true);
RDFStatement(serviceProviderModel, provider, "http://open-services.net/ns/core#service",
serviceStmt.getSubject, true);
```

```
log("-----Service Provider RDF-----");
log(RDFModelToString(serviceProviderModel, "RDF/XML-ABBREV"));
```

- f. Use the RDFRegister policy function to register the service provider in the provider registry.

```
serviceProviderURL = RDFRegister(RegistryServerProviderCFUri, RegistryServerUsername,
RegistryServerPassword, serviceProviderModel);
log("Registered service provider: " + serviceProviderURL);
```

2. Create and register an OSLC resource in the resource registry:

- a. Define the server information for the resource registry:

```
RegistryServerResourceCFUri="http://<registry_server>:9080/oslc/rr/  
registration/collection";  
RegistryServerUsername="<user>";  
RegistryServerPassword="<password>";
```

- b. Define the OSLC resource, in this example it represents a computer system:

```
computerSystem = "http://<OSLC_resource_URL>/mySystemX";  
name = "mySystemX";  
manufacturer = "VMware";  
serialNumber = "422ABA0619B0DE94B02E40870D6462AF";  
model = "VMWAREVIRTUALPLATFORM";
```

The service provider is located at the following URL. You can retrieve this URL after you register the service provider:

```
"http://<registry_server>:9080/oslc/providers/1358241368487"
```

- c. Create the RDF model that represents the computer system:

```
computerSystemModel = RDFModel();  
RDFModelUpdateNS(computerSystemModel, "crtv", "http://open-services.net/ns/crtv#");  
RDFModelUpdateNS(computerSystemModel, "oslc", "http://open-services.net/ns/core#");  
  
RDFStatement(computerSystemModel, computerSystem, "http://www.w3.org/1999/02/  
22-rdf-syntax-ns#type",  
"http://open-services.net/ns/crtv#ComputerSystem", true);  
RDFStatement(computerSystemModel, computerSystem, "http://open-services.net/  
ns/crtv#name", name, false);  
RDFStatement(computerSystemModel, computerSystem, "http://open-services.net/  
ns/crtv#model", model, false);  
RDFStatement(computerSystemModel, computerSystem, "http://open-services.net/  
ns/crtv#manufacturer", manufacturer, false);  
RDFStatement(computerSystemModel, computerSystem, "http://open-services.net/  
ns/crtv#serialNumber", serialNumber, false);  
RDFStatement(computerSystemModel, computerSystem, "http://open-services.net/  
ns/core#serviceProvider", serviceProviderURL, true);
```

```
log("-----Computer System RDF-----");  
log(RDFModelToString(computerSystemModel, "RDF/XML-ABBREV"));
```

- d. Register the computer system in the resource registry.

```
registrationRecordURL = RDFRegister(RegistryServerResourceCFUri, RegistryServerUsername,  
RegistryServerPassword, computerSystemModel);  
log("Registered service provider: " + registrationRecordURL);
```

OSLC reference topics

Read the following reference information for OSLC.

OSLC urls

You use the following URLs to access OSLC data.

Access data items

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/data/<datatype>/
```

Retrieve the OSLC resource shape for the data type

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/data/resourceShapes/  
<datatype>
```

Run policy and return the results

```
http://<server>:<port>/NCI_NCICLUSTER_oslc/policy/<polycyname>
```

Access array of variables from policy results

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/policy/<polycyname>/  
<variablearray>
```

Display results for a unique key identifier:

```
http://<server>:<port>/NCICLUSTER_NCI_oslc/policy/<polycyname>/  
<function>item;ID=<uniquekeyidentifier>
```

OSLC pagination

The Netcool/Impact OSLC provider supports pagination to make the retrieval of large amounts of data easier and more efficient.

Pagination is enabled by default for data items and policies whose variables contain data items. To manually configure pagination, add the following query parameters to the URL:

`?oslc.paging=true&oslc.page=<pagenumber>&oslc.pageSize=<pagesize>`

- `oslc.paging=true` enables pagination. This setting is enabled by default. To disable pagination, use `oslc.paging=false`.
- `oslc.page=<pagenumber>` is the page number. This property is set to page 1 by default.
- `oslc.pageSize=<pagesize>` is the page size. This property is set to 100 by default.

Administrators can add the following statement to the `NCI_server.props` configuration file to set the default limit for the page size:

`impact.oslc.pageSize=<pagesize>`

If this property is not defined, it is set to 100 by default.

If the page size in the URL is greater than the limit that is defined in the `NCI_server.props` configuration file, the size is limited to that set in the `NCI_server.props` configuration file.

You can also add the `oslc.paging=false` property to the URL to disable pagination. If this property is set, the entire result set is returned. If any additional pagination properties are defined, these properties are ignored. If you disable pagination and you also enable large data model support, this can have an adverse affect on performance.

Response information

Two properties are added to the response information: `oslc:nextPage` and `oslc:totalCount`.

The `oslc:nextPage` property is not returned when there is no next page. If the page size of the result is smaller than the specified page size property, no next page property is returned.

The `oslc:totalCount` property gives the total count information across all the pages.

Example

For example, the following URL represents the alert variables that belong to the `GetByFilter` function events:

`http://example.com:9080/NCICLUSTER_NCI_oslc/policy/events/alerts?
oslc.paging=true&oslc.page=2&oslc.pageSize=25`

In the URL, pagination is enabled. The variable results are contained in page two. The page size is limited to 25.

OSLC security

OSLC security is enabled by default. You can use the `configOSLCSecurity` script to disable OSLC security.

Disabling OSLC Security

For hover preview to work with other products such as TBSM you have the following options.

- Enable SSO or LDAP between TBSM and Netcool/Impact. A user must also have access to the OSLC security role.
- Disable the OSLC security by running the `configOSLCSecurity` script.

The scripts are in the following directories. For Netcool/Impact `/opt/IBM/tivoli/impact/bin`, for TBSM `/opt/IBM/tivoli/tbsm/bin`.

For Windows, run the following command.

```
configOSLCSecurity.bat <disable> <Server> <username> <password>
```

For Netcool/Impact

```
configOSLCSecurity.bat disable NCI tipadmin tippass
```

For TBSM

```
configOSLCSecurity.bat disable TBSM tipadmin tippass
```

For UNIX, run the following command.

```
configOSLCSecurity.sh <disable> <Server> <username> <password>
```

For Netcool/Impact

```
./configOSLCSecurity.sh disable NCI tipadmin tippass
```

For TBSM

```
./configOSLCSecurity.sh disable TBSM tipadmin tippass
```

To enable OSLC security, replace `<disable>` with `<enable>` for each command.

Support for OSLC query syntax

Netcool/Impact supports a version of the query syntax for OSLC. The implementation in Netcool/Impact supports the `oslc.properties` and `oslc.select` query parameters.

Both query parameters have very similar functionality. The only difference is the meaning of the identifiers. If the identifiers belong to the starting subject resource, use `oslc.properties`. If the identifiers belong to a member list for a resource, use `oslc.select`.

For more information about the query syntax for OSLC, see the Open Services for Lifecycle Collaboration Core Specification Version 2.0 specification (<http://open-services.net/bin/view/Main/OslcCoreSpecification>)

`oslc.properties` query parameter

Use the `oslc.properties` query parameter to display the properties for an individual resource URI that does not contain any `rdfs:member` lists

If the identifiers that you want to limit the properties of belong to the starting subject, use the `oslc.properties` query parameter to limit the properties that are returned by the identifiers.

Example

To display properties for an individual resource URI that contains no `rdfs:member` list, use the `oslc.properties` query parameter:

```
http://<server>:9080/NCICLUSTER_NCI_oslc/data/staff/item;  
FNAME=%27Todd%27;LNAME=%27Bishop%27?oslc.properties=foaf:lastName,  
foaf:firstName
```

The URI returns the following information:

```
<rdf:RDF  
  xmlns:dcterms="http://purl.org/dc/terms/"  
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
  xmlns:EMPLOYEES="http://jazz.net/ns/ism/event/impact/data/staff/"  
  xmlns:oslc="http://open-services.net/ns/core#"  
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"  
  xmlns:foaf="http://xmlns.com/foaf/">  
  <foaf:Person rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/  
staff/item;FNAME=&apos;Todd&apos;;LNAME=&apos;Bishop&apos;">  
    <foaf:lastName>Bishop</foaf:lastName>  
    <foaf:firstName>Todd</foaf:firstName>  
  </foaf:Person>  
</rdf:RDF>
```

oslc.select query parameter

If the identifiers that you want to limit the properties of belong to a member list for a resource, use the `oslc.select` query parameter.

Use `oslc.select` to complete the following tasks:

- Limit the properties of a member list that belongs to an OSLC resource. For example, to limit the properties of a member list that belongs to an OSLC resource that is generated from a data item.
- Display an OSLC resource that contains `rdfs:member` lists. For example, to display the results of a policy function variable.

Data items example

If you query a URL that contains a list of resources that belong to a database table, the system returns a `rdfs:member` list for each row. For example, to query the staff data type and to limit the properties that are contained in the list to the `foaf:lastName` and `foaf:firstName` properties, use the `oslc.select` query parameter:

```
http://<server>:9080/NCICLUSTER_NCI_oslc/data/  
staff?oslc.select=foaf:lastName,foaf:firstName
```

This URL returns only the `rdfs:member` lists that contain the `foaf:lastName` and `foaf:firstName` properties:

```
<?xml version="1.0"?>  
<rdf:RDF  
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
  xmlns:dcterms="http://purl.org/dc/terms/"  
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
  xmlns:EMPLOYEES="http://jazz.net/ns/ism/event/impact/data/staff/"  
  xmlns:oslc="http://open-services.net/ns/core#"  
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"  
  xmlns:foaf="http://xmlns.com/foaf/">
```

```

    <oslc:ResponseInfo rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff?oslc.select=foaf:lastName,foaf:firstName&
oslc.paging=true&oslc.pageSize=100">
    <oslc:totalCount>5</oslc:totalCount>
  </oslc:ResponseInfo>
  <rdf:Description rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff?oslc.select=foaf:lastName,foaf:firstName">
    <rdfs:member>
      <foaf:Person rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff/item;FNAME=&apos;Mika&apos;;LNAME=&apos;Masion&apos;;">
        <foaf:lastName>Masion</foaf:lastName>
        <foaf:firstName>Mika</foaf:firstName>
      </foaf:Person>
    </rdfs:member>
    <rdfs:member>
      <foaf:Person rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff/item;FNAME=&apos;Kevin&apos;;LNAME=&apos;Doe&apos;;">
        <foaf:lastName>Doe</foaf:lastName>
        <foaf:firstName>Kevin</foaf:firstName>
      </foaf:Person>
    </rdfs:member>
    <rdfs:member>
      <foaf:Person rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff/item;FNAME=&apos;Todd&apos;;LNAME=&apos;Bishop&apos;;">
        <foaf:lastName>Bishop</foaf:lastName>
        <foaf:firstName>Todd</foaf:firstName>
      </foaf:Person>
    </rdfs:member>
    <rdfs:member>
      <foaf:Person rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff/item;FNAME=&apos;Miriam&apos;;LNAME=&apos;Masters&apos;;">
        <foaf:lastName>Masters</foaf:lastName>
        <foaf:firstName>Miriam</foaf:firstName>
      </foaf:Person>
    </rdfs:member>
    <rdfs:member>
      <foaf:Person rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/data/
staff/item;FNAME=&apos;Brian&apos;;LNAME=&apos;Doe&apos;;">
        <foaf:lastName>Doe</foaf:lastName>
        <foaf:firstName>Brian</foaf:firstName>
      </foaf:Person>
    </rdfs:member>
  </rdf:Description>
</rdf:RDF>

```

Policy variable example

If the starting resource contains the member lists, use the `oslc.select` query parameter. For example, if the resource contains a policy variable such as `MyDirectSQL`, use `oslc.select`:

```

http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myDirectSQL2?
oslc.select=myDirectSQL2:num_events

```

This URL returns the following information:

```

<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:direct="http://policy.js/xmlns/directSQL/"
  xmlns:test_ip1="http://jazz.net/ns/ism/event/impact/policy/ipl/"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:myDirectSQL2="http://jazz.net/ns/ism/event/impact#policy/
ipl/myDirectSQL2/"
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"
  xmlns:javascript="http://policy.js/xmlns/">

```

```

    <rdf:Description rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/
policy/ipl/myDirectSQL2?
<oslc.select=myDirectSQL2:num_events">
    <rdfs:member>
        <test_ipl:myDirectSQL2 rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/
policy/ipl/
myDirectSQL2/item;min_serial=165248;num_events=928;max_serial=387781">
            <myDirectSQL2:num_events>928</myDirectSQL2:num_events>
        </test_ipl:myDirectSQL2>
    </rdfs:member>
</rdf:Description>
<oslc:ResponseInfo rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/
ipl/myDirectSQL2?
<oslc.select=myDirectSQL2:num_events&oslc.paging=true&oslc.pageSize=100">
<oslc.totalCount>1</oslc:totalCount>
</oslc:ResponseInfo>
</rdf:RDF>

```

Nested variables and wildcard queries

Use wildcard queries to display nested variables.

To display all variables and their nested values, use the following statement:

```
oslc.properties=*
```

To display all the variables, their nested values, and the policy functions, use the following statement:

```
oslc.properties=*&oslc.select=*
```

Example

For example, you define a policy and you want to use the following URL to access it:

```
http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl
```

The URL returns the following information:

```

<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:direct="http://policy.js/xmlns/directSQL/"
  xmlns:test_ipl="http://jazz.net/ns/ism/event/impact/policy/ipl/"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"
  xmlns:javascript="http://policy.js/xmlns/"
  <rdf:Description rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl">
    <test_ipl:myTimestamp>20120818</test_ipl:myTimestamp>
    <test_ipl:myString>test_ipl</test_ipl:myString>
    <test_ipl:myInteger>55</test_ipl:myInteger>
    <test_ipl:myImpactObjectArray rdf:resource="http://<server>:9080/
NCICLUSTER_NCI_oslc/policy/ipl/myImpactObjectArray"/>
    <test_ipl:myImpactObject1 rdf:resource="http://<server>:9080/
NCICLUSTER_NCI_oslc/policy/ipl/myImpactObject1"/>
    <test_ipl:myDouble>109.5</test_ipl:myDouble>
    <rdf:type rdf:resource="http://jazz.net/ns/ism/event/impact/policy/ipl/">
  </rdf:Description>
</rdf:RDF>

```

This example contains several RDF literals such as myTimestamp, myString, and myInteger. It also contains the myImpactObjectArray RDF resource.

Use the following URL to show all the variables and their nested values:


```
http://<server>:9080/NCICLUSTER_NCI_oslc/policy/  
ipl?oslc.properties=*
```

This returns the following information:

```
<rdf:RDF  
  xmlns:dcterms="http://purl.org/dc/terms/"  
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
  xmlns:myImpactObjectArray="http://jazz.net/ns/ism/event/impact/policy/  
ipl/myImpactObjectArray/"  
  xmlns:myImpactObject1="http://jazz.net/ns/ism/event/impact/policy/  
ipl/myImpactObject1/"  
  xmlns:direct="http://policy.js/xmlns/directSQL/"  
  xmlns:test_ipl="http://jazz.net/ns/ism/event/impact/policy/ip1/"  
  xmlns:oslc_pos_2="http://jazz.net/ns/ism/event/impact/policy/  
ipl/myImpactObjectArray/oslc_pos_2/"  
  xmlns:oslc="http://open-services.net/ns/core#"  
  xmlns:oslc_pos_1="http://jazz.net/ns/ism/event/impact/policy/  
ipl/myImpactObjectArray/oslc_pos_1/"  
  xmlns:oslc_pos_0="http://jazz.net/ns/ism/event/impact/policy/  
ipl/myImpactObjectArray/oslc_pos_0/"  
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"  
  xmlns:javascript="http://policy.js/xmlns/">  
<rdf:Description rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ip1">  
  <test_ipl:myTimestamp>20120818</test_ipl:myTimestamp>  
  <test_ipl:myString>test_ipl</test_ipl:myString>  
  <test_ipl:myInteger>55</test_ipl:myInteger>  
  <test_ipl:myImpactObjectArray>  
    <test_ipl:myImpactObjectArray rdf:about="http://<server>:9080/  
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObjectArray">  
      <myImpactObjectArray:oslc_pos_2>  
        <myImpactObjectArray:oslc_pos_2 rdf:about="http://<server>:9080/  
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObjectArray/oslc_pos_2">  
          <oslc_pos_2:lname>Doe</oslc_pos_2:lname>  
          <oslc_pos_2:fname>Kevin</oslc_pos_2:fname>  
          <oslc_pos_2:email>kdoe@us.ibm.com</oslc_pos_2:email>  
          <oslc_pos_2:birthday>1973-01-22</oslc_pos_2:birthday>  
        </myImpactObjectArray:oslc_pos_2>  
      </myImpactObjectArray:oslc_pos_2>  
      <myImpactObjectArray:oslc_pos_1>  
        <myImpactObjectArray:oslc_pos_1 rdf:about="http://<server>:9080/  
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObjectArray/oslc_pos_1">  
          <oslc_pos_1:lname>Doe</oslc_pos_1:lname>  
          <oslc_pos_1:fname>Danny</oslc_pos_1:fname>  
          <oslc_pos_1:email>doe@us.ibm.com</oslc_pos_1:email>  
          <oslc_pos_1:birthday>1976-05-12</oslc_pos_1:birthday>  
        </myImpactObjectArray:oslc_pos_1>  
      </myImpactObjectArray:oslc_pos_1>  
      <myImpactObjectArray:oslc_pos_0>  
        <myImpactObjectArray:oslc_pos_0 rdf:about="http://<server>:9080/  
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObjectArray/oslc_pos_0">  
          <oslc_pos_0:lname>Doe</oslc_pos_0:lname>  
          <oslc_pos_0:fname>John</oslc_pos_0:fname>  
          <oslc_pos_0:email>jdoe@us.ibm.com</oslc_pos_0:email>  
          <oslc_pos_0:birthday>1980-08-11</oslc_pos_0:birthday>  
        </myImpactObjectArray:oslc_pos_0>  
      </myImpactObjectArray:oslc_pos_0>  
    </test_ipl:myImpactObjectArray>  
  </test_ipl:myImpactObjectArray>  
  <test_ipl:myImpactObject1>  
    <test_ipl:myImpactObject1 rdf:about="http://<server>:9080/  
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObject1">  
      <myImpactObject1:lname>Doe</myImpactObject1:lname>  
      <myImpactObject1:fname>John</myImpactObject1:fname>  
      <myImpactObject1:email>jdoe@us.ibm.com</myImpactObject1:email>  
      <myImpactObject1:birthday>1980-08-11</myImpactObject1:birthday>  
    </test_ipl:myImpactObject1>
```

```

        </test_ip1:myImpactObject1>
        <test_ip1:myDouble>109.5</test_ip1:myDouble>
        <rdf:type rdf:resource="http://jazz.net/ns/ism/event/impact/policy/ip1/">
    </rdf:Description>
</rdf:RDF>

```

Notice that the myImpactObjectArray array is expanded to show each ImpactObject that it contains and the property values for each of the ImpactObjects.

To obtain a specific property value for one of the resources, use a URL that specifies the nested properties. For example:

```

http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ip1?oslc.
properties=test_ip1:myImpactObjectArray{myImpactObjectArray:
oslc_pos_0{oslc_pos_0:lname,oslc_pos_0:fname}}

```

This URL returns the following information:

```

<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:myImpactObjectArray="http://jazz.net/ns/ism/event/impact/policy/
ip1/myImpactObjectArray/"
  xmlns:myImpactObject1="http://jazz.net/ns/ism/event/impact/policy/
ip1/myImpactObject1/"
  xmlns:direct="http://policy.js/xmlns/directSQL/"
  xmlns:test_ip1="http://jazz.net/ns/ism/event/impact/policy/ip1/"
  xmlns:oslc_pos_2="http://jazz.net/ns/ism/event/impact/policy/
ip1/myImpactObjectArray/oslc_pos_2/"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:oslc_pos_1="http://jazz.net/ns/ism/event/impact/policy/
ip1/myImpactObjectArray/oslc_pos_1/"
  xmlns:oslc_pos_0="http://jazz.net/ns/ism/event/impact/policy/
ip1/myImpactObjectArray/oslc_pos_0/"
  xmlns:tivoli-impact="http://jazz.net/ns/ism/event/impact#"
  xmlns:javascript="http://policy.js/xmlns/">
  <rdf:Description rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/
policy/ip1">
    <test_ip1:myImpactObjectArray>
      <test_ip1:myImpactObjectArray rdf:about="http://<server>:9080/
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObjectArray">
        <myImpactObjectArray:oslc_pos_0>
          <myImpactObjectArray:oslc_pos_0 rdf:about="http://<server>:9080/
NCICLUSTER_NCI_oslc/policy/ip1/myImpactObjectArray/oslc_pos_0">
            <oslc_pos_0:lname>Doe</oslc_pos_0:lname>
            <oslc_pos_0:fname>John</oslc_pos_0:fname>
          </myImpactObjectArray:oslc_pos_0>
        </myImpactObjectArray:oslc_pos_0>
      </test_ip1:myImpactObjectArray>
    </test_ip1:myImpactObjectArray>
    <rdf:type rdf:resource="http://jazz.net/ns/ism/event/impact/policy/
ip1/">
  </rdf:Description>
</rdf:RDF>

```

To obtain resources that contain member lists, like the member lists contained in the results of the DirectSQL and GetByFilter policy functions, use a combination of the oslc.properties and the oslc.select query parameters. For example:

```

http://<server>:9080/NCICLUSTER_NCI_oslc/policy/
ip1?oslc.properties=*&oslc.select=test_ip1:myGetByFilter
{myGetByFilter:LNAME,myGetByFilter:FNAME}

```

This URL returns the following RDF:

```

<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:myGetByFilter="http://jazz.net/ns/ism/event/impact/policy/
ipl/myGetByFilter/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:direct="http://policy.js/xmlns/directSQL/"
  xmlns:test_ipl="http://jazz.net/ns/ism/event/impact/policy/ipl/"
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:impact="http://jazz.net/ns/ism/event/impact#"
  xmlns:javascript="http://policy.js/xmlns/">
  <rdf:Description rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/
policy/ipl">
    <test_ipl:myTimestamp>20120818</test_ipl:myTimestamp>
    <test_ipl:myString>test_ipl</test_ipl:myString>
    <test_ipl:myInteger>55</test_ipl:myInteger>
    <test_ipl:myGetByFilter>
      <test_ipl:myGetByFilter
rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myGetByFilter"
      <rdfs:member>
        <test_ipl:myGetByFilter
rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myGetByFilter/
item;FNAME=&apos;Mika&apos;;LNAME=&apos;Masion&apos;">
          <myGetByFilter:LNAME>Masion</myGetByFilter:LNAME>
          <myGetByFilter:FNAME>Mika</myGetByFilter:FNAME>
        </test_ipl:myGetByFilter>
      </rdfs:member>
    </rdfs:member>
    <test_ipl:myGetByFilter
rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myGetByFilter/
item;FNAME=&apos;Kevin&apos;;LNAME=&apos;Doe&apos;">
      <myGetByFilter:LNAME>Doe</myGetByFilter:LNAME>
      <myGetByFilter:FNAME>Kevin</myGetByFilter:FNAME>
    </test_ipl:myGetByFilter>
  </rdfs:member>
</rdfs:member>
    <test_ipl:myGetByFilter
rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myGetByFilter/
item;FNAME=&apos;Todd&apos;;LNAME=&apos;Bishop&apos;">
      <myGetByFilter:LNAME>Bishop</myGetByFilter:LNAME>
      <myGetByFilter:FNAME>Todd</myGetByFilter:FNAME>
    </test_ipl:myGetByFilter>
  </rdfs:member>
</rdfs:member>
    <test_ipl:myGetByFilter
rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myGetByFilter/
item;FNAME=&apos;Miriam&apos;;LNAME=&apos;Masters&apos;">
      <myGetByFilter:LNAME>Masters</myGetByFilter:LNAME>
      <myGetByFilter:FNAME>Miriam</myGetByFilter:FNAME>
    </test_ipl:myGetByFilter>
  </rdfs:member>
</rdfs:member>
    <test_ipl:myGetByFilter
rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/policy/ipl/myGetByFilter/
item;FNAME=&apos;Brian&apos;;LNAME=&apos;Doe&apos;">
      <myGetByFilter:LNAME>Doe</myGetByFilter:LNAME>
      <myGetByFilter:FNAME>Brian</myGetByFilter:FNAME>
    </test_ipl:myGetByFilter>
  </rdfs:member>
</rdfs:member>
    <oslc:ResponseInfo>
      <oslc:ResponseInfo rdf:about="http://<server>:9080/NCICLUSTER_NCI_oslc/
policy/ipl/myGetByFilter<oslc.paging=true&oslc.pageSize=100">
        <oslc:totalCount>5</oslc:totalCount>
      </oslc:ResponseInfo>
    </oslc:ResponseInfo>
  </test_ipl:myGetByFilter>
</test_ipl:myGetByFilter>

```

```
<test_ip1:myDouble>109.5</test_ip1:myDouble>
<rdf:type rdf:resource="http://jazz.net/ns/ism/event/impact#
policy/ip1/" />
</rdf:Description>
</rdf:RDF>
```

RDF functions

You can use RDF functions to make Netcool/Impact compatible with open services for lifecycle collaboration (OSLC).

RDFModel

You can use the `RDFModel` function to create an RDF model without any runtime parameters.

To create an empty RDF model, you call the `RDFModel` function without entering any runtime parameters. The function returns an empty RDF model.

Syntax

The `RDFModel` function has the following syntax:

```
[Model =] RDFModel()
```

Parameters

The `RDFModel` function has no runtime parameters.

RDFModelToString

You can use the `RDFModelToString` function to export an RDF model to a string in a particular language.

When you create or write an RDF model, you can use the `RDFModelToString` function to export a model to a string in a particular language. You can define a model object and a string that contains the language that is used as runtime parameters. If the language string is null or an empty string, the default language RDF/XML is used. The following language strings are supported:

- RDF/XML
- RDF/XML-ABBREV
- TURTLE
- TTL
- N3

`RDFModelToString` returns a string

Syntax

The `RDFModelToString` function has the following syntax:

```
[String =] RDFModelToString (Model, Language)
```

Parameters

The `RDFModelToString` function has the following parameters:

Table 82. `RDFModelToString` function parameters.

Parameter	Type	Description
Model	Model	Model object to output
Language	String	Language type

The following example updates the namespaces in a model:

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName,FormParameters, FilesToSend, HeadersToSend,
HttpProperties);
//Create Model from RDF payload
rdf=RDFParse(x);

//Retrieve all statements from model
allStatements=RDFSelect(rdf,null,null,null);

//Output RDF to log using N3
log(RDFModelToString(rdf, "N3"));

//Output RDF to log using the default language (RDF/XML)
log(RDFModelToString(rdf, null));
```

RDFModelUpdateNS

You can use the `RDFModelUpdateNS` function to insert, update, or remove a namespace from an RDF model.

When you create an RDF model, you can use the `RDFModelUpdateNS` function to insert, update, or remove a namespace from the model. You can define a model object, prefix string, and a URI string as runtime parameters. If the URI is null or an empty string, the function removes the prefix string from the model. If the URI contains a string with a non-empty value and the prefix exists, the URI is updated. If the prefix does not exist, a new prefix and URI is added to the model. `RDFModelUpdateNS` returns this model.

Syntax

The `RDFModelUpdateNS` function has the following syntax:

```
[Model =] RDFModelUpdateNS (Model, Prefix, URI)
```

Parameters

The `RDFModelUpdateNS` function has the following parameters:

Table 83. RDFModelUpdateNS function parameters

Parameter	Type	Description
Model	Model	Model object to update
Prefix	String	Contains the prefix to be updated in the model
URI	String	Contains the URI to associate with prefix

The following example updates the namespaces in a model:

```
//Create model
model = RDFModel();

//Update or insert namespace to model
RDFModelUpdateNS(model,"oslc","http://open-services.net/ns/core#");
RDFModelUpdateNS(model,"rdfs","http://www.w3.org/2000/01/rdf-schema#");
RDFModelUpdateNS(model,"dcterms","http://purl.org/dc/terms/");
```

The following piece of code deletes an existing model's namespace:

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey,
Method, AuthHandlerActionTreeName, FormParameters,
FilesToSend, HeadersToSend, HttpProperties);

//Create Model from RDF payload
model=RDFParse(x);
```

```
//Delete namespace from model that has prefix 'oslc'
RDFModelUpdateNS(model, "oslc", null);
```

RDFNodeIsResource

You can use the `RDFNodeIsResource` function to help other functions read and parse objects that are also an RDF resource. You can define an RDF node as a runtime parameter in this function. If the object is an RDF resource, the function returns a true value. If the object is an RDF literal, the function returns a false value. Other functions can use the model returned by the `RDFNodeIsResource` function to continue reading and parsing the RDF object.

Syntax

The `RDFNodeIsResource` function has the following syntax:

```
[Boolean =] RDFNodeIsResource (Object)
```

Parameters

The `RDFNodeIsResource` function has the following parameters:

Table 84. RDFNodeIsResource function parameters

Parameter	Type	Description
Object	RDF node	RDF object type check

The following example shows statements based on an RDF that is retrieved by the `GetHTTP` function:

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName, FormParameters, FilesToSend, HeadersToSend,
HttpProperties);

//Create Model from RDF payload
rdf=RDFParse(x);

//Retrieve all statements from model
allStatements=RDFSelect(rdf,null,null,null);

//Output subject, predicate, and objects from all statements returned, whose
object is a literal, to the log

Size=Length(allStatements);
log(Size);
Count=0;
While (Count < Size) {
    if (!RDFNodeIsResource (allStatements [Count].object)) {
        log (allStatements [Count].subject + " " + allStatements [Count].predicate + " "
+ allStatements [Count].object + ".");
    }
    Count = Count + 1;
}
```

RDFNodeIsAnon

You can use the `RDFNodeIsAnon` function to assist in reading and parsing an RDF. The `RDFNodeIsAnon` takes in a subject/object containing an `RDFNode` as a runtime parameter and returns true if the resource is anonymous. If the return value is false, the RDF resource is not anonymous. The model generated by the function can then be used by other functions to continue reading and parsing the RDF.

`RDFNodeIsAnon` returns true or false, depending if the `RDFNode` is anonymous

Syntax

The `RDFNodeIsAnon` function has the following syntax:

```
[Boolean =] RDFNodeIsAnon (Node)
```

Parameters

The `RDFNodeIsAnon` function has the following parameter:

Table 85. RDFNodeIsAnon parameter

Parameter	Type	Description
Node	RDFNode	Subject or object to check for

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="http";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=newobject();
FilesToSend=newobject();
HeadersToSend=newobject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName,FormParameters, FilesToSend, HeadersToSend,
HttpProperties);

//Create Model from RDF payload
rdf=RDFParse(x);

//Retrieve all statements from model
allStatements=RDFSelect(rdf,null,null,null);

//Output subject, predicate, and objects from all statements returned,
whose object is a literal, to the log

Size=Length(allStatements);
Log(Size);
Count=0;
While (Count < Size) {
    if (!RDFNodeIsAnon(allStatements[Count].subject)) {
        log (allStatements [Count].subject + " " + allStatements [Count].predicate + " "
+ allStatements
[Count].object + ".");
    }
    Count = Count + 1;
}
```

RDFParse

You can use the `RDFParse` function to help other functions read and parse an RDF object. It retrieves the data from a string that contains an RDF payload and returns a model that contains the RDF payload passed to it. Other functions can use this model to further read and parse an RDF object.

Syntax

The `RDFParse` function has the following syntax:

```
[Model =] RDFParse(Payload)
```


Parameters

The RDFParse function has the following parameters:

Table 86. RDFParse function parameter

Parameter	Type	Description
Payload	String	Payload containing the RDF

The following example provides statements based on an RDF that is retrieved by the GetHTTP function:

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName, FormParameters, FilesToSend, HeadersToSend,
HttpProperties);

//Create Model from RDF payload
rdf=RDFParse(x);
```

RDFRegister

You can use the RDFRegister function to help you to register service providers or OSLC resources with the registry server.

Before you can register a service provider or resource, you must use the other RDF policy functions to build an RDF model that meets the OSLC and Registry Services requirements.

After you build the RDF model, use the RDFRegister function to register the RDF with the resource registry contained in the Registry Services integration service.

If the service provider or OSLC resource is registered successfully, the RDFRegister function returns the resource location of the registration record. The following variables and their return values are also returned to provide more information:

- ResultCode contains the result code for the response.
- HeadersReceived contains the headers received in the response.
- HeadersSent contains the headers sent in the response.
- ResponseBody contains the response body text.

If the query parameters are set in the URL and you use the RDFRegister policy function to register a service provider, you must manually add the location of the service provider to the policy. For example:

```
RDFStatement(newModel, manu[0].subject, "http://open-services.net/ns/
core#serviceProvider", serviceProviderURL, true);
```

If you use the query string inside the path, you must also ensure that the `FormParameters` parameter is set to null. For example:

```
FormParameters=null;
```

Finally, you must ensure that the policy contains pagination information. For example:

```
Path="/NCICLUSTER_NCI_oslc/data/mysql1?oslc.paging=true&oslc.pageSize=100";
```

If unsuccessful, the return value of the resource location registration record is null. Error code information is returned in the `ErrorReason` and `ResultCode` variables.

Syntax

The `RDFRegister` function has the following syntax:

```
[ String =] RDFRegister(URI, Username , Password, Model)
```

where `Username` can be a null or void string to specify that no authentication is required.

Parameters

The `RDFRegister` function has the following parameters:

Table 87. RDFRegister function parameters

Parameter	Type	Description
URI	String	Registry Services server creation factory URI
Username	String	User name for the Registry Services server
Password	String	Password for the Registry Services server
Model	Model	Model that contains the RDF

Example

The following example manually registers a service provider and a set of resources that have been exposed by the OSLC server provider in Netcool/Impact.

The Registry Services server information is as follows:

```
RegistryServerProviderCFUri="http://<registry_services_server>:  
9080/oslc/pr/collection";  
RegistryServerResourceCFUri="http://<registry_services_server>:  
9080/oslc/rr/registration/collection";  
RegistryServerUsername="system";  
RegistryServerPassword="manager";
```

The Netcool/Impact server information is as follows:

```
HTTPHost="<impact_server>";  
HTTPPort=9080;  
Protocol="http";  
Path1="/NCICLUSTER_NCI_oslc/provider/provider01";  
Path2="/NCICLUSTER_NCI_oslc/data/computer";  
ChannelKey="";  
Method="GET";  
AuthHandlerActionTreeName="";
```

```

FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
HttpProperties.AuthenticationScheme="basic";

```

Get the service provider RDF from Netcool/Impact:

```

serviceProviderResponse=GetHTTP(HTTPHost,HTTPPort, Protocol, Path1,
ChannelKey,Method, AuthHandlerActionTreeName, null, FilesToSend,
HeadersToSend,HttpProperties);

```

Create an RDF model that is based on the service provider response:

```

serviceProviderModel=RDFParse(serviceProviderResponse)

```

Register the service provider in the provider registry:

```

serviceProviderURL = RDFRegister(RegistryServerProviderCFUri,
RegistryServerUsername, RegistryServerPassword,serviceProviderModel);
log("Provider Registry-Service Provider URL: " + serviceProviderURL);

```

Get all the computer system resources from Netcool/Impact:

```

allResources=GetHTTP(HTTPHost,HTTPPort, Protocol, Path2, ChannelKey,
Method,AuthHandlerActionTreeName, null, FilesToSend, HeadersToSend,
HttpProperties);

```

Create an RDF model that is based on the resource response:

```

allResourceModel=RDFParse(allResources);

```

Register each computer system and a set of properties with the resource registry:

```

statements=RDFSelect(allResourceModel, null, "http://jazz.net/ns/ism/
events/impact/data/computer/ID", null);
size=Length(statements);
count=0;
while(count<size) {
Path3=statements[count].subject;
//Get the individual computer system resource
resourceResponse=GetHTTP(HTTPHost,HTTPPort, Protocol, Path3, ChannelKey,
Method,AuthHandlerActionTreeName, null, FilesToSend, HeadersToSend,
HttpProperties);
resourceModel=RDFParse(resourceResponse);

```

Create a model that contains the properties and data that you want to register:

```

newModel=RDFModel();
manu=RDFSelect(resourceModel, null, "http://open-services.net/ns/
crtv#manufacturer",null);
model=RDFSelect(resourceModel, null, "http://open-services.net/ns/
crtv#model", null);
serial=RDFSelect(resourceModel, null, "http://open-services.net/ns/
crtv#serialNumber", null);
RDFModelUpdateNS(newModel, "crtv", "http://open-services.net/ns/crtv#");
RDFModelUpdateNS(newModel, "oslc","http://open-services.net/ns/core#");
RDFStatement(newModel, manu[0].subject, "http://www.w3.org/1999/02/
22-rdf-syntax-ns#type",
"http://open-services.net/ns/crtv#ComputerSystem", true);
RDFStatement(newModel, manu[0].subject, manu[0].predicate, manu[0].object,
RDFNodeIsResource(manu[0].object));
RDFStatement(newModel, manu[0].subject, model[0].predicate, model[0].object,
RDFNodeIsResource(manu[0].object));
RDFStatement(newModel, manu[0].subject, serial[0].predicate,
serial[0].object, RDFNodeIsResource(manu[0].object));

```

Update the model with the service provider location:

```
RDFStatement(newModel, manu[0].subject, "http://open-services.net/ns/  
core#serviceProvider", serviceProviderURL, true);
```

Register the resource in the resource registry:

```
resourceURL = RDFRegister(RegistryServerResourceCFUri,  
RegistryServerUsername, RegistryServerPassword, newModel);  
log("Resource Registry-Resource URL: " +resourceURL);  
  
count=count+1;  
}
```

RDFUnRegister

To remove the registration record of a service provider or resource from the registry server, use the `RDFUnRegister` function to supply the location of the registration record, the Registry Services server username and password, and the registration record that you want to remove.

Before you can remove the registration record of a service provider, you must remove all the registration records for the associated OSLC resources.

If successful, the `RDFUnRegister` function returns the message code 204 and the value `true`. The following variables and their return values are also returned to provide additional information:

- `ResultCode` contains the result code for the response.
- `HeadersReceived` contains the headers received in the response.
- `HeadersSent` contains the headers sent in the response.
- `ResponseBody` contains the response body text.

If unsuccessful, the return value of the resource location registration record is `false`. Error code information is returned in the `ErrorReason` and `ResultCode` variables.

Syntax

The `RDFUnRegister` function has the following parameters:

```
[ String =] RDFUnRegister(URI, Username , Password)
```

where `Username` can be a null or void string to specify that no authentication is required.

Parameters

Table 88. *RDFUnRegister* function parameters

Parameter	Type	Description
URI	String	Location that contains the registration record for the resource or service provider
Username	String	User name for the Registry Services server
Password	String	Password for the Registry Services server

Example of how to remove the registration of a service provider

The following example demonstrates how to remove the registration of the service provider.

The service provider location is:

`http://<registryserver>:9080/oslc/providers/6577`

Use the `RDFUnRegister` function to remove the registration. For example:

```
//Registry server information
ServiceProviderUri="http://<registryserver>:9080/oslc/
providers/6577";
RegistryServerUsername="system";
RegistryServerPassword="manager";
result = RDFUnRegister(ServiceProviderUri, RegistryServerUsername,
RegistryServerPassword);
```

Example of how to remove the registration of an OSLC resource

The following example demonstrates how to use the policy function to remove the registration of an OSLC resource.

```
registrationURL = "http://nc004075.romelab.it.ibm.com:16310/oslc/registration/
1351071987349";
providerURL = "http://nc004075.romelab.it.ibm.com:16310/oslc/providers/
1351071987343";
RegistryServerUsername="sadmin";
RegistryServerPassword="tbsm01bm";
```

```
returnString = RDFUnRegister (registrationURL, RegistryServerUsername,
RegistryServerPassword);
```

RDFSelect

You can use the `RDFSelect` function to assist in reading and parsing an RDF. To retrieve statements based on an RDF model, you call the `RDFSelect` function and pass the RDF model that is created by the `RDFParse` function. You can filter based on subject, predicate, and object.

The `RDFSelect` function returns an array of statements that are based on the filter, retrieving values for the subject, predicate, and object variables. You can use it to create RDF statements or triples. You can also use it to filter statements. If you do not want to filter your results, you specify null or empty values for the runtime parameters.

Syntax

The `RDFSelect` function has the following syntax:

```
[Array =] RDFSelect(Model, Subject, Predicate, Object)
```

Parameters

The `RDFSelect` function has the following parameters:

Table 89. RDFSelect function parameters

Parameter	Type	Description
Model	Model	The model that contains the RDF payload

Table 89. *RDFSelect* function parameters (continued)

Parameter	Type	Description
Subject	String	Filters for the subject value in RDF statements
Predicate	String	Filters for the predicate value in RDF statements
Object	String	Filters for the object value in RDF statements

The following example provides statements based on an RDF that is retrieved by the GetHTTP function:

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName, FormParameters, FilesToSend, HeadersToSend,
HttpProperties);

//Create Model from RDF payload
rdf=RDFParse(x);

//Retrieve all statements from model
allStatements=RDFSelect(rdf,null,null,null);

//Output subject, predicate, and objects from all statements returned to the log
Size=Length(allStatements);
log(Size);
Count=0;
While (Count < Size) {
    log (allStatements [Count].subject + " " + allStatements [Count].predicate + "
    " + allStatements [Count].object + ".");
    Count = Count + 1;
}
```

The following piece of code provides all statements that contain a particular subject name:

```
//Retrieve the RDF from OSLC provider through the GetHTTP method
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
```

```

x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName, FormParameters, FilesToSend,
HeadersToSend, HttpProperties);

//Create Model from RDF payload
rdf=RDFParse(x);

//Retrieve statements containing subject from model
statements=RDFSelect(rdf,"
http://ibm.com/ns/netcool-impact/data/SCR_Components#MYCLASS",null,null);

//Output subject, predicate, and objects from all statements returned to the log
Size=Length(stmt);
log(Size);
Count=0;
While (Count < Size) {
    log (stmt[Count].subject + " " + stmt[Count].predicate + " " +
stmt[Count].object + ".");
    Count = Count + 1;
}

```

RDFStatement

You can use the `RDFStatement` function to create and add statements to an RDF model.

You specify the following parameters in the function:

- Model object
- Subject string or resource
- Predicate string or property
- Object string or RDF node

If the Object runtime parameter is a string, you must specify a flag to determine whether the object input parameter is RDF literal or an RDF resource type

To create an anonymous resource in the statement, define the value for the Subject as null. When this parameter value is set to null, the policy function creates an anonymous resource in the statement.

Syntax

The `RDFStatement` function has the following syntax:

```
[Statement =] RDFStatement (Model, Subject, Predicate, Object, isResource)
```

Parameters

If the Object runtime parameter is a string, you must specify the `isResource` parameter. The `RDFStatement` function has the following parameters:

Table 90. RDFStatement function parameters

Parameter	Type	Description
Model	Model	Model object that the statement is added to.

Table 90. *RDFStatement* function parameters (continued)

Parameter	Type	Description
Subject	String, resource, or null	Subject value of statement. If this parameter is set to null, the function creates an anonymous resource in the statement. The policy function returns a statement instead of a model.
Predicate	String or property	Predicate value of statement.
Object	String or RDF node	Object value of statement.
isResource	Boolean	Determines whether an object is a resource or a literal.

The following example shows how to create a basic RDF with a single statement.

1. Use the `RDFModel` policy function to create a model:

```
Model = RDFModel();
RDFModelUpdateNS(model,"oslc","http://open-services.net/ns/core#");

subject = "http://ibm.com/ns/netcool-impact/data/SCR_Components#MYCLASS";
property = "http://open-services.net/ns/core#name";
value = "Brian";
isResource = false;
```

2. Use the `RDFStatement` policy function to create a statement:

```
RDFStatement(model,subject,property,value,isResource);
```

3. Finally, specify how the RDF model is output:

```
body = RDFModelToString(model, null);
```

4. Finally, specify how the RDF model is output:

```
body = RDFModelToString(model, null);
```

The following example shows how to create a model that is based on an existing model and that uses only the subjects that the user is interested in:

1. Use the `GetHTTP` method to retrieve the RDF from the OSLC provider:

```
HTTPHost="omega02.tivlab.austin.ibm.com";
HTTPPort=9081;
Protocol="https";
Path="/NCICLUSTER_NCI_oslc/data/resourceShapes/alerts";
ChannelKey="tom";
Method="";
AuthHandlerActionTreeName="";
FormParameters=NewObject();
FilesToSend=NewObject();
HeadersToSend=NewObject();
HttpProperties = NewObject();
HttpProperties.UserId="tipadmin";
HttpProperties.Password="passw0rd";
x=GetHTTP(HTTPHost, HTTPPort, Protocol, Path, ChannelKey, Method,
AuthHandlerActionTreeName, FormParameters, FilesToSend,
HeadersToSend, HttpProperties);
```

2. Create the RDF model from the RDF payload:

```
rdf=RDFParse(x);
```

3. Define a subject to filter:

```
mySubject=" http://ibm.com/ns/netcool-impact/data/SCR_Components#ID";
```


4. Retrieve all the statements that contain mySubject from the model:
`allStatements=RDFSelect(rdf,mySubject,null,null);`
5. Use the `RDFModel` function to create a new model:
`newModel = RDFModel()`
6. Use the `RDFModelUpdateNS` function to add the required namespaces to the model:
`RDFModelUpdateNS(newModel,"oslc","http://open-services.net/ns/core#");`
`RDFModelUpdateNS(newModel,"rdfs","http://www.w3.org/2000/01/rdf-schema#");`
`RDFModelUpdateNS(newModel,"dcterms","http://purl.org/dc/terms/");`
7. Use the `RDFStatement` function to add the statements from the old model to the new model
`Size=Length(stmt);`
`Count=0;`
`While (Count < Size) {`
`RDFStatement(newModel, stmt[Count].subject, stmt[Count].predicate,`
`stmt[Count].object,IsRDFNodeResource(stmt[Count].object));`
`Count = Count + 1;`
`}`
8. Output the new model to the log:
`log(RDFModelToString(model, null));`

Chapter 16. Service Level Objectives (SLO) Reporting

Service Level Objectives (SLO) Reporting is an optional feature that you can set up in the existing Netcool/Impact 6.1.1 product. The SLO Reporting package provides policies and schema files, which you can use to store Service Level Metrics in a DB2 database. These metrics are used to generate a report.

SLO Reporting uses Tivoli Common Reporting to develop, create, and generate reports. Tivoli Common Reporting consists of data stores, reporting engines, their corresponding web user interfaces displayed in Dashboard Application Services Hub, and a command-line interface. For more, information about Tivoli Common Reporting, see the Jazz for Service Management documentation.

Configuring SLO reporting

The SLO reporting function can be configured in 2 ways:

SLO reporting for TBSM

This feature is primarily intended for use with Tivoli Business Service Manager (TBSM). Use the feature to create SLO reports that display the outage times for business services in TBSM.

SLO reporting for third-party data

You can also configure SLO reporting to report on data from a third-party database.

To configure SLO reporting, you need to complete the following steps:

1. Create a database that is called SLORPRT in your DB2 database.
2. Install the SLO reporting features.
3. Define the service definitions in the service definition properties file.
4. Define the business calendar in the business calendar properties file. This is optional.
5. Run the createServiceDefinition policy to create the service definition.
6. Create policies to retrieve the outage data from the specified data source and record it in the SLORPRT database. If you want to configure SLO reporting with TBSM, you can use the sample policies that are provided.
7. Deploy the SLO Availability report in Tivoli Common Reporting.

Architecture

SLO consists of 3 components:

- The SLORPRT database that you use to store the SLO definitions and outage data that is gathered for those definitions.
- Projects in Netcool/Impact that contain functions for interacting with the SLORPRT database.
- The SLO Availability report definition for Tivoli Common Reporting.

You need to decide which of your business services that you want to report on and the metrics that need to be associated with the services. These metrics are used in the availability report in Tivoli Common Reporting.

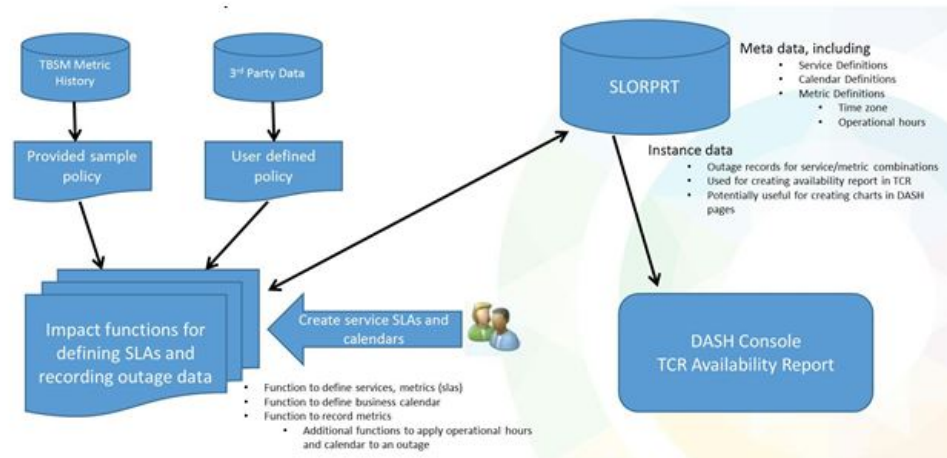
First, you define the business services that are displayed in the report in the SLORPRT database.

Next you can either use the sample policies, or define your own policies, to collect, analyze, and record outage data in the SLORPRT database.

The SLO Availability Report uses the outage data to display the availability for a specified period.

The following graphic illustrates this architecture:

Architecture



SLO terminology overview

Before you start to configure SLO reporting, read the following terms to help you to understand the solution.

Business service

A business service is an aspect of your enterprise, like Internet Banking and Payroll, that is defined in Tivoli Business Service Manager. You choose one or more of these business services to report on.

Service definitions

Service definitions are used to specify metadata for the service and metric combinations. Service definitions are defined in Impact. The SLO Availability report in Tivoli Common Reporting uses the service definition for reporting.

Service Level Agreements (SLAs)

A service level agreement defines a metric and a set of operational hours where the service is required to be available. The SLAs are defined as part of a service definition.

Metric A measurable property for a service. For example, you can create metrics to monitor downtime or transaction volume. The SLO Availability report in Tivoli Common Reporting uses the metric to display the availability of the business service based on the metrics that you specify. A metric is specified in an SLA definition.

Operational hours

The operational hours defines the hours during which the business service is expected to be available. The availability report summarizes the

percentage of time the service was available during the operational hours. Operational hours are defined as part of the service or SLA definition.

Calendar

A calendar indicates the days in a year that are holidays and weekends. If a calendar is defined as part of a service definition, outages that occur in these periods are recorded separately in the SLORPRT database. Calendars are specified in a properties file that is passed to an Impact policy that stores the definition.

SLO reporting prerequisites

Before you install the SLO Reporting package, complete the prerequisites.

- Netcool/Impact 6.1.1.4 must be installed and configured before the SLO Reports package can be applied. The Reports package extensions are a set of policies, data source, and data type configurations.
- The version of DB2 must be 9.7 Fix pack 4 or later, which is available as a bundle with Netcool/Impact.
- Tivoli Common Reporting version 2.1 or higher is required to install the SLO Reports package.
- You must create a database named SLORPRT in the DB2 system and catalog the TCPIP remote node and the remote database. If a local version of DB2 does not exist on the Tivoli Common Reporting (TCR) server, install the IBM DB2 Connect Server that is available as part of the fix pack installers for DB2 Server, Enterprise and Workstation.

Installing and enabling SLO report package

How to install and enable the Netcool/Impact SLO extensions in Tivoli Common Reporting.

Before you begin

- Netcool/Impact 6.1.1.4 must be installed and configured before the SLO Reports package can be applied. The Reports package extensions are a set of policies, data source, and data type configurations.
- The version of DB2 must be 9.7 Fix pack 4 or later, which is available as a bundle with Netcool/Impact.
- Tivoli Common Reporting version 3.1 or higher is required to install the SLO Reports package.
- You must create a database named SLORPRT in the DB2 system and catalog the TCPIP remote node and the remote database. If a local version of DB2 does not exist on the Tivoli Common Reporting (TCR) server, install the IBM DB2 Connect Server that is available as part of the fix pack installers for DB2 Server, Enterprise and Workstation.

About this task

The SLO Reports package is in the `install_home/impact/add-ons/slo` directory.

The `importData`, `sloutil`, `db`, and `Report` directories are included in the SLO Reports package.

Procedure

1. Create the SLORPRT database in DB2. For example:

```
db2 create database SLORPRT using CODESET UTF-8 territory en-US
```

If a local version of DB2 does not exist on the Tivoli Common Reporting server, install the IBM DB2 Connect Server that is available as part of the fix pack installers for DB2 Server, Enterprise and Workstation.
2. The db directory contains an sql file. Connect to the database and add this file to the same system where the SLORPRT database is created. For example:

```
db2 connect to SLORPRT
db2 -tvf slo_dbschema.sql
db2 connect reset
```
3. Import the **SLA** and **SLA_Utility** projects.
Navigate to `<TBSM_HOME>/bin` and run the following command:

```
nci_import.[bat/sh] <servername> IMPACT_HOME/add-ons/slo/importData
```

For example:

```
./nci_import TBSM /opt/IBM/tivoli/impact/add-ons/slo/importData
```

Then run the following command:

```
nci_import.[bat/sh] <servername> IMPACT_HOME/add-ons/slo/sloutility
```

For example:

```
./nci_import TBSM /opt/IBM/tivoli/impact/add-ons/slo/sloutility
```
4. The Report directory contains the model to be used in the Framework Manager in Tivoli Common Reporting. The model is provided for your reference. Use the model if you want to extend the schema or create more views for the report.
5. The Report package contains the package that must be imported in to the Tivoli Common Reporting Server. The package contains the sample report and the queries that can be used to generate the reports. Complete the following steps to import the Netcool/Impact SLO reports package into Tivoli Common Reporting version 3.1.
 - a. Navigate to the Tivoli Common Reporting bin directory. For example, `/opt/IBM/JazzSM/reporting/bin`.
 - b. Use the **trcmd** command to create a data source for the SLORPRT database:

```
trcmd.sh -user <TCR user> -password <TCR password>
-datasource -add SLORPRT -connectionString <db2 connection string>
-dbType DB2 -dbName <database name>
-dbLogin <database username> -dbPassword <database password>
```

For example:

```
./trcmd.sh -user tipadmin -password password1 -datasource -add SLORPRT
-connectionString jdbc:db2://server.ibm.com:50000/SLORPRT
-dbType DB2 -dbName SLORPRT -dbLogin db2inst1 -dbPassword password1
```
 - c. Use the **trcmd** command to import the Netcool/Impact SLO report package:

```
./trcmd.sh -import -bulk <file> -user <TCR User>
-password <TCR password>
```

For example:

```
./trcmd.sh -import -bulk /tmp/ImpactSLOReportPackage.zip -user smadmin
-password password2
```
6. In Netcool/Impact, configure the **SLOReportDatasource** and **SLOUtilityDatasource** data sources to access the SLORPRT database that you

created in step 1. The **SLOReportDatasource** is available in the **SLA** project in Netcool/Impact, and the **SLOUtilityDatasource** is available in the **SLA_Utility** project.

What to do next

The SLO reporting package is installed and enabled. Next, you need to create the service definitions files.

Defining service definition properties

Before you can view reports for a service, you need to define the service definition parameters.

About this task

You specify the service definition parameters, like the SLA metric names and the operational hours, in a service definition properties file that you create. After you create the properties file, you need to run the `createServiceDefinition` policy and pass the service definition to it as an input parameter. When you run the policy, the service definition is implemented.

In some cases, you might want to reuse service definitions and metric names. You need to note the following logic:

- You can reuse the same service definition in multiple service definition properties files to define multiple SLAs. The properties that describe the service like the label and description, are specified by the last properties file used for the service.
- If you use the same SLA metric name in multiple service definitions, only the last settings are used, including any operational hours or time zone properties. Each service must include the complete definition for the SLA and it needs to match any previous definition.

For example service definition properties files, see “Properties files examples” on page 298.

Procedure

1. Log in to the server where the Impact Server server was installed.
2. Create a service definition properties file. Note the file name. You need to specify the file name as a parameter value in the policy that you use to implement the service definition properties.
3. Specify the parameters in the services definition file. For more information, see “Service definition properties file” on page 282. Service definition names and service level agreement (SLA) names must be unique in the SLORPRT database.
4. Save the file.

What to do next

After you define the service definition properties in the properties file, run the `createServiceDefinition` policy to implement the service definition. You need to specify the directory where you saved the file in step 2 in the **Service Definition File Name** parameter.

If you want to update properties of the service definitions, just update the properties file and rerun the createServiceDefinition policy to create the service definition. The new definition will replace the existing definition.

If you want to delete a service definition or an SLA defined in the service, refer to “SLO Utility Functions” on page 304.

Service definition properties file

Use the service definition properties file to define a service definition.

Naming conventions

Properties use the following naming conventions:

`<propertyName>.num`

This indicates that multiple sets of related properties are specified. For example, `sla.num=2` indicates that there are two sets of related properties. The related properties are named `sla.1.<propertyName>`, `sla.2.<propertyName>`, and so on.

To nest multiple properties, use `<propertyName>.n.<relatedPropertyName>.num`. For example, `sla.1.operationalHour.num=2` specifies two sets of operational hour values for the first SLA definition.

General properties

Table 91. General properties in the service definition properties file

Property	Description
serviceName	The name of the service for which the service level agreement (SLA) definitions are provided. The name must be unique. This is a required property.
label	Display name for the service. This is optional. If you do not specify a value, the value defaults to an empty string.
description	Description for the service. This is optional. If you do not specify a value, the serviceName value is used.
businessCalendar	Defines the name of the business calendar that is used by the SLAs defined for the service. If no calendar is specified, the outage time for the service is defined only as operational or non-operational. There is no holiday outage. For more information, see “Configuring business calendars” on page 287.

Operational hours properties

Table 92. Operational hours properties in the service definition properties file

Property	Description
operationalHour.num	Defines the number of operational hour periods that can be defined for the service. If this is not specified, at least 1 period is defined for the service. If the value is set, you need to specify a start and end time for the number of periods that are specified in this parameter.
operationalHourStartTime	The start time of a single operational hours period. You must specify values in the 24-hour clock format, for example, 13:00:00. If no value is specified, the default value 00:00:00 is assigned. This value is used for any SLAs that do not include operational hours.
operationalHourEndTime	The end time of a single operational hours period. You must specify values in the 24-hour clock format, for example, 17:00:00. If no value is specified, the default value 23:59:59 is assigned.
operationalHourStartTime.n	The start time of the operational period n. You must specify values in the 24-hour clock format, for example, 13:00:00. If this parameter is not specified, the operational hour period is not defined.
operationalHourEndTime.n	The end time of the operational period n. You must specify values in the 24-hour clock format, for example, 17:00:00. If this parameter is not specified, the operational hour period is not defined.

Identity Properties

Table 93. Identity properties in the service definition properties file

Property	Description
identity.num	Defines the number of identities that are defined for a service. If you specify a value for this property, you must specify the required values for the same number of identity types.
identity	An identity represents an alternative method for identifying a service when it gathers the outage data for the service. An identity is defined as <code>identityType:::identityString</code> . If you only specify a service name, the default identity is <code>tbsmIdentity:::servicename</code> . If no identity is specified, the service definition cannot be created. For TBSM services, use the instance name as the identity. The type defaults to <code>tbsmIdentity</code> .

Table 93. Identity properties in the service definition properties file (continued)

Property	Description
identity.n	<p>The identity type and string for identity n. This is required if you specify a value for the identity.num parameter. If any required identity is not included, the service definition cannot be created.</p> <p>For example, you can use multiple identities if the outage data is gathered from multiple data sources. In this example, you are receiving data from another monitoring system. You define the identities as:</p> <pre>identity.num=2 identity.1=internetBanking identity.2=MSN::IB001</pre> <p>where the identity type is MSN for Managed System Name (MSN). IB001 is the value that is specified for the MSN.</p>

SLA Properties

Table 94. SLA properties in the service definition properties file

Property	Description
sla.num	Defines the number of SLA definitions. If this property is set, you must specify a matching number of SLA definitions.
sla.name	<p>The name of the SLA. This is also used as the metric name that is listed as an option in the SLO Availability report. The name is required and must be unique.</p> <p>Note: If an SLA exists that uses the same name, the existing SLA is updated with the new properties based on the current service definition. No properties are inherited from the definition that is saved in the SLORPRT database.</p>
sla.n.name	<p>The name of the nth SLA in the service definition file. This is also used as the metric name that is listed as an option in the SLO Availability report. This parameter is required if the sla.num parameter is specified. The SLA definition is not used if the sla.n.name parameter is not defined.</p> <p>The name must be unique.</p> <p>Note: If an SLA exists that uses the same name, the existing SLA is updated with the new properties based on the current service definition. No properties are inherited from the definition that is saved in the SLORPRT database.</p>

Table 94. SLA properties in the service definition properties file (continued)

Property	Description
sla.displayName, sla.n.displayName	The label for the metric that is associated with the SLA. If you do not specify a value for the display name, the default empty string is used.
sla.description, sla.n.description	The description for the metric that is associated with this SLA. If no value is specified, the default empty string is used.
sla.timezone, sla.n.timezone	Time zone that is used by the SLA. If you do not specify a value, GMT is used by default. The time zone ID must be a valid value. For more information, see “Configuring the time zone.”
sla.operationalHourThreshold, sla.n.operationalHourThreshold	An availability threshold for operational hours in the SLA. You specify a numeric percentage, for example 98.5. The threshold value is displayed in the SLO Availability report. The default is zero.
sla.nonOperationalHourThreshold, sla.n.nonOperationalHourThreshold	An availability threshold for non-operational hours in the SLA. You specify a numeric percentage, for example 98.5. The threshold value is displayed in SLO Availability report. The default is zero.
sla.operationalHour.num	Defines the number of operational hour periods that are defined for the SLA. If a value is not specified, then at most 1 operational hour period is defined for the SLA.
sla.operationalHourStartTime, sla.operationalHourStartTime.n	The start time of a single operational hour period. You must specify the value in 24-hour clock format. For example, 08:00:00 for 8 AM.
sla.operationalHourEndTime, sla.operationalHourEndTime.n	The end time of a single operational hour period. You must specify the value in 24-hour clock format. For example, 17:00:00 for 5 PM.
sla.n.operationalHourStartTime, sla.n.operationalHourStartTime.n	The start time of a single operational hour period for the nth SLA. You must specify the value in 24-hour clock format. For example, 08:00:00 for 8 AM.
sla.n.operationalHourEndTime, sla.n.operationalHourEndTime.n	The end time of a single operational hour period for the nth SLA. You must specify the value in 24-hour clock format. For example, 17:00:00 for 5 PM.

For examples of using properties file, see “Properties files examples” on page 298.

Configuring the time zone

If a single business service operates in multiple time zones, you can specify a timezone in the SLA definition that is included in the service definition for the business service.

About this task

The time zone value is used to modify how the business calendar and the operational hours are interpreted by the service definition. Netcool/Impact includes utility functions that you can call to log information that helps you to choose the time zone you need.

This setting is optional. It is not required for the service definition to work.

Procedure

1. Specify the time zone value in the service definition properties file. For more information, see the documentation about the **sla.timezone** property in "Service definition properties file" on page 282.
2. To get information about the available time zone identifiers, create a policy that calls the utility functions.
3. Use the sample code to help you to create the policy that calls the utility function.

To log information about all the time zone IDs, add the following function:

```
/*
 * Log information for all time zones.
 */
function logAllTimeZoneInfo()
```

To log information about a specific time zone ID, add the following function:

```
/*
 * Log information for a specific time zone id
 */
function logTimeZoneInfo(<timezone_id>)
```

where <timezone_id> is the ID of the time zone.

To log information for all the time zone IDs that use a specific offset, add the following function:

```
/*
 * Log information for time zones with a specific offset from GMT.
 */
function logTimeZoneInfoForOffset( <hours> )
```

For example, you can use the following policy to load the utility function and log the timezone IDs:

```
/* This statement is required for the utility functions
 * to be loaded. */
Load("serviceDefinition");

Log( 0,"Starting policy timeZoneInfoLogger" );

/* Uncomment the function you need to help you pick
a time zone id */

/* Call a function to log information about all valid
 * time zones */
/* logAllTimeZoneInfo(); */

/* Call a function to log information about a specific
 * time zone.*/
/* Replace "TimeZoneID" with the ID for which you
 * need information. */
/* logTimeZoneInfo( "TimeZoneID" ); */

/* Call a function to log information about time zones
 * offset by n hours from GMT. Replace "offset" */
```

```
/* with a number of hours offset from GMT. For example,  
/* 5, +6, -3, -2.5 are valid offsets.*/  
/* logTimeZoneInfoForOffset( offset ); */
```

4. Save your changes.
5. Run the policy. The time zone information that you requested in the policy is listed in the `policylogger.log` file.

Configuring business calendars

You can use the business calendar feature to identify the holidays and weekends during a specific time period. This is optional.

About this task

Note: If you specified a time zone value in the SLA, Netcool/Impact uses this value to compare the outage times with the holiday and weekend values that are defined in the business calendar. For example, if you specify GMT as the time zone in the SLA, Netcool/Impact compares the time in GMT to the holiday and weekend times that are defined in the business calendar.

For more information about the properties that you can specify in a business calendar definition, see “Business calendar properties file” on page 289.

If you want to use common holiday and weekend values for use with multiple business calendars, you can create common business calendars. For more information, see “Creating common properties in business calendars.”

For examples of different types of business calendars, see “Properties files examples” on page 298.

Procedure

1. Create a file called `<business_calendar>.props`. Open the file.
2. Define a business calendar. The following example defines a business calendar called US. This business calendar defines 2 holidays and 2 weekend days:

```
calendar.name = US  
calendar.holidays.dateformat = MMM dd,yyyy  
calendar.holidays.num = 2  
calendar.holidays.1 = Jan 1,2016  
calendar.holidays.2 = Jul 4,2016  
calendar.weekends.num = 2  
calendar.weekends.1 = 1  
calendar.weekends.2 = 7
```

3. Save the file.
4. Run the `createBusinessCalendarDefn` policy. You need to use the **Business Calendar Definition Filename** parameter to pass the name of the file in the policy.

If you want to change the holiday or weekend days for a calendar, just update the properties file and rerun the `createBusinessCalendarDefn` policy to create the calendar. The new definition will replace the existing definition. If you want to delete a calendar, refer to “SLO Utility Functions” on page 304.

Creating common properties in business calendars

If you need several business calendars that share weekend days or holidays, you can define the common properties in a common calendar.

About this task

After you define the common calendar, you specify the name of the common calendar in the `calendar.duplicateCalendarName` property in the business calendar definition. After you specify the common calendar, you need to define the unique holiday and weekend days for the business calendar in a second business calendar.

Procedure

1. Define a calendar that is called `COMMON.US`. For example:

```
calendar.name=COMMON.US
calendar.holidays.dateformat= MMM dd,yyyy
calendar.holidays.num = 6
calendar.holidays.1 = Jan 1,2015
calendar.holidays.2 = Feb 14,2015
calendar.holidays.3 = Dec 25,2015
calendar.holidays.4 = Jan 1,2016
calendar.holidays.5 = Feb 14,2016
calendar.holidays.6 = Dec 25,2016
calendar.weekends.num = 2
calendar.weekends.1 = 1
calendar.weekends.2 = 7
```

2. Define a second business calendar to specify the unique holiday and weekend values. In the example, this business calendar specifies the holidays that are unique to the United States. For example, create a calendar that is called `US` that contains the following properties:

```
calendar.name=US
calendar.holidays.dateformat= MMM dd,yyyy
calendar.holidays.num = 2
calendar.holidays.1 = Jul 4,2015
calendar.holidays.2 = Jul 4,2016
```

3. To add another region, you need to create a common file. For example, create a calendar that is called `COMMON.Canada`. This calendar duplicates the properties that are specified in the `COMMON.US` file:

```
calendar.name=COMMON.Canada
calendar.duplicateCalendarName=COMMON.US
```

4. To specify the values for the holidays and weekend days, create another business calendar. For example, create a calendar that is called `Canada` that specifies the unique holidays for Canada:

```
calendar.name=Canada
calendar.holidays.dateformat= MMM dd,yyyy
calendar.holidays.num = 2
calendar.holidays.1 = Jul 1,2015
calendar.holidays.2 = Jul 1,2016
```

5. To add any other region, repeat steps 3 and 4.

Results

When the business calendar is used by an SLO service definition, the function checks the specified business calendar and the common version of the calendar.

For example, if the service definition includes the `businessCalendar=US` property, the policy function checks both the common calendar, `COMMON.US` and the calendar that specifies the unique values for the country, the `US` business calendar in the example. The function uses the values in both to calculate the holidays and weekend days.

Business calendar properties file

Use the business calendar properties file to specify the business calendars that are used in SLO reporting.

Table 95. Business calendar properties

Property	Description
calendar.name	<p>Specify the name of the business calendar that you want to define. If you do not specify a value for this property, the <code>createBusinessCalendarDefn</code> policy creates an exception.</p> <p>Some calendars are prefixed with <code>COMMON..</code>. For more information, see “Creating common properties in business calendars” on page 287.</p>
calendar.duplicateCalendarName	<p>If you want to create a duplicate of an existing calendar, specify the name of an existing calendar.</p> <p>If the specified duplicate does not exist, the <code>createBusinessCalendarDefn</code> policy creates an exception.</p>
calendar.holidays.dateformat	<p>Specify the date format that is used to specify the holiday dates for the calendar.</p> <p>For example, you can specify the date format as <code>MMM dd,YYYY</code> and specify New Year's day as <code>Jan 01,2015</code>. This parameter is required if you want to specify holidays in the calendar.</p> <p>You must use a valid format. These formats are specified by the <code>SimpleDateFormat</code> class in Java.</p> <p>If this property is omitted, no holidays are defined in the calendar.</p>
calendar.holidays.num	<p>Specify the number of holiday dates that are specified for the business calendar.</p> <p>This parameter is required if you want to specify holidays.</p> <p>If this property is omitted, no holidays are defined in the business calendar.</p>

Table 95. Business calendar properties (continued)

Property	Description
calendar.holidays.n	<p>Specify the date of the nth holiday that is specified for the business calendar. You need to use the date format that is specified in the calendar.holidays.dateformat parameter. For example, specify Jan 01,2015 for New Year's day.</p> <p>This parameter is required for each value of n up to the number of properties that are specified in the calendar.holidays.num parameter.</p> <p>If you do not specify a value for any of these parameters, the policy does not create holidays for the parameter.</p>
calendar.weekends.num	<p>Specify the number of weekend days for the business calendar. This parameter is required if you want to specify weekends in your business calendar.</p> <p>If you omit this parameter, no weekend days are defined in the business calendar.</p>
calendar.weekends.n	<p>Specify the date of the nth weekend day that is specified for the business calendar.</p> <p>This parameter is required for each value of n up to the number of properties that are specified in the calendar.weekends.num parameter.</p> <p>If you do not specify a value for any of these parameters, the policy does not create weekend days for the parameter.</p>

For examples of using properties file, see “Properties files examples” on page 298.

Retrieving SLA metric data

After you create the service definition file, you need to create policies to retrieve the SLA metric data. The package includes sample policies and functions that you can use to retrieve data from TBSM.

About this task

The SLO Reporting package includes sample policies that you can use to store outage data from TBSM. For more information, see “SLO reporting policies” on page 291.

You can also use a number of functions to help you to retrieve metric data. For more information, see “SLO reporting policy functions” on page 291.

Procedure

1. Define and implement the service definition and SLA definitions for the service:
 - Create the service definition properties file, including the SLA definitions.
 - To implement the business service definition, run the **createServiceDefinition** policy, configuring the policy to pass the service definition properties files as a parameter.
 - If you require business calendars, run the **createBusinessCalendarDefn** policy, configuring the business calendar properties files as a parameter.
2. Use a Netcool/Impact policy to record the metric information. Create a policy that uses the recordSLAMetric policy function. The db2GetData sample policy provides instructions and sample code that can help you create your policy. The Netcool/Impact policies are written in JavaScript language. Use the Load function to load the recordSLAMetric policy in to the data retrieval policy. To use the SLO functions, add the following two commands to the start of your policy

```
Load("slaDefGlobalSettings");
Load("recordSLAMetric");
```
3. Save the policy.

SLO reporting policies

The SLO reports package contains the following policies:

- **BusinessCalendar:** Provides functional support for business calendars.
- **createBusinessCalendarDefn:** Creates a business calendar that is based on a properties file.
- **createServiceDefinition:** Creates a service definition that is based upon a properties file.
- **recordSLAMetric:** Provides supporting functions that are used to record SLA metrics
- **serviceDefinition:** Provides functional support for service definitions
- **slaDefGlobalSettings:** Provides support for global settings
- **db2GetData:** Sample Policy
- **getDataFromTBMSAvailability:** Sample Policy
- **serviceDowntimeBasedOnTBMSStatusChange:** Sample Policy

SLO reporting policy functions

Use the following policy functions to help you to record outage data in the SLORPRT database.

The following policy functions are included as part of the SLO Reporting package:

- addCorrelationValue
- recordMetric
- addCheckpointValue
- getCheckpointValue
- addSLAMetricWithOperationalHoursAndBusinessCalendar

addCorrelationValue

The addCorrelationValue function records a correlation value for the SLA metric that is being recorded. The correlated value is stored as a string. The format of the string is defined by the user. The user is responsible for data maintenance.

Table 96. addCorrelationValue parameters

Parameter	Description
serviceName:	Service Name for the correlation value that is being stored.
CorrelationValue:	The user's value, which is stored in the database.
MetricName:	Metric Name the correlation value that is being stored.
TimeRecorded:	Time record for this correlation value.

The following examples show where the addCorrelationValue function can be used.

- Example 1:

The downtime of a service must be calculated based on the correlated value of the status of an event. When the event is generated (open state), record the Serial Number in the correlated value with the time recorded. When the event is updated with the close state, retrieve the correlated “open time” from the correlated table. Use the time recorded field as the “Creation time” and the current time as the resolved time.

- Example 2:

If you want to store data to be used in the report later, the addCorrelationValue function can be used. For example, the ticket number for which the service downtime is being recorded can be stored in this table. Using the timeRecorded field, service name, and the metric name, the user can generate a report of all the tickets that are associated with a SLA metric.

recordMetric

The recordMetric function records a single SLA metric. The SLA metric name must be defined during the services definition section.

Table 97. recordMetric function parameters

Parameter	Description
serviceName:	The service name for the SLA metric that is being recorded.
metricName:	The SLA Metric Name.
operValue:	The value that needs to be stored for operational hours.
nonOperValue:	The value that needs to be stored for non-operational hours.
holidayValue:	The value that needs to be recorded for holiday hours.
CreationTimeInSeconds:	The time when this metric was created or is recorded. The value must be in seconds from Jan 1 1970.
operHourResourceId:	Pass -1 for operHourResourceID parameters.

If you need to record the time for the SLA, always use the recordSLAMetricWithOperationalHoursAndBusinessCalendar function.

addCheckpointValue

The addCheckpointValue function adds a check point value for the solution. This value can be used to store check points while the retrieval of the source data is being processed.

Table 98. addCheckpointValue Parameters

Parameters	Description
serviceName:	Service Name for the check point that is to be stored.
MetricName:	Metric name for the check point value that is to be stored
Value:	The check point value that is to be recorded.

Example:

If the service downtime is based on the amount of time a ticket is opened, you can use the addCheckpointValue function to track the last record that is read from the source database. Store the last resolved time that was read from the database. The next query can use the value that is stored in the checkpoint database for the filter.

getCheckpointValue

The getCheckpointValue function is used to retrieve the checkpoint value for a service name and metric name. This function is used to get the value that was added by the addCheckpointValue function.

Table 99. getCheckpointValue Parameters

Parameter	Description
serviceName:	The service name for the checkpoint value to be retrieved from the table.
MetricName:	The metric name for the checkpoint value to be retrieved from the table.

addSLAMetricWithOperationalHoursAndBusinessCalendar

The addSLAMetricWithOperationalHoursAndBusinessCalendar function inserts downtime based on operational hours and the business calendar, if a business calendar is defined for the SLA associated with the service. If the business calendar is specified, then business calendar is applied. Similarly, if the operational hours time is specified the time is broken down by the operational and non-operational hours.

Note: The start and end time values must be passed as a GMT value. This ensures that it is calculated correctly based on the time zone property that is defined in the service's SLA definition.

Table 100. addSLAMetricWithOperationalHoursAndBusinessCalendar Parameters

Parameters	Description
ctime:	The start time for the outage to be recorded.
Rtime:	The end time for the outage to be recorded.

Table 100. *addSLAMetricWithOperationalHoursAndBusinessCalendar*
Parameters (continued)

Parameters	Description
serviceName:	The service name, as defined in the service definition properties file, for the metric name to be recorded. If there is an identity that is defined for that service, then the identity can be passed to the function. For more information about this property, see the entry for the identity property in "Service definition properties file" on page 282.
MetricName:	The metric name, as defined in the service definition properties file, for the SLA to be recorded.

Note: The values that you specify in the `ctime` and `Rtime` parameters are converted into the time zone that is defined in the SLA. If no time zone is specified, the default value, GMT, is used. Therefore, you may need to adjust the values that you use here, depending on the source of the data. For example, if you are using metric history data from Tivoli Business Service Manager and you use the default time zone in the SLA definition, you do not need to change anything because the source data is also calculated in GMT in Tivoli Business Service Manager.

Using the `getDataFromTBSMAvailability` sample policy

The sample policy `getDataFromTBSMAvailability` obtains the status change from the TBSM history database in TBSM to record the downtime for the service.

Procedure

1. In Netcool/Impact, create a data source, for example `TBSM_History` that connects to the TBSM metric history database where the status changes are stored. For more information about installing the TBSM metric history database, see the TBSM documentation available from the following URL: http://www-01.ibm.com/support/knowledgecenter/SSSPFK_6.1.1.3/com.ibm.tivoli.itbsm.doc/timewa_server/twa_metrics_c_intro.html
2. In Netcool/Impact, create a data type called `HISTORY_VIEW_METRIC_VALUE`, and associate this data type with the `HISTORY_VIEW_RESOURCE_METRIC_VALUE` view in the TBSM metric history database.
3. In Netcool/Impact, create a policy activator service to activate the `getDataFromTBSMAvailability` policy. For information about how to create a policy activator service, see "Policy activator service" on page 73.
The SLA project that is imported when you deploy the SLO add-on function includes a sample policy activator service called `GetData`. When this service is started, it activates the `db2GetData` sample policy on a 300-second interval. Use this example to help you to create your own policy activator service.
4. The `getDataFromTBSMAvailability` policy reads records that show changes in the TBSM status for the services identified by your SLO service definitions. An outage is calculated in one of two ways:
 - a. From the time a service is marked "Bad" in TBSM until the time the status changes to anything other than "Bad". This is the default behavior when you deploy the SLO feature for the first time.

- b. From the time a service is marked "Bad" in TBSM until the time the status changes to "Good". This is the behavior if you deployed the SLO feature prior to installing Fixpack 4.

The **getDataFromTBSMAvailability** policy can retrieve the outage times for active outages. Active outages are defined as those where the status is currently "Bad" and not yet resolved. The end time is recorded as the current time when the active outage is first recorded. Each subsequent run of the policy updates the outage with the current time as the updated end time, until the final outage time is recorded when the status either becomes "Good" or not "Bad", depending on how the policy is configured to run.

Configuring getDataFromTBSMAvailability

There are two different algorithms that can be used to decide when an outage starts and ends using the TBSM metric history data. The default if the SLO function is deployed after installing Fixpack 4 is that an outage is defined as only the time when the TBSM status is "Bad".

Prior to Fixpack 4, the algorithm calculated the outage from the time that the status first changes to "Bad" until the status becomes "Good", regardless of other statuses that might be reported by TBSM in the interim. This algorithm can still be used in case you already have SLO deployed and prefer this method.

Consider the following example:



The default algorithm with Fixpack 4 will record an outage time of $T4 - T3$, which is only the time period when the status was "Bad". The legacy algorithm will record the outage as $T5 - T3$, ignoring time when the status changed back to "Marginal".

You can use the **sloSetConfigurationValue** policy to ensure that you are using the algorithm you prefer. For more information on setting the SLO configuration properties, refer to "SLO Utility Functions" on page 304.

Reports

The application availability report package contains a sample application availability report.

Record the service downtime for the application. The availability report is generated at the entity level for all the applications. The report requests the service and the SLA that you want the report to run, and a date. The report template has the following options:

- A line chart which reports the availability for up to a month for the date you selected.
- A line report which reports the availability for up to a year up to the date you selected.
- The table contains the data for each application.

Example SLO reporting configuration

To help you to understand how SLO reporting can be integrated with Tivoli Business Service Manager (TBSM), you can use this example configuration.

Before you begin

For a complete set of prerequisites, see “SLO reporting prerequisites” on page 279.

This sample configuration assumes the following:

- You have installed Tivoli Business Service Manager 6.1.1 Fix Pack 4.
- You have configured the Historic Reporting database on DB2 10.1 or higher in TBSM.
- You have installed Jazz 1.1.2 with Tivoli Common Reporting 3.1 or higher on DB2 10.1 or higher in TBSM.

About this task

After you complete this procedure, you can understand how to implement SLO reporting for historic data that is stored in TBSM

Procedure

1. Create the SLORPRT reporting database. For example:

```
db2 create database SLORPRT using CODESET UTF-8 territory en-US
```

If the database is on the same server as Tivoli Common Reporting, then you do not need to install a DB2 client. If the database is remote, you need to install a DB2 client and catalog the node and database on the local machine.
2. Copy the `slo_db2schema.sql` file from `<IMPACT_HOME>s/add-ons/slo/db` to the server where you want to create the SLORPRT database. Login with the DB2 user and run the following commands:

```
db2 connect to SLORPRT
db2 -tvf slo_db2schema.sql
db2 connect reset
```

3. Import the **SLA** and **SLA_Utility** projects.
Navigate to `<TBSM_HOME>/bin` and run the following command:

```
nci_import.[bat/sh] <servername> IMPACT_HOME/add-ons/slo/importData
```

For example:

```
./nci_import TBSM /opt/IBM/tivoli/impact/add-ons/slo/importData
```

Then run the following command:

```
nci_import.[bat/sh] <servername> IMPACT_HOME/add-ons/slo/sloutility
```

For example:

```
./nci_import TBSM /opt/IBM/tivoli/impact/add-ons/slo/sloutility
```

4. Import the report package to the Tivoli Common Reporting server.
 - a. Copy the `ImpactSLOReportPackage.zip` file from the `<IMPACT_HOME>/add-ons/slo/Reporting` directory in Netcool/Impact to the server where Tivoli Common Reporting is installed.
 - b. Log in to the Tivoli Common Reporting server with the user who installed Tivoli Common Reporting.

- c. Navigate to the /bin directory. For example /opt/IBM/JazzSM/reporting/bin.
- d. To create a data source for the SLORPRT database, enter the trcmd command:

```
trcmd.sh -user <TCR user> -password <TCR password>
-datasource -add SLORPRT -connectionString <db2 connection string>
-dbType DB2 -dbName <database name>
-dbLogin <database username> -dbPassword <database password>
```

For example:

```
./trcmd.sh -user smadmin -password password -datasource -add SLORPRT
-connectionString jdbc:db2://TCRserver.example.com:50000/SLORPRT
-dbType DB2 -dbName SLORPRT -dbLogin db2inst1 -dbPassword password
```

- e. To import the SLO reporting package, enter the trcmd command:

```
./trcmd.sh -import -bulk <file> -user <TCR User>
-password <TCR password>
```

For example:

```
./trcmd.sh -import -bulk /tmp/ImpactSLOReportPackage.zip -user smadmin
-password password
```

5. Create the SLO Reporting data source.
 - a. To log in to the Tivoli Integrated Portal GUI, use the appropriate url.
 - b. To open the **Data Model** page, click **System Configuration > Event Automation > Data Model**
 - c. From the drop-down list, click **Project SLA**.
 - d. Edit the **SLOReportDatasource**. Change the user name, password, host name, port, and database values to match the SLORPRT database.
 - e. To confirm that the values are correct, click **Test Connection**.
 - f. Save your changes.
 - g. From the drop-down list, click **Project SLA_UTILITY**.
 - h. Edit the **SLOUtilityDatasource**. Change the user name, password, host name, port, and database values to match the SLORPRT database.
 - i. To confirm that the values are correct, click **Test Connection**.
 - j. Save your changes.

6. Create the service definitions.

Services are specified by users in a properties file on the Data Server in TBSM. You use the createServiceDefinition policy in Netcool/Impact to import these into Netcool/Impact.

- a. Log in to the TBSM server with the user who installed TBSM.
- b. Select the service that you want to create a definition for.
- c. Create a copy of the service definition properties file in a temporary folder, calling the file ServiceDefinition.props.
- d. Add the properties. For more information, see “Service definition properties file” on page 282.
- e. Save your changes.
- f. Log in to the Tivoli Integrated Portal GUI.
- g. To open the **Policies** page, click **System Configuration > Event Automation > Policies**
- h. Click the createServiceDefinition policy and click the **Run with parameters** button.

- i. Enter the path to the directory where ServiceDefinition.props is stored.
 - j. To run the policy, click **Execute**.
 - k. To verify that the policy has run correctly, click **View Policy Log**.
7. Retrieve outage data from TBSM to store in the SLORPRT database
 - a. Log in to the Tivoli Integrated Portal GUI.
 - b. To open the **Data Model** page, click **System Configuration > Event Automation > Data Model**.
 - c. Create a DB2 data source called TBSM_HISTORY. Specify the values for the TBSM Metric History Database.
 - d. Test the connection and save the data source.
 - e. Create a data type called HISTORY_VIEW_METRIC_VALUE for the TBSM metric history data source.
 - f. Select TBSMHISTORY and HISTORY_VIEW_RESOURCE_METRIC_VALUE.
 - g. Click **Refresh**.
 - h. Select HISTORYRESOURCEID as the key field.
 - i. Save the data type.
 - j. To open the **Policies** page, click **System Configuration > Event Automation > Policies**.
 - k. Select the **getDataFromTBSMAvailability** policy and click **Run**. To update the data at regular intervals, you can use the GetData service. Edit the service so that the **getDataFromTBSMAvailability** policy runs at regular intervals, which you can specify in seconds. By default, the policy will record outages only when the TBSM status is "Bad".
8. Running the Application Availability report in Tivoli Common Reporting
 - a. Log in to the IBM Dashboard Application Services Hub GUI.
 - b. To open the **Common Reporting** page, click **Reporting > Common Reporting**.
 - c. Click **Impact SLO Report > SLO application availability report**.
 - d. Select the appropriate resource in the **Parameter Selection** field.
 - e. Select the metric name.
 - f. Specify an end date that occurs after the latest import of historic data for the service and click **Finish**.

Properties files examples

Use these examples to help you to define the service definition properties file and business calendar properties files.

Operational hours service level example

Use this example to help you to understand how to specify the same operational hours for all the SLAs in a service definition.

This example defines the operational hours at the service level. These hours apply to all the SLAs in the example because there are no operational hours defined for the SLAs.

```
# This sample service definition file defines the operational
# hours as properties of the service. Multiple slas are defined
# without operational hours. Each sla in this file will use the
# operational hours defined for the service.
# These hours apply ONLY to slas defined in this same file.
serviceName=SLOService
```



```

description=Service for testing SLO
identity=SLOService

# The next set of properties define the operational hours for the
# service. These are applied to all the slas defined later in the
# file, assuming the slas do not include specific operational hour
# definitions.
operationalHour.num=2
operationalHourStartTime.1 = 8:00:00
operationalHourEndTime.1 = 12:00:00
operationalHourStartTime.2 = 13:00:00
operationalHourEndTime.2 = 17:00:00

# Now define the slas, but do not define any operational hour periods,
sla.num=2
sla.1.name=serviceDowntimeEST
sla.1.displayName=Service Down Time
sla.1.description=Service Down Time
sla.1.timezone=EST
sla.1.operationalHourThreshold = 99.5
sla.1.nonOperationalHourThreshold = 98.5
sla.2.name=serviceDowntimePST
sla.2.displayName=Service Down Time
sla.2.description=Service Down Time
sla.2.timezone=PST
sla.2.operationalHourThreshold = 99.0
sla.2.nonOperationalHourThreshold = 98.0

```

Single SLA example

Use this example service definition to help you to understand how to define operational hours for a single SLA in a service definition properties file.

This example service definition specifies a single set of operational hours for the SLA:

```

# This sample service definition file contains a single sla
# definition with a single operational hour period. A business
# calendar is also defined.
serviceName=SLOService
description=Service for testing SLO
label=SLO Test Service
identity=SLOService
businessCalendar=US

sla.name=serviceDowntime
sla.displayName=Service Down Time
sla.description=Service Down Time

# Define a single operational hour period, 8AM to 5PM.
# Time zone defaults to GMT.
sla.operationalHourStartTime = 8:00:00
sla.operationalHourEndTime = 17:00:00

sla.operationalHourThreshold = 99.5
sla.nonOperationalHourThreshold = 98.5

```

Time zone example

Use the following example to help you to understand how to define operational hours in different time zones.

This example defines two sets of operational hours and time zones, one is for each coast of the United States:

```

# This sample service definition file contains multiple sla definitions
# to support operational hours across multiple time zones.
# Multiple operational hour periods are defined. A business calendar is
# also defined that will apply to both slas.
serviceName=SLOService
description=Service for testing SLO
label=SLO Test Service
identity=SLOService

# The following business calendar specification will apply to all
# slas defined in this properties file. If there is another sla or slas
# for the same service that require a different calendar, then you can
# copy this file, change the businessCalendar value, and replace the sla
# definitions below. Note that the description, label, and identity
# will be replaced for the service if any of those property values are
# changed in the copied file.
businessCalendar=US

# Creating two slas to measure availability across multiple time zones.
sla.num=2

# The first sla reflects the operational hours on the East Coast of the US.
# Normal hours are 8AM to 5PM, with an hour down for lunch at noon.
sla.1.name=serviceDowntimeEST
sla.1.displayName=Service Down Time
sla.1.description=Service Down Time
sla.1.timezone=EST
sla.1.operationalHour.num=2
sla.1.operationalHourStartTime.1 = 8:00:00
sla.1.operationalHourEndTime.1 = 12:00:00
sla.1.operationalHourStartTime.2 = 13:00:00
sla.1.operationalHourEndTime.2 = 17:00:00
sla.1.operationalHourThreshold = 99.5
sla.1.nonOperationalHourThreshold = 98.5

# The second sla reflects the operational hours on the West Coast of the US.
# Starting time is an hour later, but no down time for lunch.
# The thresholds displayed on the availability report are slightly lower.
sla.2.name=serviceDowntimePST
sla.2.displayName=Service Down Time
sla.2.description=Service Down Time
sla.2.timezone=PST
sla.2.operationalHourStartTime = 9:00:00
sla.2.operationalHourEndTime = 17:00:00
sla.2.operationalHourThreshold = 99.0
sla.2.nonOperationalHourThreshold = 98.0

```

Simple service definition example

Use the following example to help you to understand how you can configure a simple service definition properties file.

This example consists of the minimal required parameters:

```

# This sample service definition file consists of only the required properties
serviceName=SLOService

# label defaults to the empty string and description defaults to the
# serviceName value businessCalendar is optional, with no default value

# identity is a required property, but can be the same as the serviceName
# value when using a TBSM service
identity=SLOService

# Though technically not required, a service definition with no
# SLA metric name defined will not result in any outage data being
# created in SLORPRT database

```

```
sla.name=serviceDowntime

# sla.displayName and sla.description both default to the empty string
# sla.timezone defaults to GMT
# sla.operationalHourThreshold and sla.nonOperationalHourThreshold
# values default to 0. Since no operational hours are defined for
# the SLA or the service, the single default operational hour
# period "00:00:00" to "23:59:59" is used.
```

Multiple identities in a service definition example

Use this example to help you to understand how to define multiple identities in a single service definition file.

This example shows how you can use multiple identities in a single service definition properties file:

```
# This sample service definition file illustrates how to specify
# multiple identities for a service. This can be used when there are
# multiple sources of outage data for a service, but these sources use a
# different identity for the service.
serviceName=SLOService
description=Service for testing SLO

# There will be outage data gathered from 3 different sources,
# including the TBSM Metric History database. An identity is of the form
# "identityType::identityString". The default identity is
# "tbsmIdentity::serviceName".
# The SLO metric functions can be passed any of these identities as the
# "resource name" and the outage data will be calculated for service "SLOService".
identity.num=3
identity.1=SLOService # This is the same as tbsmIdentity::SLOService
identity.2=MSN::ManagedSystemName/ABCCompany/SLOService
identity.3=GUID::SLOService:guid:1ab2cd3ef4gh

# The rest of this file contains the single sla being defined by this
# file for SLOService.
sla.name=serviceDowntime
sla.displayName=Service Down Time
sla.description=Service Down Time
sla.operationalHourStartTime = 8:00:00
sla.operationalHourEndTime = 17:00:00
sla.operationalHourThreshold = 99.5
sla.nonOperationalHourThreshold = 98.5
```

Common US calendar properties

Use this example to help you to understand how to create a common business calendar properties file.

This business calendar specifies the common holidays and weekend days for the United States:

```
# This sample calendar definition file defines the "common"
# holidays and weekend days. Other calendars should specify this
# calendar for the property calendar.duplicateCalendarName to
# share these definitions.

# The name must start with the prefix "COMMON.". The SLO function
# uses the calendar definitions in calendars "US" and "COMMON.US"
# if the service definition specifies businessCalendar=US as a property.
calendar.name=COMMON.US

# The date format must meet the requirements defined by the
# Java SimpleDateFormat class.
calendar.holidays.dateformat= MMM dd,yyyy
```

```
# Defining 6 holidays. Any numbered property that is missing is skipped.
# Any numbered property above 6 (in this example) will be ignored.
calendar.holidays.num = 6
calendar.holidays.1 = Jan 1,2015
calendar.holidays.2 = Feb 14,2015
calendar.holidays.3 = Dec 25,2015
calendar.holidays.4 = Jan 1,2016
calendar.holidays.5 = Feb 14,2016
calendar.holidays.6 = Dec 25,2016

# Defining 2 weekend days. Any numbered property that is missing will be skipped.
# Any numbered property above 2 (in this example) will be ignored.
calendar.weekends.num = 2

# The "day number" is defined by the Java Calendar class. In the calendar for the
# US locale the constant SUNDAY is defined as 1 and the constant 7 is SATURDAY.
# MONDAY is 2, TUESDAY is 3, etc.
calendar.weekends.1 = 1
calendar.weekends.2 = 7
```

US Calendar example

Use this example to help you to understand how to create a business calendar properties file that defines unique holidays and weekends.

This example defines holidays and weekend days that are unique to the calendar. However, the holidays and weekends are supplemented by the properties in the COMMON.US calendar.

```
# This sample calendar definition file defines the holidays and weekend days unique
# to this specific calendar. This calendar will be supplemented by entries in the
# "COMMON" calendar of the same name.
```

```
# The name of this calendar. The entries in this calendar and the entries in
# calendar COMMON.US will all be used when evaluating outage time.
calendar.name=US
```

```
# The date format, as defined by Java class SimpleDateFormat, is required
# in order to specify holiday dates.
calendar.holidays.dateformat= MMM dd,yyyy
```

```
# The US calendar will have 2 holidays that are unique to the calendar.
# All other holidays and the weekend days are defined in the
# calendar COMMON.US.
calendar.holidays.num = 2
calendar.holidays.1 = Jul 4,2015
calendar.holidays.2 = Jul 4,2016
```

Common calendar properties file example

Use the example properties file to help you to create your own business calendar properties files.

This example duplicates the common holidays and weekend days from the COMMON.US calendar:

```
# This sample calendar definition file duplicates another calendar which
# defines "common" holidays and weekend days.
```

```
# The name must start with the prefix "COMMON.". The SLO function will use
# the calendar definitions in calendars "Canada" and "COMMON.Canada" if the
# service definition specifies businessCalendar=Canada as a property.
calendar.name=COMMON.Canada
```

```
# The following property instructs the calendar definition policy to just
```

```
# copy all the entries in calendar COMMON.US to this calendar.
calendar.duplicateCalendarName=COMMON.US

# Any other properties are ignored when calendar.duplicateCalendarName is specified.
```

Canada calendar example

Use this example business calendar properties file to define the holidays that are unique to Canada.

This business calendar specifies the unique holidays in Canada. The holidays and weekend days that are specified in the COMMON.Canada business calendar are also used when calculating the outage time for services using the calendar called Canada.

```
# This sample calendar definition file defines the holidays and weekend days unique
# to this specific calendar. This calendar is supplemented by entries in the
# "COMMON" calendar of the same name.

# The name of this calendar. The entries in this calendar and the entries in
# calendar COMMON.Canada are used when evaluating outage time.
calendar.name=Canada

# The date format, as defined by Java class SimpleDateFormat, is
# required to specify holiday dates.
calendar.holidays.dateformat= MMM dd,yyyy

# The Canada calendar will have 2 holidays that are unique to the calendar.
# All other holidays and the weekend days are defined in the
# calendar COMMON.Canada.
calendar.holidays.num = 2
calendar.holidays.1 = Jul 1,2015
calendar.holidays.2 = Jul 1,2016
```

SLA Utility properties

Use this example to help you to understand how to create a SLA Utility properties file.

```
serviceName=Service1
description=Service to test SLO
identity=tbsmIdentity:::Service1
sla.1.name=example_1
sla.1.displayName=ServiceExample1
sla.1.description=Service Example 1
sla.1.timezone=GMT
sla.1.operationalHour.num=2
sla.1.operationalHourStartTime.1=06:00:00
sla.1.operationalHourEndTime.1=10:00:00
sla.1.operationalHourStartTime.2=14:00:00
sla.1.operationalHourEndTime.2=18:00:00
sla.2.name=example_2
sla.2.displayName=ServiceExample2
sla.2.description=Service Example 2
sla.2.timezone=GMT
sla.2.operationalHour.num=2
sla.2.operationalHourStartTime.1=07:00:00
sla.2.operationalHourEndTime.1=11:00:00
sla.2.operationalHourStartTime.2=15:00:00
sla.2.operationalHourEndTime.2=19:00:00
```

SLO Utility Functions

There are utility policies available in project **SLA_Utility** that provide additional functionality for configuring the SLO feature, as well as functions for maintaining the outage data that is accumulated for reporting purposes.

These utilities complement the initial implementation found in the **SLA** project. If you deployed the SLO feature prior to Fixpack 4, you were instructed to import the **SLA_Utility** project when installing Fixpack 4.

If you have not imported the **SLA_Utility** project, you must complete the import in order to use these utility functions. You must update the **SLOUtilityDatasource** datasource after importing to connect to the SLORPRT database being used by existing datasource **SLOReportDatasource**.

The utility functions will generally throw exceptions for errors like missing parameters or names that do not exist. You should check the Impact `policylogger.log` file to make sure you got the expected results from the utility.

Note: When using utility functions that alter the SLO configuration or the outage data collected by the SLO feature, consider making a backup of the database before proceeding. As with any database, the SLORPRT database should be backed up on a regular interval as well as having regular DB2 maintenance applied for optimal performance.

Maintaining the reporting data in the SLORPRT database

Project **SLA_Utility** includes the policy **sloManageOutageTables** that can be activated to prune outage records from the SLO_METRIC_VALUE table in the SLORPRT database. This can help control the amount of data retained for reporting purposes by removing the older records. Using the default configuration, outage records more than 365 days old will be removed and archived.

In addition to the policy, an Impact activator service called **ManageSLOTables** is included that will activate the policy. It is set to activate once a day by default. This service is not started by default.

When the outage records are pruned, they are written to an archive table. By default the records will remain in the archive for 30 days, allowing for the possibility to restore outages if too much data has been removed. The policy **sloManageOutageTables** will also handle pruning records from the archive each time it runs.

See “Setting SLO configuration values” on page 308 for information on changing the default retention periods for both the outage table and the archived outage table.

See “Restoring outage data” on page 307 for information on restoring outage data from the archived outage table.

Removing service, SLA, and calendar definitions

Policies are included in the **SLA_Utility** project that will allow you to remove service, SLA, and calendar definitions that are no longer needed. The policies that remove the service and SLA definitions will archive any outage data that has already been recorded for the service and/or SLA.

Use the following policies to remove service definitions, SLA definitions, or the mapping of a service to an SLA definition:

sloDeleteServiceDefinition

This policy will delete a service definition and archive all outage records associated with the service. Removing the service also removes all mappings to any SLA that was used with the service. The SLA definitions may still be used by other services and are not affected.

You can add the service back later and restore the outages if they have not been pruned from the archive.

The “Service Name” parameter is required.

sloDeleteSLADefinition

This policy will delete an SLA definition and archive all outage records associated with the SLA. Removing the SLA also removes all mappings to any service that was using the SLA. The service definitions may still be used by other SLAs and are not affected.

You can add the SLA back later and restore the outages if they have not been pruned from the archive.

The “SLA Metric Name” parameter is required.

sloDeleteSLAForService

This policy will delete the mapping between an SLA and a service. All outage records for this SLA and service combination are archived. Only the mapping is removed – the service and/or SLA may be used in other mappings, so the definitions are not affected.

You can add the SLA to resource mapping back later and restore the outages if they have not been pruned from the archive.

The “SLA Metric Name” and “Service Name” parameters are required.

Use the following policy to remove calendar definitions.

sloDeleteCalendarDefinition

This policy will delete a calendar definition from the SLO configuration. If you have created a “COMMON” calendar for this calendar, then it will not be deleted. You must explicitly delete the “COMMON” calendar if it is also no longer required.

For example, when you have a calendar called **US**, the SLO feature will use information from this calendar and from the calendar called **COMMON.US**, if that calendar also exists. You would need to run the delete policy once for each calendar if you no longer need either.

Exporting service and calendar definitions

Policies are included in the **SLA_Utility** project that will allow you to export service and calendar definitions. This may be useful if you no longer have the properties files you used to create the services and calendars, or if you need to set up the same SLO configuration on another system.

Before you can export any SLO elements, you must define the configuration property **SLO_EXPORT_DIRECTORY**, which defines the target directory to be used for export operations. See “Setting SLO configuration values” on page 308 for more information.

Use the following policies to export service definitions. When a service is exported, a properties file is created for each SLA Metric name that is mapped to the service. The exported files will be placed in the directory defined by SLO configuration value **SLO_EXPORT_DIRECTORY** and named **SLO_Export_<service name>_<SLA metric name>.props**.

sloExportServiceDefinition

This policy will export one or more properties files that represent the definition of the service and each SLA mapped to the service.

The “Service Name” parameter is required

The “SLA Metric Name” parameter is required and must be either:

- The name of a specific SLA metric mapped to the service. This name must exist and be mapped to the service specified by “Service Name”.
- Or the single character *, which indicates that all SLAs mapped to the service should be exported.

sloExportAllServiceDefinitions

This policy will look up all service definitions defined in the SLO configuration and export each one. This will produce a properties file for every combination of a service and SLA defined in the SLO configuration.

There are no parameters for this policy.

Use the following policies to export calendar definitions.

When a calendar is exported, a properties file is created in the directory defined by SLO configuration value **SLO_EXPORT_DIRECTORY**. The file is named **SLO_Export_<calendar name>.props**.

sloExportCalendarDefinition

This policy will export a calendar definition properties file. If you have created a “COMMON” calendar for this calendar, then that calendar will also be exported.

For example, when you have a calendar called US, the SLO feature will use information from this calendar and from the calendar called **COMMON.US**, if that calendar also exists. When you run the export, two properties files are created, one for **US** and one for **COMMON.US**.

The “Calendar Name” parameter is required.

sloExportAllCalendarDefinitions

This policy will look up all calendar definitions defined in the SLO configuration and export each one. This will produce a properties file for each unique calendar definition, including any “COMMON” calendars.

There are no parameters for this policy.

Removing specific outage data

The **SLA_Utility** project includes functions that can be used to remove specific outages from the table used for the SLO Reporting. These functions should be used only to make finer adjustments to the outage data that was generated by the SLO function.

Note: You should always consider backing up the SLORPRT database before using functions to remove outage data.

The functions included can remove outages for a service, for an SLA metric, or for a specific SLA metric and service mapping. In addition, you can remove outages based on time criteria, for example outages before a certain time, after a certain time, or in a specific interval defined by a beginning and ending timestamp. Some functions combine the capabilities to support removing outages for services or SLAs while also defining a time range. When a time range is defined, it is applied to the VALUETIME column of the SLO_METRIC_VALUE table, which is the timestamp that defines when the outage occurred.

For all functions that remove outages, the outages are archived into another table and stored with the name of the service and SLA metric. If you later attempt to restore the outages from the archive, the service name and SLA metric name must exist in the configuration or the records will not be restored. Archived records include an “archive time”, which is set to the current time. This timestamp is used by the policy **sloManageOutageTables** if you have activated this policy to automatically prune the outage and archive tables.

Policy **sloRemoveOutages**

Policy **sloRemoveOutages** is provided as a sample policy for calling the various remove functions. The policy has extensive comments that describe how to call each function. In its shipped state the policy does not perform any actions, as all sample function calls are commented out.

You should create your own policy and use the code in **sloRemoveOutages** as a model for building the function calls you need.

Restoring outage data

The **SLA_Utility** project includes functions that can be used to restore specific outages from the archive table.

These outages were put in the archive when one of the following actions occurred:

- Regular maintenance was performed on the outage table by activating policy **sloManageOutageTables**
- A service, SLA, or mapping between a service and SLA was deleted
- One or more of the remove functions described in the previous section was used to remove specific outages from the SLO Reporting data

It is important to remember that archived outages are only retained for 30 days by default. This assumes you are activating policy **sloManageOutageTables** to perform regular pruning of the outage and archive tables.

Note: You should always consider backing up the SLORPRT database before using functions to restore archived data.

The restore functions mirror the remove functions described in the previous section, with the same capabilities for defining the restore criteria that are available when defining the remove criteria. The restore functions can recover outages for a service, for an SLA metric, or for a specific SLA metric and service mapping. In addition, you can restore outages based on time criteria or a combination of the time and name parameters.

For all functions that restore outages, the outages are restored into the SLO_METRIC_VALUE table and removed from the archive table. It is important to note the following when restoring outages:

- Each archived record has the name of the service and the name of the SLA metric. The service and SLA must exist in the configuration, and must have a mapping defined, or the records cannot be restored. For example, if a service was deleted, causing outages to be archived, then that service and the appropriate SLA mappings for the service must be created in the SLO configuration before the archived records can be restored.
- When archive records are restored, be aware that the next run of `sloManageOutageTables`, if activated, may archive the outage again, depending on the `VALUETIME` of the outage and your configuration value for how long to retain outage records (365 days by default). However, the archive time will be set to the current time, giving you more time to update the configuration and retry the restore before the outages are permanently deleted from the archive.
- Some restore functions may perform more slowly if there are a lot of archived records to examine. As noted above, the service name, SLA metric name, and mapping must be validated before an archived outage can be restored. Thus if only the time criteria is specified, then the archived records may include a variety of service and SLA combinations, each of which must be validated as part of the restore operation.

Always check the Impact `policylogger.log` file to ensure the expected outages were restored. If service, SLA, or mapping configuration is missing, correct the configuration and re-run the restore operation.

Policy `sloRestoreOutages`

Policy `sloRestoreOutages` is provided as a sample policy for calling the various restore functions. The policy has extensive comments that describe how to call each function. In its shipped state the policy does not perform any actions, as all sample function calls are commented out.

You should create your own policy and use the code in `sloRestoreOutages` as a model for building the function calls you need.

Setting SLO configuration values

This section describes each of the SLO configuration values and shows how to query and set the values.

`OUTAGE_RETENTION_DAYS`

Set this configuration value to the number of days that outages should be kept in table `SLO_METRIC_VALUE` before being archived. The default value is 365 days. This configuration value is used by policy `sloManageOutageTables` to prune the outage table, if the policy has been activated.

A higher value for this setting means that more outage data will be retained and a longer historical pattern can be shown in the reports. Reduce this value to prune the data more quickly and retain less historical data.

`OUTAGE_ARCHIVE_RETENTION_DAYS`

Set this configuration value to the number of days that archived outages should be kept in table `SLO_METRIC_VALUE_ARCHIVE` before being permanently deleted. The default value is 30 days. This configuration value is used by policy `sloManageOutageTables` to prune the archive table, if the policy has been activated.

A higher value for this setting means that more archive data will be retained, providing a longer opportunity to restore the data if needed. Reduce this value to prune the data more quickly, reducing the window of time when the data can be restored.

SLO_EXPORT_DIRECTORY

Set this configuration value to the directory to be used for exporting SLO services and calendars. This value must be set before running any export function. There is no default value.

TBSM_OUTAGE_STATUS

Set this configuration value to control how the sample policy **getDataFromTBSMAvailability** determines an outage from data in the TBSM metric history database.

The default, recommended value is “BAD”, which indicates that outages are defined as ONLY when the TBSM status is “Bad”. This is the default value if you deploy SLO for the first time after installing Fixpack 4.

If you already have SLO deployed when you install Fixpack 4, then an outage is defined to end ONLY when the TBSM status becomes “Good”. This was the previous behavior and will not be changed by just installing Fixpack 4.

Use **sloSetConfigurationValue** to set this property to “BAD” to configure the recommended behavior. Set the value to anything else (for example, “”) to use the legacy behavior, which requires the TBSM status to be “Good” to end an outage.

Policy sloSetConfigurationValue

The **SLA_Utility** project includes policy **sloSetConfigurationValue** for querying and setting the configuration values described above.

The parameter **Configuration Property** is required and must take one of the following values:

- OUTAGE_RETENTION_DAYS
- OUTAGE_ARCHIVE_RETENTION_DAYS
- SLO_EXPORT_DIRECTORY
- TBSM_OUTAGE_STATUS

The parameter **Configuration Value** specifies the value to be set for the property. If you specify “?” for this parameter, then the policy will just log the current value to the Impact policylogger.log file.

Chapter 17. Configuring Maintenance Window Management

Maintenance Window Management (MWM) is an add-on for managing Netcool/OMNIBus maintenance windows.

MWM can be used with Netcool/OMNIBus versions 7.x and later. 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 **MWMAActivator** 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 313.

Activating MWM in a Netcool/Impact cluster

Maintenance Window Management (MWM) interacts with Netcool/OMNIBus using an Netcool/Impact policy activator service called **MWMAActivator**. 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 **MWMAActivator**, right click, and select **Edit** or click the **Edit services** icon to open the **MWMAActivator** service properties. Make any required changes. For information about these properties, see “Configuring MWMAActivator service properties” on page 312.
7. To start the service, in the service status pane, select **MWMAActivator** 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 the Tivoli 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 with Netcool/OMNIBus events.

Netcool/OMNIBus events are based on OMNIBus field values such as **Node** or **Location**. The Netcool/OMNIBus events 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, `SuppressEsc1` 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.

Tip: For a **Like** command, there is no requirement for regular expressions. You can specify a substring and select the **Like** operator from MWM.

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

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 18. Configuring 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 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 6.1.1.4 information center available from the following URL, [Documentation for all Tivoli products](#).

You can load customized name space or your own model into SCR in by using the SCR API in TBSM. For more information see *Tivoli Business Service Manager Customization Guide, Customizing the import process of the Service Component Repository, Service Component Repository API overview*.

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 6.1.1.4 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 policies 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.

- **bsmidentities**
- **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 22.
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 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/topic/com.ibm.netcool_OMNIBus.doc_7.4.0/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 322. 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

A

- accessibility viii, 325
- activating 119
- add-ons
 - Maintenance Window
 - Management 311, 312, 313, 314, 315
- array 81
- arrays 134
 - finding distinct values 135
 - finding the length 135

B

- books
 - see publications vii, viii
- Button widget 203

C

- Cache Settings tab
 - External Data Types editor 27
- caching
 - count caching 33
 - data caching 32
 - query caching 33
- CommandResponse 130
- Configuration 149
- configuring data sources 319
- configuring data types 320
- context 83
- conventions
 - typeface xii
- Creating a policy 150
- Creating a service 152
- Creating an event rule 321
- Creating editing and deleting an event rule 321
- Custom Fields tab
 - internal data types editor 19
- customer support x

D

- data
 - adding 110, 111
 - deleting 113, 114
 - retrieving by filter 101
 - retrieving by key 107
 - retrieving by link 109
 - updating 112
- data caching 27
- data items 6, 101
 - field variables 101
- data model 1
 - components 5
 - creating 3
- data models
 - architecture 7
 - examples 7
- data models (*continued*)
 - setting up 6
- data source
 - connection information 13
- data sources
 - architecture 12
 - categories 10
 - creating 13
 - JMS 12
 - LDAP 11
 - Mediator DSA 11
 - overview 5, 10
 - setting up 13
 - SQL database 11
- data type
 - auto-populating 28
 - data item ordering 29
 - getting name of structural element 18
 - LDAP 14, 29
 - SQL 14
- data type caching 27
- data type field
 - description 17
 - display name 17
 - field name 16
 - format 17
 - ID 16
- data types 5, 25
 - caching 32
 - categories 14
 - configuring internal 18
 - configuring LDAP 30
 - configuring SQL 21
 - configuring SQL data types
 - Table Description tab 22
 - data item filter 28
 - external 13
 - fields 16
 - internal 13, 15
 - internal data types editor 19
 - keys 18
 - mediator 14
 - Mediator DSA 32
 - overview 13
 - predefined 13
 - predefined internal 15
 - setting up 18
 - SQL 21
 - system 15
 - user-defined internal 16
- database event listener
 - creating call spec 54
 - creating triggers 55, 57, 58, 59
 - editing listener properties file 52
 - editing nameserver.props file 51
 - example triggers 60, 61, 62, 63, 64, 65, 66
 - granting database permissions 53
 - installing client files into Oracle 52
 - sending database events 54

- database event listener (*continued*)
 - setting up database server 50
 - writing policies 67, 68
- database functions
 - calling 114
- DataItem (built-in variable) 101
- DataItems (built-in variable) 101
- directory names
 - notation xii
- disability 325
- dynamic links 34
 - link by filter 35
 - link by key 35
 - link by policy 35
 - setting up 36

E

- education
 - See Tivoli technical training
- enterprise service model
 - elements 8
- environment variables
 - notation xii
- event container 95
- event container variables
 - user-defined 96
- event enrichment 2
- event fields
 - accessing 96
 - updating 96
 - variables 95
- event gateway 3
- Event Isolation and Correlation 317, 318, 319, 320, 321
- Event Isolation and Correlation operator
 - views 319
- Event Isolation and Correlation
 - polices 319
- event notification 2
- event querying
 - reading state file 44
- event reader
 - actions 49
 - event locking 49
 - event matching 49
 - event order 50
 - mapping 48
- event readers
 - architecture 43
 - configuration 44
 - process 43
- event sources 37
 - Architecture 38
 - non-ObjectServer 37
 - ObjectServer 37
- event state variables 96
- EventContainer (built-in variable) 95
- events 95
 - adding journal entries to 97
 - deleting 99

- events (*continued*)
 - sending new 98
- Exporting and Importing
 - ForImpactMigration 150
- external data type
 - editor 22
- external data types
 - configuring SQL 21
 - editor 25
 - LDAP 30
 - Mediator DSA 32

F

- filters 102
 - LDAP filters 103
 - Mediator filters 104
 - SQL filters 102
- fixes
 - obtaining ix
- functions
 - user-defined 87

G

- glossary 327

H

- hibernating policy activator
 - configuration 76
- hibernations 117
 - removing 120
 - retrieving 118
 - waking 119

I

- If statements 84
- Impact policy language 79
- Installing Discovery Library Toolkit 318
- Installing Self Service Dashboard
 - widgets 200
- Installing the DB2 data base 318
- instant messaging 123
- Integrating data from a policy with the
 - topology widget 177
- internal data repository 12
- internal data types
 - configuring 18
 - editor
 - Custom Fields tab 19
- IPL
 - See* Impact policy language

J

- Jabber 123
- JavaScript and the UI Data Provider 211
- JMS
 - data source 12
- JRExec server
 - configuring 128
 - logging 129
 - overview 127

- JRExec server (*continued*)
 - running commands 129
 - starting 127
 - stopping 128

K

- key expressions 107
- keys 107
 - multiple key expressions 107

L

- Large data model support 205
- large data models 206
- Launching the Event Isolation and
 - Correlation analysis page 323
- LDAP data sources
 - creating 11
- LDAP External Data Type editor
 - LDAP Info tab 30
- LDAP external data types 30
- LDAP filters 103
- links 6, 33
 - categories 34
 - dynamic 34
 - overview 33, 109
 - setting up 35
 - static 34

M

- manuals
 - see* publications vii, viii
- Mediator DSA
 - data sources 11
 - data types 32
- Mediator filters 104
- multiple key expressions 107
- MWM
 - See* Maintenance Window Management

N

- Netcool/Impact data types as OSLC
 - resources 216
- notation
 - environment variables xii
 - path names xii
 - typeface xii

O

- ObjectServer event source
 - setting up 38
- omnibus event listener
 - triggers 70
 - using ReturnEvent 71
- omnibus event reader
 - event querying 44
 - event queueing 44
- OMNIBus triggers 72
- OMNIBusEventListener 72

- online publications
 - accessing viii
- operator view EIC_Analyze 323
- ordering publications viii
- OSLC 213
- OSLC and data types 216
- OSLC introduction 213
- OSLC resource shapes for data
 - types 219
- OSLC resources and identifiers 215
- OSLC Security 255
- Overview 149, 317

P

- PassToTBSM 149, 150, 152
- path names
 - notation xii
- percentage 179
- policies
 - creating 3
 - hibernating 117
- policy 1
 - language 79
 - retrieving data by filter 105, 106
 - retrieving data by key 108
 - retrieving data by link 109
- policy activators
 - configuration 73
- policy context 79
- policy log 79
 - printing to 80
- policy logger
 - configuration 74
- policy scope 79
- policylogger 76
- problem determination and resolution xi
- projects 150
- publications vii
 - accessing online viii
 - ordering viii

Q

- query caching 27

R

- RDFRegister function 232, 267
- RDFUnRegister function 235, 270

S

- scheduling policies 89
 - policy activator 89
 - schedules 90, 91, 92, 93
- Self service dashboard widgets 200
- service
 - command execution manager 77
 - command line manager 77
 - database event listener 50, 54
 - hibernating policy activator 76
 - omnibus event listener 70
 - OMNIBus event listener 69
 - OMNIBus event reader 42, 45, 46

- service (*continued*)
 - policy activator 73
 - policy logger 74
- service model
 - enterprise 8
- services
 - overview 41
 - predefined 41
 - setting up 3
 - user-defined 42
 - working with 1, 41
- Software Support
 - contacting x
 - overview ix
 - receiving weekly updates ix
- solution
 - running 4
- solution components 1
- solutions
 - setting up 3
 - types 2
- SQL data types
 - adding a field to the table 25
 - configuring 21, 22
- SQL filters 102
- static links 34
- Status 179
- strings 131
 - changing the case 134
 - concatenating 131
 - encrypting and decrypting 134
 - extracting a substring 132
 - finding the length 131
 - replacing a substring 133
 - splitting into substrings 132
 - stripping a substring 133
 - trimming whitespace 133
- Sybase data types
 - Setting the Exclude this field option 25

T

- Table Description tab
 - SQL External Data Types editor 22
- Tivoli Information Center viii
- Tivoli technical training viii
- training
 - Tivoli technical viii
- typeface conventions xii

U

- Uninstalling the Self Service Dashboard
 - widgets 201
- updating data 111
- user-defined functions 87
- using Spid 72

V

- variables
 - event field 95
 - event state 96
 - notation for xii
 - user-defined 80

- Viewing Event Isolation and Correlation
 - results 322, 323

W

- Web hosting model 9
 - elements 9
- WebGUI 322
- While statements 85
- working with data models 5

X

- x events in y time 2



Printed in USA

SC14-7560-00

