Netcool/Impact
Version 6.1.1.5

DSA Reference Guide

SC27-4852-01

Netcool/Impact
Version 6.1.1.5

DSA Reference Guide

IBM

SC27-4852-01

Note

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

Edition notice

This edition applies to version 6.1.1.5 of IBM Tivoli Netcool/Impact and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 2006, 2014.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2006, 2014

DSA Reference Guide

The Netcool/Impact *DSA Reference Guide* contains information about Impact data source adaptors (DSAs).

Intended audience

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

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*, SC14755901

  Provides information about installing, running and monitoring the product.
- *User Interface Guide*, SC27485101

  Provides instructions for using the Graphical User Interface (GUI).
- *Policy Reference Guide*, SC14756101

  Contains complete description and reference information for the Impact Policy Language (IPL).
- *DSA Reference Guide*, SC27485201

  Provides information about data source adaptors (DSAs).
- *Operator View Guide*, SC27485301

  Provides information about creating operator views.
- *Solutions Guide*, SC14756001

  Provides end-to-end information about using features of Netcool/Impact.
- *Integrations Guide*, SC27485401

  Contains instructions for integrating Netcool/Impact with other IBM® software and other vendor software.
- *Troubleshooting Guide*, GC27485501

  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.ibmlink.ibm.com/publications/servlet/pbi.wss.

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

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

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

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

Accessibility

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

For additional information, see Appendix A, “Accessibility,” on page 167.

Tivoli technical training

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

Support for problem solving

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

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

Obtaining fixes

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

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

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

Receiving weekly support updates

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

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

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

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

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

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

Contacting IBM Software Support

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

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

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

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

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

To contact IBM Software support, follow these steps:

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

Determining the business impact

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

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

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

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

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

Describing problems and gathering information

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

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

Submitting problems

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

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

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

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

Conventions used in this publication

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

Typeface conventions

This publication uses the following typeface conventions:

Bold

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

Italic

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

`Monospace`

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

Operating system-dependent variables and paths

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

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

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

Chapter 1. DSAs overview

DSAs are software components that are used to communicate with external data sources. DSAs broker information to and from SQL databases, LDAP servers, JMS topics and queues, and software systems that allow communication through web services APIs. You also use DSAs also to parse XML strings and documents, communicate with web servers through HTTP, and communicate with custom applications through generic socket transactions.

Chapter 2. Data source adapters (DSA)

Data source adapters (DSA) are software components that are used to communicate with external data sources.

Categories of DSAs

There are the following categories of DSAs:

SQL database DSAs
SQL database DSAs are used to access information stored in SQL database data sources. For more information about SQL database DSAs, see "Working with SQL database DSAs" on page 5.

LDAP DSA
The LDAP DSA are used to access information stored in an LDAP server. For more information about LDAP DSA, see Chapter 4, "Working with the LDAP DSA," on page 29.

Mediator DSAs
Mediator DSAs are used to communicate with various third-party applications or generic data interfaces (such as a Web services API or custom socket interfaces). For more information about Mediator DSAs, see "Mediator DSAs."

Mediator DSAs

Mediator DSAs are used to communicate with various third-party applications or generic data interfaces (such as a Web services API or custom socket interfaces).

Some Mediator DSAs are built in DSAs and do not require any additional installation or configuration. Other Mediator DSAs require you to manually install and configure them.

Table 1 lists the provided built-in Mediator DSAs:

Table 1. Mediator DSAs

Mediator DSA	For more information, see
Web services DSA	Chapter 5, "Working with the web services DSA," on page 33
JMS DSA	Chapter 8, "Working with the JMS DSA," on page 69
XML DSA	Chapter 9, "Working with the XML DSA," on page 81
SNMP DSA	Chapter 11, "Working with the SNMP DSA," on page 105
ITNM DSA	Chapter 12, "Working with the ITNM DSA," on page 135
Socket DSA	Chapter 13, "Working with the socket DSA," on page 141

The following Mediator DSAs are provided but you must install and configure them independently of the application:

- Alcatel 5620 DSA
- GE Smallworld DSA
- Cramer DSA

© Copyright IBM Corp. 2006, 2014

For more information about these Mediator DSAs, see the *Data Source Adapter Reference Guide*.

Managing data models

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

DSA (Data Source Adapter) data models are sets of data sources, data types, and data items that represent information that is managed by the internal data repository or an external source of data. For each category of DSA, the data model represents different structures and units of data that are stored or managed by the underlying source. For example, for SQL database DSAs, data sources represent databases; data types represent database tables; and data items represent rows in a database table.

The following DSAs; Web Services, SNMP, ITNM (Precision), XML, Cramer, and Socket, store some of the configuration in the `$IMPACT_HOME/dsa` directory. In a clustered environment, the `$IMPACT_HOME/dsa` directory will be replicated in the secondary servers in a cluster from the primary server during startup.

If you are changing these directories and configurations, it is best to make these changes on the primary server while the servers are down. When the changes are complete, start primary server followed by the secondary servers in the cluster. Some of the changes replicate in real time, for example if you use the Web Services and XML wizards. There is also a directory, `$IMPACT_HOME/dsa/misc`, where you can store scripts and flat files for example, which will be replicated across the cluster during startup of secondary servers that are retrieving this data from the primary server.

Event readers

Event readers are services that query a data source at intervals for events and then run a policy based on the incoming event data.

Two types of event readers are provided: standard event readers and database event readers. Standard event readers query a Netcool/OMNIbus ObjectServer database using the ObjectServer DSA. Database event readers query other relational databases using other types of SQL database DSAs

Event listeners

Event listeners are services that listen for incoming communication from an external data source through a DSA.

Event listeners are implemented by certain DSAs that provide the means for asynchronous exchange of data with the underlying sources of data. These DSAs include the database listener service for some SQL database DSAs (such as the Oracle DSA), OMNIbusEventListener for OMNIbus version 7.2 and later. They also include other listeners for Web services, JMS, and ITNM.

Policies

DSA policies are policies that contain instructions for interacting with a data source using a DSA. These policies contain calls to data-handling functions (such as `GetByFilter`) or DSA-specific functions that are instructions to send or retrieve information to and from the external data sources.

Working with SQL database DSAs

SQL database DSAs (data source adapters) are used to retrieve information from relational databases.

SQL database DSAs are also used to retrieve information from other types of data sources (like Netcool/OMNIbus ObjectServers, character-delimited files), and data sources that provide a public interface through JDBC (Java™ Database Connectivity). They are also used to add, modify, and delete information stored in these data sources.

The SQL database DSAs are direct-mode DSAs that run in-process with the Impact Server. SQL database DSAs are built in DSAs and do not require installation or configuration, but they require a JDBC driver to access data in the database. Only these SQL database DSAs have JDBC drivers provided automatically with Netcool/Impact:

- DB2
- Derby
- Informix
- HSQLDB
- ObjectServer
- Oracle
- PostgreSQL

Before you can use any other SQL database DSA, you must add its JDBC drivers to the class path. For a detailed procedure, see "Adding JDBC drivers and third-party JAR files to the shared library" on page 9.

You use SQL database DSAs by creating a data model, and writing policies. For more information, see "SQL database data model" on page 10, and "SQL database policies" on page 11.

List of provided SQL database DSAs

This topic provides a list, and a brief overview of SQL database DSAs.

Table 2. SQL database data source adapters

Data source adapter	Description
DB2 DSA	You use the DB2 DSA to access information in an IBM DB2 database. For more information about DB2 DSA, see "DB2 DSA" on page 6.
Derby DSA	The Derby DSA is used to access information in an Apache Derby database. For more information about Derby DSA, see "Derby DSA" on page 6.
Flat File DSA	You use the Flat File DSA to read information in a character-delimited text file. For more information about Flat File DSA, see "Flat File DSA" on page 7.

Table 2. SQL database data source adapters (continued)

Data source adapter	Description
Generic SQL DSA	You use the Generic SQL DSA to access information in any database application through a JDBC driver. For more information about Generic SQL DSA, see "Generic SQL DSA" on page 7.
HSQLDB DSA	You use the HSQL DSA to access information in a HSQL database. For more information about HSQL DSA, see "HSQLDB DSA" on page 7.
Informix® DSA	You use the Informix DSA to access information in an IBM Informix database. For more information about Informix DSA, see "Informix DSA" on page 7.
MySQL DSA	You use the MySQL DSA to access information in a MySQL database. For more information about MySQL DSA, see "MySQL DSA" on page 7.
MS-SQL Server DSA	You use the MS-SQL Server DSA to access information in a Microsoft SQL Server database. For more information about MS-SQL Server DSA, see "MS-SQL Server DSA" on page 7.
ObjectServer DSA	You use the ObjectServer DSA to access information in the Netcool/OMNIbus ObjectServer. For more information about ObjectServer DSA, see "ObjectServer DSA" on page 8.
ODBC DSA	Use the ODBC DSA to access information in an ODBC data base. For more information about ODBC DSA, see "ODBC DSA" on page 8.
Oracle DSA	You use the Oracle DSA to access information in an Oracle database. For more information about Oracle DSA, see "Oracle DSA" on page 8.
PostgreSQL DSA	Use the PostgreSQL DSA to access information in a PostgreSQL database. For more information about PostgreSQL DSA, see "PostgreSQL DSA" on page 8.
Sybase DSA	You use the Sybase DSA to access information in a Sybase database. For more information about Sybase DSA, see "Sybase DSA" on page 9.

DB2 DSA

You use the DB2 DSA to access information in an IBM DB2 database.

This DSA is used to retrieve, add, modify and delete information stored in DB2.

Derby DSA

Use the Derby DSA to access information in an Apache Derby database. The Derby DSA is used to store the underlying data that is used by the GUI reporting tools and Netcool/Impact solutions such as Maintenance Window Management.

You can also use the Apache Derby database to store other types of information that is used by Netcool/Impact. Any policies that access or update the other types of information sent to and from the Apache Derby database must be called by the policy activator service. It is not recommend to use any of the multi-threaded EventReaders.

The Derby DSA uses Apache Derby JDBC driver version 10.8.2.3. For more information about Apache Derby, see this URL http://db.apache.org/derby/.

Flat File DSA

You use the Flat File DSA to read information in a character-delimited text file.

You cannot use the Flat File DSA to write information to a text file. The Flat File DSA supports only the "AND" operator in flat file data type queries. You cannot use the "OR" operator to work with flat file data types. The flat file data source can be accessed like an SQL data source that uses standard SQL commands in Netcool/Impact for example, DirectSQL. Use an SQL database to run more complex queries. If you have to use the Flat File DSA, run multiple queries that do not require the use of the "OR" operator.

Restriction: The Flat File DSA is intended for use in demonstrating and testing Netcool/Impact and for infrequently accessing small amounts of data that is stored in a text file. Use of text files and the Flat File DSA is not an effective substitute for the use of a conventional relational database and an SQL database DSA. The Flat File DSA offers slower performance when compared to other DSAs.

Generic SQL DSA

You use the Generic SQL DSA to access information in any database application through a JDBC driver.

This DSA is used to retrieve, add, modify and delete information stored in the database. To use the Generic SQL DSA, you must specify its JDBC driver in the Generic SQL data source configuration window.

HSQLDB DSA

You use the HSQL DSA to access information in a HSQL database.

This DSA is supported with version 2.0 of the HSQL database server.

Informix DSA

You use the Informix DSA to access information in an IBM Informix database.

This DSA is used to retrieve, add, modify and delete information stored in the database. The DSA is supported with version 9.x, 10.x, and 11.x of the Informix database.

MS-SQL Server DSA

You use the MS-SQL Server DSA to access information in a Microsoft SQL Server database.

This DSA is used to retrieve, add, modify, and delete information stored in the database. It is used to run MS-SQL Server stored procedures. This DSA is supported with MS-SQL Server 2005, and 2008.

To use this data source adapter obtain the Microsoft SQL Server 3.0 Type 4 JDBC Driver, from this URL:

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=a737000d-68d0-4531-b65d-da0f2a735707&displaylang=en

MySQL DSA

You use the MySQL DSA to access information in a MySQL database.

This DSA is used to retrieve, add, modify, and delete information stored in the database. This DSA is supported with version 5.x of MySQL. To use this data source adapter obtain the latest Connector/J JDBC driver, from this URL:

http://dev.mysql.com/doc/refman/5.5/en/connector-j-versions.html

ObjectServer DSA

You use the ObjectServer DSA to access information in the Netcool/OMNIbus ObjectServer.

The ObjectServer DSA is supported by different versions of Netcool®/OMNIbus. You can use the Software Product Compatibility Reports (SPCR) to see the specific versions of Netcool/OMNIbus that are compatible with Netcool/Impact. To view a predefined report that lists the prerequisites for Netcool/Impact, see Prerequisites of Tivoli Netcool/Impact 6.1.1.

For more information about using SPCR, see Netcool Impact 6.1.1 system requirements.

Note: Use of the ObjectServer DSA is not necessary for retrieving events from the ObjectServer using the event reader server or for adding, updating, or deleting events from within a policy using the ReturnEvent function.

ODBC DSA

Use the ODBC DSA to access information in an ODBC data base.

Oracle DSA

You use the Oracle DSA to access information in an Oracle database.

The Oracle DSA is used to retrieve, add, modify, and delete information that is stored in the database. It is also used to run Oracle database stored procedures. This DSA is supports versions 9i, 10g, and 11g of the Oracle database server.

The Oracle DSA uses JDBC driver version 11.2.0.1.0, which is provided automatically in Netcool/Impact 6.1.1.5. You are not obliged to use JDBC driver version 11.2.0.1.0. You can download appropriate drivers from the Oracle website.

Remember: If you have upgraded from Netcool/Impact 6.1 to Netcool/Impact 6.1.1.5 see the section *Upgrading from Netcool/Impact 6.1, Post upgrade considerations* in the *Netcool/Impact Administration Guide* to verify that the Oracle JDBC JAR files are in the correct shared library for your environment.

PostgreSQL DSA

Use the PostgreSQL DSA to access information in a PostgreSQL database.

Netcool/Impact uses this DSA to retrieve, add, modify, and delete information stored in the database. This DSA is supported with versions 8.x and 9.x of the PostgreSQL database.

The PostgreSQL DSA uses JDBC driver version 9.2-1002 JDBC4 which is provided automatically in Netcool/Impact 6.1.1.5.

Remember: If you have upgraded from Netcool/Impact 6.1 to Netcool/Impact 6.1.1.5 see the section *Upgrading from Netcool/Impact 6.1, Post upgrade considerations* to verify that the PostgreSQL JDBC JAR file is in the correct shared library for your environment.

Sybase DSA

You use the Sybase DSA to access information in a Sybase database.

This DSA is used to retrieve, add, modify, and delete information stored in the database. It is also used to run Sybase stored procedures. This DSA is supported with version 15.x of the Sybase database server.

To use this data source adapter obtain the JDBC driver jConnect for JDBC version 6.0.5, from this URL:

http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect

Adding JDBC drivers and third-party JAR files to the shared library

Use this procedure to add a JDBC driver or third-party Java archive (JAR) files to the Netcool/Impact shared library.

About this task

You must copy the required JDBC drivers to the `$IMPACT_HOME/dsalib` directory.

You can also copy any third-party JAR files that you require to the same directory. For example, if you have specific Java classes that you want use with Java policy functions in Netcool/Impact, you add the JAR files to this directory.

Procedure

1. Obtain the appropriate JDBC driver according to the DSA specification or the third-party JAR files.
2. Copy the JDBC driver or third-party JAR files to the `$IMPACT_HOME/dsalib` directory.

   This directory is created during the installation, and initially it is empty.
3. Restart the Impact Server.

   Note: This procedure does not apply to DB2 and HSQL.

   For DB2 and HSQL, use the following procedure:

   a. Replace the jar file(s) for the DB2 JDBC driver in the `$IMPACT_HOME/lib3p/` directory.

   b. To update the Impact Server's ear, run the following command from the `$IMPACT_HOME/install/` directory:

   ```
   "..\bin\nc_ant -f new-server.xml -Dimpact.ewas.pw=<tipadmin password>
   updateears"
   ```

   This should be run with the Impact Server running.

   Prior to `6.1.1-TIV-NCI-FP0003`, you may need to edit the `new-server.xml` and `update-ear.py` files to complete a successful deployment:

   a. Edit the `$IMPACT_HOME/install/new-server.xml` file to add the following line after "`<include name="*.ear"/>`" (line 866):

   ```
   <exclude name="NCI_update.ear"/>
   ```

   b. Edit the `$IMPACT_HOME/install/update-ear.py` file to remove line 74:

   ```
   "[documenter.war documenter.war,WEB-INF/web.xml default_host] " + \
   ```

What to do next

In a clustered configuration, you must repeat this procedure for each server in the cluster because files in the `$IMPACT_HOME/dsalib` directory are not replicated between cluster members. It is recommended that all the servers in the cluster should be stopped while you are performing this procedure.

Changing the character set encoding for the database connection

Use this procedure to change the default character set encoding (UTF-8) that is used in establishing a connection to the SQL database.

Procedure

1. In the `$IMPACT_HOME/etc` directory, create a properties file for the DSA for which you want to change the default character set encoding.

   The properties filename must have the following format:

   `servername_drivermainclass``.props`

   where *servername* is the name of your Impact Server, and *drivermainclass* is the class name of the JDBC driver to connect to the SQL database.

   For example, you will create the `NCI_org.gjt.mm.mysql.Driver.props` file, if the name of your Impact Server is NCI, and if it is connecting to the MySQL database.

   Remember: You can get the *drivermainclass* values for other SQL databases, from their JDBC documenation.
2. Add a CHARSET=*encoding* property to the properties file.

   For example, `CHARSET=EUC_JP`.
3. Restart the Impact Server.

SQL database data model

An SQL database data model is an abstract representation of data stored in an underlying relational database or other data source that can be accessed through JDBC.

SQL database data models consist of SQL database data sources, SQL database data types, and SQL database data items.

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.

SQL database data types

An SQL database data type represents a table in a relational database or a similar structure that contains sets of data (like an Oracle view or a list of rows in a comma-delimited text file).

The configuration properties for the data type specify the structure and contents of data stored in the table. Some examples of SQL database data types are:

- A DB2 database table
- A MySQL database table
- The contents of a character-delimited text file

Each SQL database data type contains a set of fields that correspond to columns in the database table (or structured categories of data in other types of data sources). The data type can contain fields that represent all of the columns or a subset of the columns in the table.

You create SQL database data types using the GUI. You must create one such data type for each database table that you want to access.

When you create an SQL database data type, you need to specify such properties as the table name and the names of the table columns that you want to include in the data type. For the flat file DSA, you must specify additional configuration properties.

SQL database data items

An SQL database data item represents a table row in a relational database or another set of data (like a row in a comma-delimited text file).

You use the GUI to view, add, modify, and delete SQL database data items. Typically, however, you use the tools that are provided by the relational database server (or other third-party tools) to manage the data in an underlying data source.

SQL database policies

SQL database DSA policies work with data stored in underlying relational databases or other data sources that can be accessed using an SQL database DSA.

You can perform the following tasks by using a SQL database policy:

- Retrieve data from an SQL database data source
- Add data to an SQL database data source
- Modify data stored in an SQL database data source
- Delete data stored in an SQL database data source
- Call database functions

- Call database stored procedures

Retrieving data from an SQL database data source

The Impact Policy Language (IPL) provides a set of functions that retrieve data from an SQL database data source based on different criteria.

These functions allow you to retrieve data by key, by filter, and by link, and by directly running SQL `SELECT` queries against the underlying database or other source of data. The following table shows the IPL functions that retrieve SQL database data.

Table 3. IPL Functions that Retrieve SQL Database Data

Function	Description
GetByKey	Retrieves data items (rows in a table or other data element) whose key fields match the specified key expression.
GetByFilter	Retrieves data items whose field values match the specified SQL filter string.
GetByLinks	Retrieves data items that are dynamically or statically linked to another data item using the GUI.
DirectSQL	Retrieves data items by directly running an SQL `SELECT` query against the underlying database or other source of data.

For detailed syntax descriptions of these functions, see the *Policy Reference Guide*.

The following example shows how to use `GetByKey` to retrieve data items (rows in a table or other data element) whose key field matches the specified key expression. In this example, the SQL database data type associated with the table is `Customer` and the key expression is `12345`.

```
DataType = "Customer";
Key = 12345;
MaxNum = 1;

MyCustomer = GetByKey(DataType, Key, MaxNum);
```

The following example shows how to use `GetByFilter` to retrieve data items whose field values match the specified SQL filter string. In this example, the SQL database data type is `Node` and the filter string is `Location = 'New York City' AND Facility = 'Manhattan'`.

```
DataType = "Node";
Filter = "Location = 'New York City' AND Facility = 'Manhattan'";
CountOnly = False;

MyNodes = GetByKey(DataType, Key, MaxNum);
```

The following example shows how to use `GetByLinks` to retrieve data items that have been statically or dynamically linked to another data item using the Netcool/Impact GUI. In this example, you use `GetByLinks` to retrieve data items that are linked to the items of type `Node` returned in the previous example.

```
DataType = {"Customer"};
Filter = "";
MaxNum = 1000;
DataItems = MyNodes;

MyCustomers = GetByLinks(DataType, Filter, MaxNum, MyNodes);
```

Adding data to an SQL database data source

You can use the AddDataItem function to add data to an SQL database data source.

The following example shows how to use `AddDataItem` to add a row to an SQL database table that is represented by the `User` data type. In this example, `Name`, `Location`, `Facility`, and`Email` are columns in the database table.

```
DataType = "User";

MyUser = NewObject();

MyUser.Name = "John Smith";
MyUser.Location = "New York City";
MyUser.Facility = "Manhattan";
MyUser.Email = "jsmith@example.com";

AddDataItem(DataType, MyUser);
```

For a detailed syntax description of this function, see the *Policy Reference Guide*.

Modifying data stored in an SQL database data source

You can use the BatchUpdate function to modify the data that is stored on the SQL database. You can also assign values to variables for data items that were previously retrieved by using the GetByKey, GetByFilter, or GetByLinks to modify data stored in an SQL database.

The following example shows how to modify a row in an SQL database table by assigning values to member variables of a data item that was previously retrieved by using the GetByFilter function. In this example, the `Customer` data type represents a table in the underlying database and the `Name`, `Location`, and `Facility` fields represent columns in the table.

```
DataType = "Customer";
Filter = "Name = 'John Smith'";
CountOnly = "False";

MyCustomer = GetByFilter(DataType, Filter, CountOnly);

MyCustomer[0].Location = "Raleigh";
MyCustomer[0].Facility = "FAC_01";
```

The following example shows how to modify multiple rows in an SQL database table by using the BatchUpdate function. In this example, you update the `Location` and `Facility` columns in the table for each row where the value of `Location` is New York City.

```
DataType = "Customer";
Filter = "Location = 'New York City'";
UpdateExpression = "Location = 'Raleigh' AND Facility = 'FAC_01'";

BatchUpdate(DataType, Filter, UpdateExpression);
```

For more information about using these methods to modify SQL database data, see the *Policy Reference Guide*.

Deleting data stored in an SQL database data source

You can use policies to delete data that is stored in an SQL database data source by using the DeleteDataItem, or BatchDelete functions.

With these functions, you can delete either a single row or data element, or multiple rows. The following table shows the IPL functions that delete SQL database data.

Table 4. IPL Functions that Delete SQL Database Data

Function	**Description**
DeleteDataItem	Deletes a single data item which is a row in a table or other data element.
BatchDelete	Deletes one or more data items whose field values match the specified SQL filter string.

The following example shows how to delete a row in a database table by using the `DeleteDataItem` function. In this example, you first retrieve the data item that represents the row by using the GetByKey function and then call `DeleteDataItem`.

```
DataType = "Node";
Key = "DB2_01";
MaxNum = 1;

MyNode = GetByKey(DataType, Key, MaxNum);

DeleteDataItem(MyNode[0]);
```

The following example shows how to delete multiple rows from a database table by using the BatchDelete function. In this example, you delete all rows from the table that is represented by the `User` data type, where the value of the `Location` column is `New York City`.

```
DataType = "User";
Filter = "Location = 'New York City'";

BatchDelete(DataType, Filter, NULL);
```

For more information about using these functions to delete SQL database data, see the *Policy Reference Guide*.

Calling database functions

You can use the CallDBFunction to call any SQL function that is defined by the database server.

SQL functions vary per database. For a list of functions that are supported by a specific database server, see the documentation provided by the software vendor.

The following example shows how to call a database function named `NOW()` and return the results of the function for use in a policy.

```
// Call CallDBFunction and pass the name of a data type, a filter
// string and the function expression

DataType = "Server";
Filter = "0 = 0";
Metric = "NOW()";

DBTime = CallDBFunction(DataType, Filter, Metric);
```

For a detailed syntax description of the CallDBFunction function, see the *Policy Reference Guide*.

Calling database stored procedures

You can use the CallStoredProcedure function to call Oracle, Sybase, DB2, and SQL Server database stored procedures.

The following example shows how to call a Sybase stored procedure named `GetCustomerByLocation`. In this example, the Sybase database is represented by the data source `SYB_03`.

```
Sp_Parameter = NewObject();
Sp_Parameter.CustType = "Platinum";
Sp_Parameter.Location = "Mumbai";

DataSource = "SYB_03";
ProcName = "GetCustomerByLocation";

MyResults = CallStoredProcedure(DataSource, ProcName, Sp_Parameter);
```

For a detailed syntax description of the CallStoredProcedure function, see the *Policy Reference Guide*.

SQL database DSA failover

Failover is the process by which an SQL database DSA automatically connects to a secondary database server (or other data source) when the primary server becomes unavailable.

This feature ensures that Netcool/Impact can continue operations despite problems accessing one or the other server instance. You can configure failover separately for each data source that connects to a database using an SQL Database DSA.

SQL database DSA failover modes

Standard failover, failback, and disabled failover are supported failover modes for SQL database DSAs.

Standard failover
: Standard failover is a configuration in which an SQL database DSA switches to a secondary database server when the primary server becomes unavailable and then continues using the secondary until Netcool/Impact is restarted.

Failback
: Failback is a configuration in which an SQL database DSA switches to a secondary database server when the primary server becomes unavailable and then tries to reconnect to the primary at intervals to determine whether it has returned to availability.

Disabled failover
: If failover is disabled for an SQL database DSA the DSA reports an error to Netcool/Impact when the database server is unavailable and does not attempt to connect to a secondary server.

Standard failover:

Standard failover is a configuration in which an SQL database DSA switches to a secondary database server when the primary server becomes unavailable and then continues using the secondary until Netcool/Impact is restarted.

If the secondary server becomes unavailable, the SQL database DSA will attempt to resume connections to the original primary server.

Failback:

Failback is a configuration in which an SQL database DSA switches to a secondary database server when the primary server becomes unavailable and then tries to reconnect to the primary at intervals to determine whether it has returned to availability.

If the primary server has become available, the DSA will resume connections using that server. If the primary has not become available, the DSA will continue to use the secondary server. In a failback configuration, the SQL database DSA will always attempt to reconnect to the primary server before making a connection to the secondary.

Setting up DSA failover

You set up failover when you create and configure an SQL database data source in the GUI.

Procedure

You use the data source editor to select a failover configuration for the data source and to specify connection information for the primary and secondary database servers. For more information about creating and configuring SQL database data sources, see the *User Interface Guide.*

DSA failover defaults

An SQL database DSA determines that a database server is unavailable when it cannot connect to the database server, or when the database server returns an error message that is not related to SQL or stored procedure syntax.

Netcool/Impact provides a built-in list of errors messages that indicate that a database server has received an incorrectly formed SQL or stored procedure query. SQL database DSAs exclude these errors when determining whether a database server is unavailable. This means that, by default, a DSA does not failover or fail back when a syntax error occurs at the database level.

The following shows the built-in list of errors that Netcool/Impact excludes.

Table 5. SQL Database Error Messages for Failover

Database	**Error Codes**
DB2	No default error codes
Derby	No default error codes
GenericSQL	No default error codes
HSQLDB	No default error codes
Informix	Error codes from `-899` to `-200` inclusive
MySQL	Error codes `1047`, `1048`, `1051`, `1052`, `1054` to `1064` inclusive, `1071`, `1106` to `1111` inclusive, `1122`, `1138`, `1146`, `1217`, `1222`
ObjectServer	Error codes `667`, `5555`, `20000`, `20001`, `20002`
ODBC	No default error codes
Oracle	Error codes `100`, `900` to `999` inclusive, `17006`
PostgreSQL	SQL states `03000`, `42000`, `42601`, `42602`, `42622`, `42701`, `42702`, `42703`, `42704`, `42803`, `42804`, `42809`, `42883`, `42939`, `42P01`, `42P02`, `42P10`, `42P18`

Table 5. SQL Database Error Messages for Failover (continued)

Database	Error Codes
SQL Server	Error codes `105`, `207`, `208`, `213`, `229`, `230`, `260`
Sybase	Error codes `100` to `300` inclusive, `403`, `404`, `407`, `413`

For instructions in providing an alternate customized list, see "Customizing DSA failover."

Customizing DSA failover

You can provide an alternate list of error codes that the SQL database DSAs exclude when determining whether a database server is unavailable.

You store this list in a file named `$IMPACT_HOME/etc/`*`NCI`*`_non_failover_errors.props`, where *NCI* is the name of the Impact Server instance. This file is not automatically created so you must manually create and edit this file using a text editor.

Properties in this file have the following format:

```
impact.database=error_codes
```

where *database* is the name of the database and *error_codes* is a comma-separated list of error identification numbers. To specify a range of codes, place a less-than character between the lower limit and upper limit numbers as follows: `200<300`. The error code range is inclusive of the numbers specified.

The following table shows the internal database names that you must use in the properties file.

Table 6. Database Internal Names

Database	Internal Name
DB2	`db2`
Derby	derby
GenericSQL	`genericsql`
HSQL	hsqldb
Informix	`informix`
MS SQL Server	`mssql`
MySQL	`mysql`
Netcool/OMNIbus ObjectServer	`objectserver`
ODBC	`odbc`
Oracle	`oracle`
PostgreSQL	`postgresql`
Sybase	`sybase`

Error codes are defined by at the database level. For a list of possible error codes, see the documentation provided with the database application.

The following example shows a properties file that lists the default built-in error codes excluded by Netcool/Impact when determining if a database server is unavailable.

```
impact.db2=
impact.informix=-899<-200
impact.mssql=105,207,208,213,229,230,260
impact.mysql=1047,1048,1051,1052,1054<1064,1071,1106<1111,1122,1138,1146,
1217,1222
impact.objectserver=667,5555,20000,20001,20002
impact.odbc=
impact.oracle=100,900<999,17006
impact.postgresql=03000,42000,42601,42602,42622,42701,42702,42703,42704,
42803,42804,
42809,42883,42939,42P01,42P02,42P10,42P18
impact.sybase=100<300,403,404,407,413
```

Chapter 3. Working with the UI data provider DSA

The UI data provider DSA is used to return results from any UI data provider

To set up a UI data provider DSA complete the following steps:
- Create a UI data provider data source
- Create a UI data provider data type
- Create a policy that uses the `GetByFilter` function
- Run the policy to return the results from the selected UI data provider

UI data provider data model

A UI data provider data model is an abstract representation of data stored in an underlying relational database or other data source that can be accessed through a UI data provider.

The UI data provider data model has the following elements:
- UI data provider data sources
- UI data provider data types

UI data provider data sources

A UI data provider data source represents a relational database or another source of data that can be accessed by using a UI data provider DSA.

You create UI data provider data sources in the GUI. You must create one such data source for each UI data provider that you want to access.

Creating a UI data provider data source

Use this information to create a UI data provider 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 **UI Data Provider**. The tab for the data source opens.
4. In the **Data Source Name** field:

   Enter a unique name to identify the data source. You can use only letters, numbers, and the underscore character in the data source name. If you use UTF-8 characters, make sure that the locale on the Impact Server where the data source is saved is set to the UTF-8 character encoding.
5. In the **Host Name** field, add the location where the UI data provider is deployed. The location can be the local host or a fully qualified domain name.
6. In the **Port** field, add the port number of the UI data provider.
7. **Use SSL**: Select this check box to use SSL to transfer data. For more information, see the *Netcool/Impact Administration Guide* under the section *Secure Communication*.

8. **Base Url**: Type the directory location of the Tivoli Integrated Portal rest application, for example, `/ibm/tivoli/rest`.
9. **User Name**: Type a user name with which you can access the UI data provider.
10. **Password**: Type a password with which you can access the UI data provider.
11. Click the **Test Connection** button to test the connection to the UI data provider to ensure that you entered the correct information. Success or failure is reported in a message box. If the UI data provider is not available when you create the data source, you can test it later.

    To test the connection to the UI data provider at any time, from the data source list, right click the data source and select **Test Connection** from the list of options.
12. Click the **Discover Providers** button to populate the **Select a Provider** list.
13. From the **Select a Provider** list, select the provider that you want to return the information from.
14. From the **Select a Source** list, select the data content set that you want to return information from. The **Select Source** list is populated with the available UI data provider data content sets on the specified machine.
15. Click **Save** to create the data source.

UI data provider data types

A UI data provider data type represents a structure similar to a table that contains sets of data in a relational database. Each UI data provider database data type contains a set of fields that correspond to data sources in the UI data provider. You create UI data provider data types in the GUI. You must create one such data type for each data set that you want to access.

The configuration properties for the data type specify which subset of data is retrieved from the UI data provider data source.

Creating a UI data provider data type

Use this information to create a UI data provider data type.

Procedure

1. Right click the UI data provider data source you created, and select **New Data Type**.
2. In the **Data Type Name** field, type the name of the data type.
3. The **Enabled** check box is selected to activate the data type so that it is available for use in policies.
4. The **Data Source Name** field is prepopulated with the data source.
5. From the **Select a Dataset** list, select the data set you want to return the information from. The data sets are based on the provider and the data sets that you selected when you created the data source. If this list is empty, then check the data source configuration.
6. Click Save. The data type shows in the list menu.

Viewing data items for a UI data provider data type

You can view and filter data items that are part of a UI provider data type.

Procedure

1. In the **Data Model** tab, right click the data type and select **View Data Items**. If items are available for the data type, they show on the right side in tabular format.
2. If the list of returned items is longer than the UI window, the list is split over several pages. To go from page to page, click the page number at the bottom.
3. To view the latest available items for the data type, click the **Refresh** icon on the data type.
4. You can limit the number of data items that display by entering a search string in the **Filter** field. For example, add the following syntax to the **Filter** field, `totalMemory=256`. Click **Refresh** on the data items menu to show the filtered results.

   Tip: If your UI Data Provider data type is based on a Netcool/Impact policy, you can add `&executePolicy=true` to the **Filter** field to run the policy and return the most up to date filtered results for the data set.

   For more information about using the **Filter** field and GetByFilter function runtime parameters to limit the number of data items that are returned, see "Using the GetByFilter function to handle large data sets."

Using the GetByFilter function to handle large data sets

You can extend the GetByFilter function to support large data sets. To fetch items from a UI data providerwith the GetByFilter, additional input parameters can be added to the filter value of the GetByFilter function. Additional filter parameters allow you to refine the result set returned to the policy.

The UI data provider REST API supports the following runtime parameters:

- **`count`**: limits the size of the returned data items.
- **`start`**: specifies the pointer to begin retrieving data items.
- **`param_*`**: sends custom parameters to data sets that the UI data provider uses during construction and data presentation. The UI Data Provider server recognizes any additional parameters and handles the request if the parameter has the prefix **`param_`**. These values are also used to uniquely identify a data set instance in the REST service cache.
- **`id`**: If used, it fetches a single item. The **`id`** parameter specifies the **`id`** of item you want to retrieve. For example, **`&id=1`**. If the **`id`** parameter is used, all other filtering parameters are ignored.

Tip: If your UI Data Provider data type is based on a policy, then you can add `executePolicy=true` to the `FILTER` parameter in `GetByFilter( Filter, DataType, CountOnly)` to run the policy and ensure the latest data set results are returned by the provider.

This policy example uses the `FILTER` runtime parameters in a `GetByFilter (Filter, DataType, CountOnly)` implementation in a UI data provider.

```
DataType="123UIdataprovider";
CountOnly = false;

Filter = "t_DisplayName ='Windows Services'";
Filter = "t_DisplayName starts 'Wind'";
Filter = "t_DisplayName ends 'ces'";
Filter = "t_DisplayName contains 'W'&count=6&param_One=paramOne";
Filter = "t_DisplayName contains 'W'&count=3&start=2";
Filter = "((t_DisplayName contains 'Wi')
or  (t_InstanceName !isnull))";
```

```
Filter = "((t_DisplayName contains 'Wi')
or  (t_InstanceName='NewService'))&count=3";
Filter = "((t_DisplayName contains 'Wi')
or  (t_InstanceName='NewService'))&count=5&start=1";

MyFilteredItems = GetByFilter( DataType, Filter, CountOnly );

Log( "RESULTS: GetByFilter(DataType="+DataType+", Filter="+Filter+",
CountOnly="+CountOnly+")" );

Log( "MATCHED item(s): " + Num );

index = 0;
if(Num > 0){
  while(index <Num){
    Log("Node["+index+"] id = " + MyFilteredItems[index].id +
        "---Node["+index+"]  DisplayName= " +
MyFilteredItems[index].t_DisplayName);
    index = index + 1;
  }
}
Log("========= END =========");
```

Here are some more syntax examples of the FILTER runtime parameters that you can use in a `GetByFilter (Filter, DataType, CountOnly)` implementation in a UI data provider.

Example 1:

```
Filter = "&count=6";
```

No condition is specified. All items are fetched by the server, but only the first 6 are returned.

Example 2:

```
Filter = "&count=3&start=2";
```

No condition specified. All items are fetched by the server, but only the first 3 are returned, starting at item #2

Example 3:

```
Filter = "t_DisplayName ends 'ces'
```

Only items that match the condition = `"t_DisplayName ends 'ces'` are fetched.

Example 4:

```
Filter = "t_DisplayName contains 'W'&count=6&param_One=paramOne";
```

Only items that match the condition `"t_DisplayName contains 'W'&count=6&param_One=paramOne";` are fetched. Only the first six items that contain 'W' and `paramOne` are returned and `paramOne` is available for use by the provider when it returns the data set.

Example 5:

```
Filter = "&param_One=paramOne";
```

All items are fetched by the server, and `paramOne` is available for use by the provider when it returns the data set.

Adding Delimiters

The default delimiter is the ampersand (&) character. You can configure a different delimiter by editing the property `impact.uidataprovider.query.delimiter` in the `NCI_server.props` file. Any time you add a delimiter you must restart the Impact Server to implement the changes.

The delimiter can be any suitable character or regular expression, that is not part of the data set name or any of the characters used in the filter value.

The following characters must use double escape characters \\ when used as a delimiter:

```
* ^ $ . |
```

Examples:

An example using an Asterisk (*) as a delimiter:
- Property Syntax: `impact.uidataprovider.query.delimiter=\\*`
- Filter query: `t_DisplayName contains 'Imp'*count=5`

An example with a combination of characters:
- Property Syntax:`impact.uidataprovider.query.delimiter=ABCD`
- Filter query: `t_DisplayName contains 'Imp'ABCDcount=5`

An example of a regular expression, subject to Java language reg expression rules:
- Property Syntax: `impact.uidataprovider.query.delimiter=Z|Y`
- Filter query`t_DisplayName contains 'S'Zcount=9Zstart=7YexecutePolicy=true`

An example of a combination of special characters: `* . $ ^ |`
- Property Syntax: `impact.uidataprovider.query.delimiter=\\*|\\.|\\$|\\^|\\|`
- Filter query `t_DisplayName contains 'S'.count=9|start=7$executePolicy=true`

Retrieving data from a UI provider data source

Create a policy that includes the `GetByFilter` function to retrieve data by filter from a UI data provider data source.

To retrieve data from a UI data provider data source, you must create a Netcool/Impact policy that uses the `GetByFilter` function to return theUI data provider data items. The `GetByFilter` function is modified for use with data sources. This function retrieves data items whose properties match the specified UI data provider filter string. The UI data provider filter string is made up of three parts property 'id', operator, and the value.

You can use the operator `AND` and the operator `OR` to repeat the conditions. If you use these operators together, then the full expression must be in parentheses. For example:

```
((NAME contains 'abcd') or (TYPE isnull) or (DESCRIPTION starts 'abcd'))
and (SIZE >= 100) and (LAST_UPDATE > 1)
```

UI data provider data items contain many properties. Each of these properties has two attributes that are relevant for filtering UI data provider data items, a display value attribute and the actual value attribute. Operators are evaluated against the display value by default. If you want to filter for the actual values instead, you must add an asterisk (*) before the property. For example:

```
(*TYPE = 'SERVER')'s
```

For a full list of the available operators, see "UI data provider operators" on page 27

You can use the `Keys` function to return an array of strings that contain the field names for a specific UI data provider data item. For more information about the `Keys` function, see the *Netcool/Impact Policy Reference Guide*.

After you create the policy, you must create a user output parameter and associated custom schema values for the `GetByFilter` function to ensure that Netcool/Impact can process the values that the function returns from the external UI data provider:

1. In the policy editor, click the **Configure User Parameters** icon.
2. Click the **New Policy Output Parameter: New** button
3. Select **DirectSQL / UI Provider Datatype** in the **Format** field.
4. Enter a name for the parameter in the **Name** field.
5. Enter the same name as defined in the policy in the **Policy Variable Name** field.
6. To create the custom schema values, click the **Open Schema Definition Editor** icon. You must create custom schema values for each schema that is defined in the database and included in the returned results. To view the schema values that are required for your policy, right click the associated data type and click **View Data Items**. You must create a custom schema value for each column that you want to view in the widget in the console.

If you enable eventing for widgets that retrieved data from a UI data provider data type that uses the `GetByFilter` policy function, see the topic about eventing between widgets that access data from a UI data provider that uses GetbyFilter in the Troubleshooting Guide.

For more information about how to create user output parameters and custom schema values, see "Creating custom schema values for output parameters" on page 26.

Example

In the following policy example, the UI data provider data type called `uidataprovider-ImpactROI` is sourcing the data from the `REPORT_ImpactROI` data type that uses the `GetByFilter` function and the IPL policy language. The `REPORT_ImpactROI` data type is a standard data type delivered with Netcool/Impact.

```
DataType="uidataprovider-ImpactROI";
Filter = "PROCESS_NAME='Escalate'";
CountOnly = false;
```

The `GetByFilter` function returns an `OrgNodes` object that represents an array of values:

```
OrgNodes = GetByFilter( DataType, Filter, CountOnly );
```

The filter matches only one item in the data, and the `GetByFilter` function returns one item as a result:

```
Log("Number of org nodes returned:" + Num); // will be = 1
Log("Key = " + OrgNodes[0].Key); // will be = Escalate
```

In the following policy example, the data type is `myuidataproviderDataType`

```
DataType="myuidataproviderDataType";
Filter = "SAVED_TIME > 1000";
CountOnly = false;
```

This example returns the following `OrgNodes` object:

```
OrgNodes = GetByFilter( DataType, Filter, CountOnly );
```

If the filter matches two items, the `GetByFilter` function returns these two items as follows:

```
Log("Number of org nodes returned:" + Num);
// will be = 2
Log("Key = " + OrgNodes[0].Key);
// will be = Escalate
Log("Key = " + OrgNodes[1].Key);
// will be = Resolve
```

The following example demonstrates how to create a user output parameter and custom values to represent the output of the `GetByFilter` function. The following policy uses the `GetByFilter` function to retrieve data from an external UI data provider. The values that are returned are contained in the `DemoUISchema` parameter.

```
Filter="&count=200";
DemoUISchema=GetByFilter('UITestCuriMySQL',Filter,CountOnly);
Log(DemoUISchema);
```

You create the following output parameter for to represent the `DemoUISchema` parameter. You do not have to enter a data source or data type name.

Table 7. Output parameter for the `DemoUISchema` parameter

Field	**User entry**
Name	`DemoUISchema`
Policy variable name	`DemoUISchema`
Format	**DirectSQL / UI Provider Datatype**

After you create the output parameter, you must create custom schema values for `id`, `firstName`, and `lastName`. To view the schema values that are required for your policy, right click the associated data type and click **View Data Items**.

Table 8. Custom schema value for id

Field	**Entry**
Name	`id`
Format	**Integer**

Table 9. Custom schema value for fname

Field	**Entry**
Name	`firstName`
Format	**String**

Table 10. Custom schema value for lastName

Field	**Entry**
Name	`lastname`
Format	**String**

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:

```
O1.city="NY"
O1.ZIP=07002
```

You define the following custom schemas values for this policy:

Table 11. Custom schema values for City

Field	**Entry**
Name	`City`
Format	**String**

Table 12. 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**.

UI data provider operators

You use these operators to create a filter string for UI data provider data sources.

Table 13. Operators for creating filter strings

String	Numeric and date	Boolean and enumerated
contains	=	=
!contains	!=	!=
starts	<	
!starts	<=	
ends	>	
!ends	>=	
isnull		
!isnull		
=		
!=		

Chapter 4. Working with the LDAP DSA

The LDAP DSA are used to access information stored in an LDAP server.

This type of DSA is read-only. You cannot use Netcool/Impact to insert new LDAP data into the server data store. The LDAP DSA is s built in DSA and does not require any additional installation or configuration.

LDAP DSA overview

Netcool/Impact uses the Lightweight Directory Access Protocol (LDAP) data source adaptor to retrieve data managed by an LDAP server.

The LDAP DSA is a direct-mode data source adaptor that runs in-process with Netcool/Impact. This DSA is automatically loaded during application run time. You do not have to start or stop this DSA independently of the application. Netcool/Impact is not able to use this DSA to add, modify, or delete information managed by the LDAP server

To use the LDAP DSA, complete the following tasks:

- Create an LDAP DSA data model that provides an abstract representation of the data managed by the LDAP server.
- Write one or more LDAP DSA policies that retrieve data from the underlying LDAP server.

For more information about LDAP data model, see."LDAP data model"

For more information about LDAP policies, see."LDAP policies" on page 31

Supported LDAP servers

Netcool/Impact supports directory servers that fully implement the LDAP v2 and v3 specifications, including Netscape, iPlanet, OpenLDAP, and Microsoft Active Directory servers.

LDAP data model

A Lightweight Directory Access Protocol (LDAP) data model is an abstract representation of data that is managed by an LDAP directory server.

LDAP data models have the following elements:

- LDAP data sources
- LDAP data types
- LDAP data items

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.

© Copyright IBM Corp. 2006, 2014

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.

LDAP data types

A Lightweight Directory Access Protocol (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 entities that match a specified LDAP filter within a certain scope. The DSA searches in relation to a location in the tree known as the base context.

Use the GUI to create LDAP data. You must create one LDAP data type for each set of entities that you want to access.

The following table shows the configuration properties for an LDAP data type.

Table 14. LDAP Data Type Configuration Properties

Configuration Property	**Description**
Data type name	Name of the new LDAP data type.
Search scope	Keyword that indicates the scope for the LDAP search. Possible values are: `OBJECT_SCOPE`, `ONELEVEL_SCOPE`, and `SUBTREE_SCOPE`. `OBJECT_SCOPE` causes the LDAP DSA to search only the specified base context for matches. `ONELEVEL_SCOPE` causes the DSA to search only the child entities of the base context for matches. `SUBTREE_SCOPE` causes the DSA to search all descendants of the base context.
Base context	Location in the LDAP tree with respect to which the LDAP DSA searches for matching entities. An example is `ou=people, o=IBM.com`.
Key search field	Attribute in the LDAP entity that uniquely identifies it as a key. Used when you retrieve data items from an LDAP data type with the `GetByKey` function in a policy.
Display name field	Attribute in the LDAP entity that is displayed when you use the GUI to browse LDAP data items.
Restriction filter	LDAP search filter as described in Internet RFC 2254: String Representation of LDAP Search Filters.

LDAP data items

A Lightweight Directory Access Protocol (LDAP) data item represents an entity in the LDAP directory tree.

Each field in an LDAP data item corresponds to an attribute in the LDAP entity.

You use the GUI to view LDAP data items. You cannot use the GUI to add, modify, or delete LDAP data items.

LDAP policies

Information from LDAP data sources is retrieved by the LDAP policies, which are Netcool/Impact policies. You cannot add, modify, or delete LDAP data from within a policy.

Retrieving data from an LDAP data source

You can retrieve data, by key, filter, or link, from an LDAP data source by using the GETbyKey, GetByFilter and GetByLinks functions when writing a policy.

The following table describes the functions that retrieve LDAP data.

Table 15. Functions that Retrieve LDAP Database Data

Function	Description
`GetByKey`	Retrieves data items, or entities in the LDAP directory tree, whose key fields match the specified key expression.
`GetByFilter`	Retrieves data items whose field values match the specified LDAP filter string.
`GetByLinks`	Retrieves data items that are dynamically or statically linked to another data item using the Netcool/Impact GUI.

Example

The following example shows how to use `GetByKey` to retrieve data items, or entities in the LDAP directory tree, whose key field matches the specified key expression. In this example, the LDAP data type associated with a search scope in the tree is `Customer` and the key expression is `12345`.

```
DataType = "Customer";
Key = 12345;
MaxNum = 1;

MyCustomer = GetByKey(DataType, Key, MaxNum);
```

The following example shows how to use `GetByFilter` to retrieve data items whose field values match the specified LDAP filter string. The LDAP filter is part of the specification described in Internet RFC 2254. In this example, the LDAP data type is `Facility` and the filter string is `(|(facility=Wall St.)(facility=Midtown)(facility=Jersey City))`.

```
DataType = "Facility";
Filter = "(|(facility=Wall St.)(facility=Midtown)(facility=Jersey City))";
CountOnly = False;

MyFacilities = GetByFilter(DataType, Filter, CountOnly);
```

The following example shows how to use `GetByLinks` to retrieve data items that are statically or dynamically linked to another data item using the Netcool/Impact GUI. In this example, you use `GetByLinks` to retrieve data items of type `Customer` that are linked to data items in the `MyFacilities` array returned in the previous example.

```
DataType = {"Customer"};
Filter = "";
MaxNum = 1000;
DataItems = MyFacilities;

MyCustomers = GetByLinks(DataType, Filter, MaxNum, DataItems);
```

For detailed syntax descriptions of these functions, see the *Policy Reference Guide*.

International character support

The Lightweight Directory Access Protocol (LDAP) Data Source Adaptor (DSA) follows the LDAP v3 standard for international character support.

This standard specifies that non-ASCII characters must be stored in UTF-8 format in the LDAP server to be handled correctly by client applications. If you use the LDAP DSA to access non-ASCII character data, make sure that the data is encoded using the UTF-8 standard.

Chapter 5. Working with the web services DSA

The web services data source adaptor (DSA) is a direct-mode adaptor that Netcool/Impact automatically loads during application run time.

You do not have to start or stop this DSA independently of the application. The web services DSA is installed with Netcool/Impact so you do not have to complete any additional installation or configuration steps.

Web services DSA is compatible with its older versions in Netcool/Impact 3.x and 4.x. Your old IPL policies, that were developed on Netcool/Impact 4.x and 3.x will run without modification in the current version.

The web services DSA provides support for WSDL version 1.1 and 2.0, and SOAP version 1.1.

Web services DSA overview

The web services data source adaptor (DSA) is used to exchange data with external systems, devices, and applications through web services interfaces.

The web services DSA uses blocking messages to communicate with web services. The use of blocking messages forces Netcool/Impact to wait for a reply from the web service before it can continue processing a policy. If Netcool/Impact does not receive a reply in the specified time frame, the DSA times out and returns an error message to Netcool/Impact.

During policy run time, simple object access protocol (SOAP) messages are sent through the DSA to the specified web service. The message structure is defined by a web services definition language (WSDL) file. The message content is defined in the policy.

After the DSA sends a message, it waits for a reply from the web service. When the DSA receives the reply, the returned data is converted into data items and returned to the Impact Server for further processing in the policy.

You do the following tasks when working with the web services DSA:

- Compile WSDL files that are associated with the interfaces provided by a web service.
- Create and configure a web services listener that listens on an HTTP port for SOAP/XML messages from external applications.
- Write policies that send messages to a web services interface and handle the message replies.
- Write policies that handle SOAP/XML messages received by the web services listener.

Migrating web services DSA

Use one of the following methods to run policies you created in earlier versions of Netcool/Impact on the current version. The options differ if you decide to recompile the WSDL file or not

© Copyright IBM Corp. 2006, 2014

About this task

Your policies that were developed in Tivoli Netcool/Impact 4.x and 3.x continue to run without modification on the current version of Impact Server. If you recompile your WSDL file with the current version of `nci_compile_wsdl` script and want to use the new JAR file, you must rewrite your policy with new web services DSA functions.

Note: The WSDL file that defines a rpc/encoded style web service cannot be compiled by `nci_compilewsdl` in Tivoli Netcool/Impact 5.1 or higher. The web services DSA in the current version of Netcool/Impact do not support rpc/encoded web services. Rpc/literal and Document/literal are supported web services styles. However, if you reuse the web service client JAR files generated by Netcool/Impact 3.x or 4.x from rpc-encoded style WSDL files, your old polices will run on the current version of Netcool/Impact without changes.

Procedure

1. Copy your old JAR files to the `$IMPACT_HOME/wslib` directory.
2. Create a policy and then copy and paste policy codes from your old policy.
3. Save the policy.
4. Run the policy.

Compiling WSDL files

Before you can use the web services DSA, you must compile a Web Services Description Language (WSDL) file.

When you compile a WSDL file, you create a set of Java class files that contain a programmatic representation of the WSDL data. This representation is then used by the web services DSA when it sends messages to the web service and handlesv message replies.

WSDL files are XML documents that describe the public interface that is provided by a web service.

To compile the WSDL, you complete the following tasks:

1. Obtain the WSDL file for the web service.
2. Run the WSDL compiler script.
3. The JAR files are created in the `$IMPACT_HOME/wslib` directory on the primary server. Copy the JAR files from the `$IMPACT_HOME/wslib` directory to all the secondary servers.

Note: If the WSDL file contains XSD imports, files are provided separately. The WSDL files and related XSD files must be placed in a directory with no spaces.

For more information about WSDL files, see the Web Services Description Working Group home page on the W3C website at http://www.w3c.org/2002/ws/desc.

Obtaining WSDL files

Every web service must provide one or more Web Services Description Languages (WSDL) files that define its public interfaces. WSDLs are available from known URLs.

Procedure

Use a version of the WSDL file that defines the simple object access protocol (SOAP) interface for the web service with the web services DSA. WSDL files are most often made available by a web service at a known URL. For example, the web service WSDL for a real-time stock quote service is available at `http://www.webservicex.net/stockquote.asmx?wsdl`. You can compile a WSDL by using its URL or by using a copy of the file that is stored locally in your file system.

Running the WSDL compiler script

The WSDL compiler script, `nci_compilewsdl`, creates a JAR file that contains a programmatic representation of the WSDL data.

Procedure

1. Navigate to the `$IMPACT_HOME/bin` directory.
2. At the command prompt run the compiler script with the following options:

   ```
   nci_compilewsdl package_name wsdl_file
   destination
   ```

   Table 16 contains descriptions of the command-line arguments for this script.

Table 16. WSDL compiler script command line arguments

Argument	Argument description
package_name	Name of the JAR file (without the `.jar` suffix) to be created by the script.
wsdl_file	The fully qualified URL of the WSDL file, or the location of the WSDL file on the local file system. The script looks for the file in the `$IMPACT_HOME` directory by default. You can also specify the absolute path to the file.
destination	The directory to copy the generated JAR to. Default is `$IMPACT_HOME/wslib`

   You must enter the entire command in one line, without any line breaks. For example on UNIX:

   ```
   ./nci_compilewsdl amazon US.wsdl $IMPACT_HOME/wslib
   ```

   The example command compiles a WSDL file, `US.wsdl` that is located in the current working directory, and creates the `amazon.jar` file, in the `$IMPACT_HOME/wslib` directory.

   Another example shows how to compile a WSDL file located at a URL:

   ```
   ./nci_compilewsdl weather
   http://www.webservicex.net/WeatherForecast.asmx?WSDL ../wslib
   ```

   The `weather.jar` file, is created under `$IMPACT_HOME/wslib` directory.
3. Optional: If the destination directory for the script was different than the default one, you must copy the generated Jar file into the `$IMPACT_HOME/wslib` directory.

Recompiling new and changed WSDL files

If you change an existing WSDL file or add a new file that uses classes from an existing WSDL file, you must clear the cache and compile the WSDL file again.

About this task

Netcool/Impact uses the Java archive files that the Java virtual machine stores in the `$IMPACT_HOME/wslib` directory. You must clear this cache so Netcool/Impact can process the changes included in the newly compiled WSDL file.

Procedure

1. Change an existing WSDL file or create a new file that references an existing file.
2. Move all the Java archive files from the `$IMPACT_HOME/wslib` directory to a temporary location.
3. Restart Netcool/Impact.
4. Compile the WSDL file.
5. If the compiled WSDL file is not saved to the `$IMPACT_HOME/wslib` directory, move the new JAR file to the `$IMPACT_HOME/wslib` directory.
6. Move all the Java archive files, except for the files that you either changed or referenced in your new WSDL file, from the temporary directory to the `$IMPACT_HOME/wslib` directory. If you do copy the files that you changed, your changes are overwritten with the original file.

Compiling WSDL files on Windows platforms

Before you can use the web services DSA on a Windows platform, you must compile a Web Services Description Language (WSDL) file for the web service.

When you compile a WSDL file, you create a set of Java class files that contain a programmatic representation of the WSDL data. This representation is then used by the web services DSA when it sends messages to the web service and handles message replies.

To compile the WSDL, complete the following tasks:

1. Stop Netcool/Impact.
2. Obtain the WSDL file for the web service.
3. Run the WSDL compiler script.
4. The JAR files are created in the `$IMPACT_HOME/wslib` directory on the primary server. Copy the JAR files from the `$IMPACT_HOME/wslib` directory to all the secondary servers.

Note: If the WSDL file contains XML schema definition (XSD) imports, these files are provided separately. The WSDL files and related XSD files must be placed in a directory with no spaces.

WSDL files are XML documents that describe the public interface that is provided by a web service. For more information about WSDL files, see the web Services Description Working Group home page on the W3C website at http://www.w3c.org/2002/ws/desc.

Web services DSA functions

The web services DSA provides a set of special functions that you use to send messages from Netcool/Impact to a web service.

The web services DSA functions are:

- `WSSetDefaultPKGName`
- `WSNewObject`
- `WSNewSubObject`
- `WSNewArray`
- `WSNewEnum`

- `WSInvokeDL`

WSSetDefaultPKGName

The WSSetDefaultPKGName function sets the default package that is used by WSNewObject and WSNewArray.

The package name is the name that you supplied to the `nci_compilewsdl` script when you compiled the WSDL file for the web service. It is also the name of the JAR file that is created by this script, without the `.jar` suffix.

Syntax

This function has the following syntax:

```
WSSetDefaultPKGName(PackageName)
```

Parameters

The `WSSetDefaultPKGName` function has the following parameter.

Table 17. WSSetDefaultPKGName function parameter

Parameter	Format	Description
PackageName	String	Name of the default WSDL package used by `WSNewObject` and `WSNewArray`.

Example

The following example sets the default package that is used by subsequent calls to `WSNewObject` and `WSNewArray` to `google`.

```
WSSetDefaultPKGName("google");
```

WSNewObject

The WSNewObject function creates an object of a complex data type as defined in the WSDL file for the web service.

You use this function when you are required to pass data of a complex type to a web service as a message parameter.

Syntax

This function has the following syntax:

```
Object = WSNewObject(ElementType)
```

Parameters

This `WSNewObject` function has the following parameter.

Table 18. WSNewObject function parameter

Parameter	Format	Description
ElementType	String	Name of the complex data type that is defined in the WSDL file. The name format is *[Package.]TypeName*, where *Package* is the name of the package you created when you compiled the WSDL file, without the `.jar` suffix.

Return Value

A new web services object.

Examples

The following example shows how to use `WSNewObject` to create a web services object, what you previously called `WSSetDefaultPKGName` in the policy. This example creates an object of the data type `ForwardeeInfo` as defined in the `mompkg.jar` file compiled from the corresponding WSDL.

```
// Call WSSetDefaultPKGName
WSSetDefaultPKGName("mompkg");

// Call WSNewObject

MyObject = WSNewObject("ForwardeeInfo");
```

The following example shows how to use `WSNewObject` to create a web services object, where you did not previously call `WSSetDefaultPKGName` in the policy.

```
// Call WSNewObject

MyObject = WSNewObject("mompkg.ForwardeeInfo");
```

WSNewSubObject

The WSNewSubObject function creates a child object that is part of its parent object and has a field or attribute name of ChildName.

Syntax

This function has the following syntax:

```
Object = WSNewSubObject(ParentObject, ChildName)
```

Parameters

This `WSNewSubObject` function has the following parameters.

Table 19. WSNewSubObject function parameters

Parameter	Format	Description
ParentObject	String	Name of the parent object
ChildName	String	Name of the new child object

Return Value

A new web services child object.

Examples

The following example shows how to use `WSNewSubObject` to create a web services child object:

```
// Call WSNewSubObject

ticketId=WSNewSubobject(incident, "TICKETID");
```

WSNewArray

The WSNewArray function creates an array of complex data type objects or primitive values, as defined in the WSDL file for the web service.

You use this function when you are required to pass an array of complex objects or primitives to a web service as message parameters.

Syntax

This function has the following syntax:

```
Array = WSNewArray(ElementType, ArrayLength)
```

Parameters

The `WSNewArray` function has the following parameters:

Table 20. WSNewArray function parameters

Parameter	Format	Description
ElementType	String	Name of the complex object or primitive data type that is defined in the WSDL file. The name format is [*Package*.]*TypeName*, where *Package* is the name of the package you created when you compiled the WSDL file, without the `.jar` suffix. The package name is required only if you did not previously call the `WSSetDefaultPKGName` function in the policy.
ArrayLength	Integer	Number of elements in the new array.

Return Value

The `WSNewArray`returns the new array that is created by the function.

Examples

The following example shows how to use `WSNewArray` to creates a web services array, where you previously called `WSSetDefaultPKGName` in the policy. This example creates an array of the data type `String` as defined in the `mompkg.jar` file that is compiled from a WSDL file.

```
// Call WSSetDefaultPKGName

WSSetDefaultPKGName("mompkg");

// Call WSNewArray

MyArray = WSNewArray("String", 4);
```

The following example shows how to use `WSNewArray` to create a web services array, where you did not previously call `WSSetDefaultPKGName` in the policy.

```
// Call WSNewArray

MyArray = WSNewArray("mompkg.String", 4);
```

WSInvokeDL

The `WSInvokeDL` function makes web services calls when a Web Services Description Language (WSDL) file is compiled with `nci_compilewsdl`, or when a web services DSA policy wizard is configured.

Syntax

This function has the following syntax:

```
[Return] = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams, [callProps])
```

This function returns the value of your target web services call.

Parameters

The `WSInvokeDL` function has the following parameters:

Table 21. WSInvokeDL function parameters

Parameter	Format	Description
WSService	String	This web service name is defined in the `/definitions/service` element of the WSDL file.
WSEndPoint	String	The web service endpoint URL of the target web service.
WSMethod	String	The web service method defines which method you would like to call in `WSInvokeDL()`.
WSParams	Array	The web services operation parameters are defined by `/definitions/message/part` elements in the WSDL file. It comprises an array that contains all of the parameters that are required by the specified web service operation.
`callProps`	String, Boolean, integer	The optional container in which you can set any of the properties, which are listed in the callProps properties section.

`callProps` properties

Remember: Any options that are set in `callProps` must precede the actual call to `WSInvokeDL`.

- **`Chunked`** specifies whether the request can be chunked.
- **`MTOM`** enables or disables the Message Optimization for the SOAP message.
- **`CharSet`** sets the encoding other than UTF-8.
- **`HTTP`** the default HTTP version is 1.1. You can use this property to set the protocol version to 1.0.
- **`ReuseHttpClient`** enables the underlying infrastructure to reuse the HTTP client if one is available. The **`ReuseHttpClient`** is useful if the client is using HTTPS to communicate with the server. The SSL handshake is not repeated for each request. The parameter must be set to true or false.
- **`EnableWSS`** enables web Service Security. If you specify **`EnableWSS`**, you must also specify the following properties:
  - **`WSSRepository`**, which specifies the path location of WSS Repository.
  - **`WSSConfigFile`**, which specifies configuration file for **`EnableWSS`**.
- **`Username`** specifies the user name for basic authentication.
- **`Password`** specifies the password for basic authentication.
- **`PreemptiveAuth`** enables Preemptive Authentication.
- **`Timeout`** this property is used in a blocking scenario. The client system times out after it has waited the specified amount of time.

  You can optionally set a global web Service DSA call timeout property called `impact.server.dsainvoke.timeout`. The property must be added to the Netcool/Impact server property file, `<servername>_server.props`.

The value is set in milliseconds, for example, `impact.server.dsainvoke.timeout=30000` (30 seconds).

When you set the properties in any of the `.props` files, restart theNetcool/Impact server to implement the changes.

If the `impact.server.dsainvoke.timeout` property is set, all WSInvokeDL calls use the same timeout setting.

- **`MaintainSession`** sets the session management to enabled status. When session management is enabled, the system maintains the session-related objects across the different requests. The parameter must be set to true or false.
- **`CacheStub`** caches generated stubs. This value must be set to **`true`** if either or both of the following properties are enabled, **`ReuseHttpClient`**, **`MaintainSession`**. Examples of usage:

  ```
  callProps.CacheStub=true;
  ```

  ```
  callProps.ReuseHttpClient = true;
  ```

Examples

Remember: Any options that are set in `callProps` must precede the actual call to `WSInvokeDL`.

Apart from its primary usage, the `callProps` container can be used to enable security. For example, if the basic authentication is enabled through the wizard, the sample policy contains the following lines:

```
callProps.Username="username";
callProps.Password="password";
```

The following example shows how to use the `WSInvokeDL` function to send a message to the target web service.

Example using IPL:

```
ServiceName = "StockQuote";
EndPointURL = "http://www.webservicex.net/stockquote.asmx"
MethodName = "GetQuote";
ParameterArray = { "IBM" }

[Return] = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams, [callProps])
```

Example using JavaScript:

```
ServiceName = "StockQuote";
EndPointURL = "http://www.webservicex.net/stockquote.asmx";
MethodName = "GetQuote";
ParameterArray = [ "IBM" ];

Results = WSInvokeDL(WSService, WSEndPoint, WSMethod, WSParams, [callProps])
```

WSNewEnum

The WSNewEnum function returns an enumeration value to a target web service.

Syntax

This function has the following syntax:

```
[Return] = WSNewEnum(EnumType, EnumValue);
```

Parameters

The WSNewEnum function has the following parameters.

Table 22. WSNewEnum function parameters

Parameter	Format	Description
EnumType	String	The enumeration class name that exists in the package that is created by nci_compilewsdl
EnumValue	String	The enumeration value to return

Return Value

A new enumeration type and value.

Example

The following example shows how to use the WSNewEnum function to send a message to the target web service.

```
euro = WSNewEnum("net.webservicex.www.Currency", "EUR");
usd = WSNewEnum("net.webservicex.www.Currency", "USD");
```

Writing Web services DSA policies

You can complete the following tasks with the web services DSA in a Netcool/Impact policy:

- Send messages to a web service
- Handle data that is returned from a web service as a message reply

Sending messages

You can use the web services DSA to send messages.

Procedure

1. Call WSSetDefaultPKGName.
2. Add message parameters with any required data.
3. Call WSInvokeor WSInvokeDL.

   When a WSDL file is compiled with **nci_compilewsdl** or by the web services DSA wizard, you must use the WSInvokeDL() function to make web services calls.

Calling WSSetDefaultPKGName

The default package used for communication with the web service is set by the WSSetDefaultPKGName function.

The package name can be the name you supplied to the nci_compilewsdl script when you compiled the WSDL file for the web service. This name is also the name of the JAR file created by this script, without the .jar suffix. The package name can also be any other Java package that resides in the CLASSPATH and contains the class definition of an object you want to use with the WSNewObject or WSNewArray functions (for example, java.util).

To set the default package, you call WSSetDefaultPKGName and pass the name of the package, without the .jar suffix.

Example

The following example shows how to set the default package:

```
WSSetDefaultPKGName("google");
```

In this example, `google.jar` is the package you created when you compiled the WSDL file for the web service.

Note: If you do not call this function before you call `WSNewArray` or `WSNewObject`, you must explicitly specify the package name in those function calls.

Examples using web services DSA functions

The following examples illustrate how the web services DSA functions and demonstrates its abilities.

Example using web services DSA functions to create a real-time stock quote service

You can call a combination of web services DSA functions to create a policy. In IPL, you can use the syntax `varName.subVarName = value` to set a variable. In JavaScript, you use a set method to set a variable, which is the the syntax is `varName.setsubVarName(value)`. Here is an example that uses`WSSetDefaultPKGName`, `WSNewObject`, `WSNewSubObject`, and `WSInvokeDL` functions in a policy in IPL:

```
WSSetDefaultPKGName("impactstockquote");
endpoint ="http://www.webservicex.net/stockquote.asmx";

quoteDoc=WSNewObject("net.webservicex.www.GetQuoteDocument");

quote = WSNewSubObject(quoteDoc, "GetQuote");
quote.Symbol="IBM";

params = { quoteDoc };
return = WSInvokeDL("StockQuote", endpoint, "GetQuote", params);
result = return.GetQuoteResponse.GetQuoteResult;
log("result = " + result);
```

The following example is the same but uses JavaScript, where the `params = [quoteDoc];` value is enclosed in braces (`[]`).

```
WSSetDefaultPKGName("impactstockquote");
endpoint ="http://www.webservicex.net/stockquote.asmx";

quoteDoc=WSNewObject("net.webservicex.www.GetQuoteDocument");

quote = WSNewSubObject(quoteDoc, "GetQuote");
quote.setSymbol("IBM");

params = [ quoteDoc ];
return = WSInvokeDL("StockQuote", endpoint, "GetQuote", params);
result = return.GetQuoteResponse.GetQuoteResult;
log("result = " + result);
```

Example that uses web services DSA functions to create a Global Weather service

The policy in IPL included the following web services DSA functions: `WSSetDefaultPKGName`, `WSNewObject`, `WSNewSubObject`, and `WSInvokeDL`.

```
WSSetDefaultPKGName("impactglbweather");
endpoint ="http://www.webservicex.net/globalweather.asmx";
weatherdoc=WSNewObject("net.webservicex.www.GetWeatherDocument");

weather = WSNewSubObject(weatherdoc, "GetWeather");
weather.CityName = "New York";
weather.CountryName = "United States";
params = { weatherdoc };
return = WSInvokeDL("GlobalWeather", endpoint, "GetWeather", params);
result = return.GetWeatherResponse.GetWeatherResult;
log("result = " + result);
```

The following example is the same but uses JavaScript, where the `params` value is enclosed in braces (`[]`).

```
WSSetDefaultPKGName("impactglbweather");
endpoint ="http://www.webservicex.net/globalweather.asmx";
weatherdoc=WSNewObject("net.webservicex.www.GetWeatherDocument");

weather = WSNewSubObject(weatherdoc, "GetWeather");
weather.setCityName("New York");
weather.setCountryName("United States");
params = [ weatherdoc ];
return = WSInvokeDL("GlobalWeather", endpoint, "GetWeather", params);
result = return.GetWeatherResponse.GetWeatherResult;
log("result = " + result);
```

Example that uses web services DSA functions to create a currency converter service

The policy in IPL, includes the following web service DSA functions: `WSSetDefaultPKGName`, `WSNewObject`, `WSNewSubObject`, `WSInvokeDL`, and `WSNewEnum`.

```
WSSetDefaultPKGName("impactcurrencyconverter");
endpoint ="http://www.webservicex.net/CurrencyConvertor.asmx";
convDoc=WSNewObject("net.webservicex.www.ConversionRateDocument");

rate = WSNewSubObject(convDoc, "ConversionRate");

fromCur = WSNewEnum("net.webservicex.www.Currency", "EUR");
rate.FromCurrency = fromCur;
toCur = WSNewEnum("net.webservicex.www.Currency", "USD");
rate.ToCurrency = toCur;

params = { convDoc };
return = WSInvokeDL("CurrencyConvertor", endpoint, "ConversionRate", params);
result = return.ConversionRateResponse.ConversionRateResult;
log("result = " + result);
log("--------------------------------");
```

The following example is the same but uses JavaScript, where the `params` value is enclosed in square braces `[]`.

```
WSSetDefaultPKGName("impactcurrencyconverter");
endpoint ="http://www.webservicex.net/CurrencyConvertor.asmx";
convDoc=WSNewObject("net.webservicex.www.ConversionRateDocument");

rate = WSNewSubObject(convDoc, "ConversionRate");

fromCur = WSNewEnum("net.webservicex.www.Currency", "EUR");
rate.setFromCurrency(fromCur);
toCur = WSNewEnum("net.webservicex.www.Currency", "USD");
rate.setToCurrency(toCur);

params = [ convDoc ];
```

```
return = WSInvokeDL("CurrencyConvertor", endpoint, "ConversionRate", params);
result = return.ConversionRateResponse.ConversionRateResult;
log("result = " + result);
log("--------------------------------");
```

Web services listener

The web services listener is a service that provides a Netcool/Impact web services interface to other applications to run Netcool/Impact policies.

Before you can use the web services listener, you must assign the **impactFullAccessUser** and **bsmAdministrator** roles to the user who uses the web services. For more information, see the *Authentication for the web services listener* topic in the Netcool/Impact information center or in the *DSA Reference Guide* PDF file.

Web services listener process

Policy requests from external applications are managed by the web services listener.

The web services listener listens at an HTTP port for SOAP/XML messages from external applications. These messages make requests to Netcool/Impact to run a policy. When the listener receives a request, it sends it to the Netcool/Impact policy engine along with any runtime parameters and returns the policy results to the calling application via the HTTP port.

The requests can also be made over HTTPS protocol.

WSDL file

The Web Services Description Language (WSDL) file is an XML document that describes the web services interface.

The WSDL file specifies five messages that define the terms of communication between Tivoli Netcool/Impact and calling applications. These messages allow calling applications to log in to Tivoli Netcool/Impact and to request that a policy be run. The messages are also used to respond to login requests and return policy results. The WSDL file also specifies the types of data that can be passed in the body of the messages.

Setting up the web services listener

The web services listener is automatically installed when you install Netcool/Impact.

Before you can use the web services listener, you must assign the **impactFullAccessUser** and **bsmAdministrator** roles to the user who uses the web services. For more information, see the *Authentication for the web services listener* topic in the Netcool/Impact information center or in the *DSA Reference Guide* PDF file.

Following the installation, you can obtain the web services client information, including the WSDL file and a set of utilities that help you work with web services, at the `$IMPACT_HOME/integrations/web-service-listener` directory. Table 1 shows the files that are provided with the web services listener:

Table 23. Web Services Listener contents

File	Description
`ImpactWebServiceListenerDL.wsdl`	Web service listener WSDL File.
`WSListenerTestPolicy.ipl`	Sample Policy.
`WSTestDL.java`	Sample client.
`README`	Readme file.
`bin/test_wslistener`	Script that runs the sample client.
`/lib`	This directory contains JAR files for sample application.
`$IMPACT_HOME/bin/nci_findendpoint.sh`	Script that you can use to find the SOAP endpoint for an Impact Server.

Writing web services listener policies

Web service listener policies are run in response to web messages that are sent to Tivoli Netcool/Impact from other applications.

The web messages that are sent to Tivoli Netcool/Impact specify the name of the policy to be run and a set of runtime parameters. External applications use runtime parameters to pass data to the policy. The web services listener does not pass an event container to the policy engine. Web services listener policies return data to calling applications in the form of a data item that is called `WSListenerResult`. The policies return one data item at a time.

Runtime parameters

You can use Netcool/Impact to define parameters in a web services listening policy that, when triggered, automate, a policy.

Runtime parameters in web services listener policies are handled in the same way web services listener policies handles runtime parameters in any other policy. You can use the variable name to reference the parameters in the policy. No initialization of the variables is required.

For example, if an incoming web services message contains runtime parameters named `Param1`, `Param2`, and `Param3`, when it runs the policy the web services listener creates new variables in the policy context with those parameter names. The following code shows how to reference those variables in a policy:

```
// Log incoming runtime parameters

Log("Value of Param1: " + Param1);
Log("Value of Param2: " + Param2);
Log("Value of Param3: " + Param3);
```

Note that all runtime parameters in a web services listener policy are strings. No other type of value can be passed to such a policy from calling applications.

WSListenerResult

`WSListenerResult` is a special data item that contains the result of a web services policy.

You can use `NewObject` function to create the `WSListenerResult` data and populate its member variables with values. When the policy terminates, this data item is passed to the web services listener to be returned to the calling application.

The following example shows how to create the `WSListenerResult` data item and populate its member values.

```
WSListenerResult = NewObject();
WSListenerResult.Node = "192.168.1.1";
WSListenerResult.Location = "New York";
WSListenerResult.Summary = "Node not responding to ping.";
```

`WSListenerResult` can contain other data types. The caller parses the object to get the right data from the result. The name contains the field name through which the caller can identify the type of data that is used.

For example, the "SERVICEREQUESTIDENTIFIER" column from the database, is an Integer.

The assignment `WSListenerResult.SERVICEREQUESTIDENTIFIER=_result[0] .SERVICEREQUESTIDENTIFIER` assigns the Integer value to the result. The result is the return value from the GetByFilter function. If the value of the service request is "1", then:

- The getValue method from the policyExecutionResult returns 1.
- The getName method from the policyExecutionResult returns SERVICEREQUESTIDENTIFIER.

Writing applications that call into Web services

When you write applications that call into the Web services, you must have the following information:

- Location of the SOAP endpoint.
- WSDL file.

SOAP endpoint

The Simple Object Access Protocol (SOAP) endpoint is a URL. It identifies the location on the built-in HTTP service where the web services listener listens for incoming requests. Calling applications must specify this endpoint when they send web services messages to Netcool/Impact.

The endpoint URL varies depending on the configuration of Netcool/Impact. The default is:

```
http://<hostname>:<port>/<clustername>_
<servername>_jaxrpc/impact/ImpactWebServiceListenerDLIfc".
```

where <*hostname*> is the name of the system where Netcool/Impact is installed, <*port*> is the port number that is used by the built-in HTTP service, <*clustername*>is the name of the server cluster, and <*servername*>is the name of the Impact Server instance. The default port number is 9080.

The following example shows the endpoint URL for a web services listener that is running on a system named impact_01 using the default port, where the name of the server cluster is NCICLUSTER and the name of the server instance is NCI.

```
http://impact_01:9080/NCICLUSTER_NCI_jaxrpc/impact/ImpactWebServiceListenerDLIfc".
```

You can also determine the SOAP endpoint by using the `nci_findendpoint` script in the `$IMPACT_HOME/bin` directory. When you run this script, it connects to the Netcool nameserver, looks up the SOAP endpoint, and prints the URL to the standard output. The syntax of `nci_findendpoint` is as follows:

```
nci_findendpoint server_name
```

where *`server_name`* is the name of the Impact Server cluster (for example, *`NCI`*).

Authentication for the web services listener

Before you can use the web services listener, you must assign the **impactFullAccessUser** and **bsmAdministrator** roles to the user who uses the web services.

When the external application connects to a web service URL, it uses http authentication. The first time an external application does this, it prompts the user for a user name and password. After you enter the user name and password, Netcool/Impact uses the user name and password that the user enters for future authentications. To facilitate these authentications, you must assign the **bsmAdministrator** role to the user who uses the web services listener.

When the web service listener connects to Netcool/Impact to run a policy, it requests a Netcool/Impact user name and password to authenticate. This user must have the authorization to run policies. To assign this authorization to the user, assign the **impactFullAccessUser** role to the user who uses the web services listener.

For more information about how to assign these roles, see **Working with command-line tools** > **Using WebServices through the command line** > **Mapping groups, and users to roles** on the Netcool/Impact information center at or in the *Administration Guide* PDF file.

WSDL file

The Web Services Description Language (WSDL) file is an XML document that describes the web services interface.

The WSDL file specifies five messages that define the terms of communication between Tivoli Netcool/Impact and calling applications. Calling applications use these messages to log in to Tivoli Netcool/Impact and to request the execution of a policy. You can also use the messages to respond to login requests and return policy results. The WSDL file also specifies types of data that can be passed in the body of the messages.

The WSDL specifies the following messages:
- `ImpactWebServiceListener_login`
- `ImpactWebServiceListener_loginResponse`
- `ImpactWebServiceListener_runPolicy`
- `ImpactWebServiceListener_runPolicyResponse`
- `WSListenerException`

ImpactWebServiceListener_login

The `ImpactWebServiceListener_login` message requests a login to the Impact Server.

User access for the impactFullAccessUser roles is needed to complete the task.

Table 1 shows the parameters in `ImpactWebServiceListener_login`.

Table 24. ImpactWebServiceListener_login Message Parameters

Parameter	Description
`userID`	The user must be either the `tipadmin` user or a user with the `bsmAdminstrator` role assigned. For more information about how to assign these roles, see *Working with command-line tools, Using WebServices through the command line* and the topic *Mapping groups, and users to roles* in Netcool/Impact information center or in the *Administration Guide* PDF file.
`password`	Valid password.

The calling application must send a `ImpactWebServiceListener_login` message before it sends any other messages. The web services listener responds by returning a message of type `ImpactWebServiceListener_loginResponse`.

ImpactWebServiceListener_loginResponse

The `ImpactWebServiceListener_loginResponse` message is sent by the web services listener in response to a login request from a calling application.

The `ImpactWebServiceListener_loginResponse` contains a single parameter named result. The value of this parameter is an object ID that identifies the login session. Additional calls to the web services interface from the calling application must pass the login ID.

ImpactWebServiceListener_runPolicy

The `ImpactWebServiceListener_runPolicy` message requests that Tivoli Netcool/Impact run the specified policy.

Table 1 shows the parameters in `ImpactWebServiceListener_runPolicy`.

Table 25. ImpactWebServiceListener_runPolicy

Parameter	Description
`objId`	Result value that is returned by `ImpactWebServiceListener_loginResponse`. This value identifies the login session for the calling application.
`policyName`	Name of the policy to be run.
`policyUserParams`	Array of runtime parameters to pass to the policy, where each parameter is represented in the WSDL file as a variable of the complex type `WSPolicyUserParameter`. You must set the values of the `format`, `name`, and `value` elements of each parameter in the order that they are displayed in the WSDL file. The required value of the format element is `String`. The value of the `name` element is the name of the parameter as it will be handled in the policy. The value of the `value` element is the parameter value.
`wantResult`	Specifies whether to return the results of the policy to the calling application.

A calling application sends this message after it completes a successful login. The web services listener responds by returning a message of type `ImpactWebServiceListener_runPolicyResponse`.

ImpactWebServiceListener_runPolicyResponse

The `ImpactWebServiceListener_runPolicyResponse` message is sent by the web services listener in response to a request from a calling application to run a policy.

The `ImpactWebServiceListener_runPolicyResponse` contains a single parameter `result`. This parameter contains an array of name-value pairs that correspond to the member variables in the `WSListenerResult` data item that is returned by the policy.

The web services listener sends this message to a calling application if the `wantResult` parameter was specified as true in the originating `ImpactWebServiceListener_runPolicy` message.

WSListenerException

The `WSListenerException` message is sent by the web services listener in response to invalid messages from a calling application.

The `WSListenerException` contains a single parameter named `WSListenerException` that provides detail about the error.

Creating policies by using the web services wizard

You can use the web Services wizard to develop policies. To do so, you connect to the GUI and follow the on-screen prompts.

Procedure

1. In the **Policies** tab, select the arrow next to the **New Policy** icon. To open theWeb Service Invoke-Introduction window select **Use Wizard** > **Web Services**.
2. In the Web Service Invoke-Introduction window, type in your policy name in the **Policy Name** field, 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.

   In instances where the GUI server is installed separately from the back-end server, the file path for the WSDL file refers to the back-end 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 back-end Impact Server host and the GUI server host.

   Note: If the WSDL file contains XSD imports, these files are provided separately. The WSDL files and related XSD files must be placed in a directory with no spaces.
4. In the **Client Stub** area, select one of the following available options:
   - **Select a previously generated client stub for the above WSDL file:**

     Select one of the existing client stub files from the list menu.
     - **Currency.jar**
     - **Stock.jar**
     - **length.jar**

     The **Package Name** field is automatically completed. Select the **Edit** check box to modify the package name.
   - **Provide a package name for the new client stub:**

     Select this option to create a client stub file. Complete the **Package Name** field for the new client stub file.

Click **Next**.

5. In the Web Service Invoke-Web Service Name, Port and Method window, select the general web service information for the following items: **Web Service**, **Web Service Port Type**, and **Web Service Method**. Click **Next**.
6. In the Web Service Invocation - Web Service Method parameters window, enter the parameters that are required by the target web service method and break down the complex parameter to simple types. Click **Next.**
7. Optional: In the Web Service Invoke-Web Service EndPoint window, you can 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.**
8. The Web Service Invoke-Summary and Finish window is displayed. It shows the details of the policy. Click **Finish** to create the policy.

Creating policies by using policy editor

You can use the policy editor to develop policies.

Procedure

1. Get the latest WSDL file which must match your target web service.
2. Determine the endpoint of your target running web service.
3. Run the `$IMPACT_HOME/impact/bin/nci_compilewsdl` script to compile the target WSDL file. Always place the output JAR file to `$IMPACT_HOME/wslib` directory. Otherwise, Netcool/Impact is not able to find the JAR file at run time.
4. Use policy editor or your favorite editor to write your policy to make web services calls.
5. Run the policy that you created.

Sample policy and sample client

A sample policy and a sample client, which you can use to learn about web services listener.

- `WSListenerTestPolicy.ipl` - sample policy.
- `WSTestDL.java` - sample client.

They are in the `$IMPACT_HOME/integrations/web-service-listener` directory. You can run the sample client by using the `test_wslistener` script that is in the `$IMPACT_HOME/integrations/web-service-listener/bin` directory.

1. To create a Java Client to connect to the web services listener, use the `WSTestDL.java` file that is in the `$IMPACT_HOME/integrations/web-service-listener` directory.
2. To start the policy by using the XML Soap Envelop client such as SoapUI.
   - Login call

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:typ="http://response.micromuse.com/types">
   <soapenv:Header/>
   <soapenv:Body>
      <typ:login>
         <String_1>impact_user_name</String_1>
         <String_2>impact_password</String_2>
      </typ:login>
   </soapenv:Body>
</soapenv:Envelope>:
```

- Run a policy with one input parameter that does not return a result.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:typ="http://response.micromuse.com/types">
   <soapenv:Header/>
   <soapenv:Body>
      <typ:runPolicy>
         <WSListenerId_1>
            <clientId>impact_user_name</clientId>
            <objectId>result_from_Login_call</objectId>
         </WSListenerId_1>
         <String_2>PolicyTestName</String_2>
         <!--Zero or more repetitions:-->
         <arrayOfWSPolicyUserParameter_3>
            <desc>ProductName</desc>
            <format>String</format>
            <label>ProductName</label>
            <name>ProductName</name>
            <value>Impact 6.1.1</value>
         </arrayOfWSPolicyUserParameter_3>
         <boolean_4>false</boolean_4>
      </typ:runPolicy>
   </soapenv:Body>
</soapenv:Envelope>
```

- Run a policy with multiple input parameters that returns a result.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:typ="http://response.micromuse.com/types">
   <soapenv:Header/>
   <soapenv:Body>
      <typ:runPolicy>
         <WSListenerId_1>
            <clientId>impact_user_name</clientId>
            <objectId>result_from_Login_call</objectId>
         </WSListenerId_1>
         <String_2>PolicyTestName</String_2>
         <!--Zero or more repetitions:-->
         <arrayOfWSPolicyUserParameter_3>
            <desc>ProductName</desc>
            <format>String</format>
            <label>ProductName</label>
            <name>ProductName</name>
            <value>Impact 6.1.1</value>
         </arrayOfWSPolicyUserParameter_3>

         <arrayOfWSPolicyUserParameter_4>
            <desc>Company</desc>
            <format>String</format>
            <label>Company</label>
            <name>Company</name>
            <value>IBM</value>
         </arrayOfWSPolicyUserParameter_4>

         <arrayOfWSPolicyUserParameter_5>
            <desc>Version</desc>
            <format>Integer</format>
            <label>Version</label>
            <name>Version</name>
            <value>7</value>
         </arrayOfWSPolicyUserParameter_5>
         <boolean_4>false</boolean_4>
      </typ:runPolicy>
   </soapenv:Body>
</soapenv:Envelope>
```

Integration with third-party web services

Sometimes in the development phase you must change your `wsdl` file and reuse Netcool/Impact web services wizard for testing purposes. Because JVM caches the loaded classes, the wizard cannot recognize the latest changes. Use this procedure to clear the cache.

Procedure

1. Stop the embedded version of WebSphere Application Server.
2. Remove the old JAR file from the `$IMPACT_HOME/wslib` directory.
3. Start the embedded version of WebSphere Application Server.
4. Run the wizard to generate the policy.

Chapter 6. Web services security

Web service DSA has limited support to Web services security standard defined by Oasis-Open. The following security features are supported:

- User name token authentication
- User name token authentication with a plain text password
- Message integrity and non-repudiation with signature
- Encryption
- Sign and encrypt messages

Enabling web services security

Use the following method to enable web message level web services security. You can enable HTTP basic authentication (transport level security) by adding an optional module into the policy.

Procedure

1. Stop all the Impact Servers in the cluster.
2. Complete the following steps for the primary Impact Server. These changes are replicated to the secondary servers in the cluster.
   a. Update the `<IMPACT_HOME>/dsa/wsdsa/wss/conf/wss.xml` file in your Tivoli Netcool/Impact installation directory to set up security features that are required by your web service calls. For most cases, you must update two related XML elements, which are **`OutflowSecurity`** and possibly **`InflowSecurity`** in your `wss.xml` file.
   b. Update the `<IMPACT_HOME>/dsa/wsdsa/wss/conf/wscb.properties` file to set up user ID and password that is required by particular security features. For example, `UsernameToken` or `Signature`. This file has the following format:

      ```
      num=1
      uid.1=client
      pwd.1=apache
      ```

      Note: If there is more than entry in the file, only the `uid.1` and `pwd.1` will be read.
3. Complete the following steps for each Impact Server in the cluster:
   a. Remove the `org.apache.axis2.jar` file in the `tipv2/plugins/` directory, replace it with the `org.apache.axis2-woden.jar` in the `impact/lib3p/` directory, and rename it `org.apache.axis2.jar`.
   b. If security features such as signature or encryption are required by web service calls, a signature property file or encryption property file is needed on impact Java class path. Place property files in the jar that is compiled for the web service. The following is an example property file that is called **`client.properties`**. You must enter each line of code on a separate line.

      ```
      org.apache.ws.security.crypto.provider=
      org.apache.ws.security.components.crypto.Merlin
      org.apache.ws.security.crypto.merlin.keystore.type=jks
      org.apache.ws.security.crypto.merlin.keystore.password=apache
      org.apache.ws.security.crypto.merlin.file=client.jks
      ```

This property file includes information about your keystore file that contains public and private keys that are used for signature or encryption. Except the first entry in the file, you must update the next three entries to reflect information about your own keystore file.

Note: There should be only one **`client.properties`** file in the compiled jar.

4. To enable the web services security feature in web services DSA, open the Policy Editor and add the following module to the policy you develop:

```
callProps = NewObject();
callProps.EnableWSS = true;
callProps.WSSRepository= "/tmp/impact611/impact/dsa/wsdsa/wss";
callProps.WSSConfigFile = "/tmp/impact611/impact/dsa/wsdsa/wss/conf/wss.xml";
```

5. The supporting security feature, `WSInvokeDL()` function is started with an additional **`callProps`** object:

```
result = WSInvokeDL("Sample07", endpoint, "echo", params, callProps)
```

6. Start the remaining, secondary Impact Servers.

Results

The preceding steps enable message level security. You can enable HTTP basic authentication (transport level security) by adding the following module into the policy you develop:

```
callProps=NewObject();
callProps.Username="myName";
callProps.Password="myPassword";
WSInvokeDL(....,paramArray, callProps);
```

Creating a web service policy using web service security

This example shows how to set up a stand-alone Apache Axis2 rampart server with an Netcool/Impact policy to enable Web Service Security.

Before you begin

For information about the Apache Axis2 Rampart security module, see the following URL. http://axis.apache.org/axis2/java/rampart/

Tip: If you are using another Web Service Security Server, make sure to use the client properties and `.jks` files of the server. `Client.properties` and `client.jks` files are used in Step 2 of the example.

- Java SDK or JRE 1.6 and above is required. You can use Impact SDK if you are installing the example in the same system where Netcool/Impact is installed: `IMPACT_HOME/sdk/bin`
- Ant 1.8 or above is required. You can use the Netcool/Impact Ant package `IMPACT_HOME/ant`
- Make sure that the Java and Ant executable files are in the system PATH environment variable.

About this task

This example uses Apache Axis2 version 1.6.2 and rampart version 1.6.2.

1. Set up Rampart as a stand-alone server.
   a. Download Axis2 from the following URL. http://axis.apache.org/axis2/java/core/download.cgi

b. Download the Rampart from the following URL: http://axis.apache.org/axis2/java/rampart/download.html

c. Unpack the files that you downloaded in the previous two steps and set the following environment variables:
   - `AXIS2_HOME=<`*`where axis2 package downloaded and unpacked`*`>`
   - `RAMPART_HOME=<`*`where rampart package downloaded and unpacked`*`>`

d. Copy all the JAR files from `RAMPART_HOME\lib` to `AXIS2_HOME\lib` `cp -rf RAMPART_HOME\lib\* AXIS2_HOME\lib\.`

   Remember: The Rampart package is running on the Windows server and Netcool/Impact is running on the UNIX server. Use the appropriate file system separator according to your operating system.

e. Copy the following two MAR files to `AXIS2_HOME\repository\modules`.
   - `RAMPART_HOME\modules\ rahas-1.6.2.mar`
   - `RAMPART_HOME\modules\rampart-1.6.2.mar`.
   - Use the following commands.

```
Copy  RAMPART_HOME\modules\ rahas-1.6.2.mar   AXIS2_HOME\repository\modules\.
Copy  RAMPART_HOME\modules\ rampart -1.6.2.mar   AXIS2_HOME\repository\modules\.
```

f. Go to the `RAMPART_HOME\samples\basic` directory

```
cd RAMPART_HOME\samples\basic directory
```

   The directory includes several sample examples marked `sample01` to `sample11`. This example uses `sample04`. The `Sample04` application echoes only the message that you typed in the variable.

g. Run the following command to build `sample04` application and start the stand-alone server.

```
ant clean service.04
```

   The command creates all the necessary files and starts a stand-alone application for `sample04`. The port number is displayed in the terminal.

```
Buildfile: E:\opt\rampart-1.6.2\samples\basic\build.xml

clean:
   [delete] Deleting directory E:\opt\rampart-1.6.2\samples\basic\build

check.dependency:

service.04:
    [mkdir] Created dir:
E:\opt\rampart-1.6.2\samples\basic\build\service_repositories\sample04
    [mkdir] Created dir:
E:\opt\rampart-1.6.2\samples\basic\build\service_repositories\sample04\
   services
    [mkdir] Created dir:
E:\opt\rampart-1.6.2\samples\basic\build\service_repositories\sample04\
   modules
     [copy] Copying 2 files to
E:\opt\rampart-1.6.2\samples\basic\build\service_repositories\sample04\modules
    [mkdir] Created dir: E:\opt\rampart-1.6.2\samples\basic\build\temp
    [mkdir] Created dir: E:\opt\rampart-1.6.2\samples\basic\build\temp\META-INF
    [javac] E:\opt\rampart-1.6.2\samples\basic\build.xml:191:
   warning: 'includeantruntime' was not set,
defaulting to build.sysclasspath=last; set to false for repeatable builds
    [javac] Compiling 2 source files to E:\opt\rampart-1.6.2\samples\basic\build\temp
     [copy] Copying 1 file to E:\opt\rampart-1.6.2\samples\basic\build\temp\META-INF
     [copy] Copying 1 file to E:\opt\rampart-1.6.2\samples\basic\build\temp
     [copy] Copying 1 file to E:\opt\rampart-1.6.2\samples\basic\build\temp
      [jar] Building jar:
E:\opt\rampart-1.6.2\samples\basic\build\service_repositories\
    sample04\services\sample04.aar
   [delete] Deleting directory E:\opt\rampart-1.6.2\samples\basic\build\temp
     [java] [SimpleHTTPServer] Starting
     [java] [SimpleHTTPServer] Using the Axis2 Repository
E:\opt\rampart-1.6.2\samples\basic\build\service_repositories\sample04
     [java] [SimpleHTTPServer] Listening on port 8080
     [java] [INFO] Deploying module: addressing-1.6.2 - file:
/E:/opt/rampart-1.6.2/samples/basic/build/service_repositories/sample04/modules
```

```
/addressing-1.6.2.mar
     [java] [INFO] Deploying module: rampart-1.6.2 - file:
/E:/opt/rampart-1.6.2/samples/basic/build/service_repositories/sample04/modules/ra
mpart-1.6.2.mar
     [java] [INFO] Deploying Web service: sample04.aar - file:
/E:/opt/rampart-1.6.2/samples/basic/build/service_repositories/sample04/servic
es/sample04.aar
     [java] [INFO] Listening on port 8080
     [java] [SimpleHTTPServer] Started
```

The example shows that the server is running on port 8080 and no warning or errors occur.

h. Verify the application by going to `http://server:8080/axis2/services` and view the following output.

```
Deployed services
sample04
Available operations
echo
```

Click the `sample04` link to view the WSDL file. The full link to the WSDL file is `http://server:8080/axis2/services/sample04?wsdl`.

Now that service is up and running, the next step is to create a policy and setup Netcool/Impact.

2. Create a Netcool/Impact policy and configure Web Service Security.
   a. Create a policy with the policy wizard in the usual way. The WSDL file is `http://server:8080/axis2/services/sample04?wsdl` and name of the JAR file is `sample04`.

      Remember: When the wizard prompts for end point and security, make sure to enable the security. Select the following option `SOAP message user name authentication` so you do not have to enter a user name and password.
   b. The wizard creates the policy with the following parameters:

```
//Enable web service security
callProps = NewObject();
callProps.EnableWSS = true;
callProps.WSSRepository= "/opt/IBM/tivoli/impact/dsa/wsdsa/wss";
callProps.WSSConfigFile = "/opt/IBM/tivoli/impact/dsa/wsdsa/wss/conf/Sample04_wss.xml";
```

      Tip: The path to the file can vary if you have a different Netcool/Impact installation location.
   c. The xml file `/opt/IBM/tivoli/impact/dsa/wsdsa/wss/conf/Sample04_wss.xml` is created as template.

      You may need to add a statement to update the **`InFlowSecurity`** and **`OutFlowSecurity`** parameters to match the application implementation. See the application `services.xml` file in the `$RAMPART_HOME/samples/basic/<sample directory>` directory for details.

      If you are using a different web service server, you need to update the files accordingly. A full working xml file for this example is located at the end of this section see *Example of Sample04_wss.xml*. Make sure to overwrite the existing template with the updated file.
   d. Update the file `/opt/IBM/tivoli/impact/dsa/wsdsa/wss/conf/wscb.properties to num=1 uid.1=client pwd.1=apache`
   e. Copy or FTP `client.properties` and `client.jks` from `RAMPART_HOME\samples\keys` to the `IMPACT_HOME\wslib` directory.
   f. The wizard creates a JAR file in `<Impact_HOME>/wslib/sample04.jar`. Update the JAR file with `client.properties` and `client.jks` by using the following command:

```
<Impact_HOME>/sdk/bin/jar –uf sample04.jar <Impact_HOME>/wslib/ client.*
```

To verify that the JAR file was updated:

```
<Impact_HOME>\sdk\bin\jar –tf sample04.jar |grep
```

to display the content of the JAR file.

g. Move the `client.properties` and `client.jks` from the `wslib` directory
h. Restart Netcool/Impact.
i. Run the policy that you created with the policy wizard.

Results

The following results are displayed.

```
Parser log: Web Service call echo return result:
<ns:echoResponse xmlns:ns="http://sample04.samples.rampart.apache.org"
xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <ns:return>Hello from Impact 7 Server</ns:return>
</ns:echoResponse
```

User name token authentication

Authentication uses a security token to validate the user and determine whether a client is valid in a particular context. User name tokens are used to validate user names and passwords.

Procedure

Update the `$IMPACT_HOME/impact/dsa/wsdsa/wss/conf/wss.xml` parameters in the following way:

```
<parameter name="OutflowSecurity"
<action>
<items>UsernameToken Timestamp</items>
,user>bob</user>
<passwordCallbackClass>com.micromuse.common.util.WSPWCBHandler
</passwordCallbackClass>
</action>
<parameter>
```

In the corresponding `wscb.properties` file, the parameter values are like:

```
num=1
uid.1=bob
pwd.1=bobPassword
```

User name token authentication with a plain text password

Authentication uses a security token to validate the user and determine whether a client is valid in a particular context. User name tokens are used to validate user names and passwords.

Procedure

Update the `$IMPACT_HOME/impact/dsa/wsdsa/wss/conf/wss.xml` parameters:

Remember: If you are using Netcool/Impact bundled with TBSM the path is `$TBSM_HOME/dsa/wsdsa`.

```
<parameter name="OutflowSecurity">
<action>
<items>UsernameToken</items>
<user>bob</user>
<passwordCallbackClass>com.micromuse.common.util.WSPWCBHandler
</passwordCallbackClass>
</action>
<parameter>
```

In the corresponding `wscb.properties` file, the parameter values are like:

```
num=1
uid.1=bob
pwd.1=bobPassword
```

Message integrity and non-repudiation with signature

Procedure

In the `$IMPACT_HOME/impact/dsa/wsdsa/wss/conf/wss.xml` file, make sure the **`OutflowSecurity`** and **`InflowSecurity`** are set in the following way:

```
<parameter name="OutflowSecurity">
 <action>
   <items>Timestamp Signature</items>
   <user>client</user>
   <signaturePropFile>client.properties</signaturePropFile>
   <passwordCallbackClass>com.micromuse.common.util.WSPWCBHandler
   </passwordCallbackClass>
   <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
 </action>
<parameter>
```

```
<parameter name="InflowSecurity">
 <action>
   <items>Timestamp Signature</items>
      <signaturePropFile>client.properties</signaturePropFile>
 </action>
<parameter>
```

The `<user>client</user>` expression here denotes the outgoing Web services calls, which will be signed by the private key of user *client*.
In the corresponding `wscb.properties` file, the parameter values are like:

```
num=1
uid.1=client
pwd.1=apache
```

Encryption

Procedure

In the `$IMPACT_HOME/impact/dsa/wsdsa/wss/conf/wss.xml` file, make sure the **`OutflowSecurity`** and **`InflowSecurity`** are set in the following way:

```
<parameter name="OutflowSecurity">
 <action>
   <items>Encrypt</items>
   <encryptionUser>service</EncryptionUser>
   <encryptionPropFile>client.properties</encryptionPropFile>
 </action>
</parameter>
```

```
<parameter name="InflowSecurity">
 <action>
   <items>Encrypt</items>
       <passwordCallbackClass>com.micromuse.common.util
```

```
        .WSPWCBHandler</passwordCallbackClass>
       <decryptionPropFile>client.properties</decryptionPropFile>
 </action>
</parameter>
```

The `<user>client</user>` expression here denotes the outgoing Web services calls, which will be signed by the private key of user *client*.
In the corresponding `wscb.properties` file, the parameter values are like:

```
num=1
uid.1=service
pwd.1=apache
```

All outbound Web service calls will be encrypted by the public key entry with alias **`service`**. The keystore file, as described in `client.properties` file, should contain the public key entry for alias **`service`**. For example, you can get the public key of **`service`** as X.509 certificate and import the certificate into your own keystore.

Sign and encrypt messages

Procedure

In the `$IMPACT_HOME/impact/dsa/wsdsa/wss/conf/wss.xml` file, make sure the **`OutflowSecurity`** and **`InflowSecurity`** are set in the following way:

```
<parameter name="OutflowSecurity">
      <action>
        <items>Timestamp Signature Encrypt</items>
        <user>client</user>
 <passwordCallbackClass>com.micromuse.common.util.WSPWCBHandler
</passwordCallbackClass>
        <signaturePropFile>client.properties</signaturePropFile>
        <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
        <encryptionKeyIdentifier>SKIKeyIdentifier</encryptionKeyIdentifier>
        <encryptionUser>service</encryptionUser>
      </action>
</parameter>
```

```
<parameter name="InflowSecurity">
      <action>
        <items>Timestamp Signature Encrypt</items>
<passwordCallbackClass>
com.micromuse.common.util.WSPWCBHandler
</passwordCallbackClass>
        <signaturePropFile>client.properties</signaturePropFile>
      </action>
</parameter>
```

In the corresponding `wscb.properties` file, the parameter values are like:

```
num=2
uid.1=service
pwd.1=apache
uid.2=client
pwd.2=apache
```

Chapter 7. Working with web services and WSDM

[Important: This feature is deprecated.] Web Services Distributed Management (WSDM) is a web services framework that manages resources and consumers use to exchange information related to the status of systems, applications, and devices on a network.

WSDM overview

[Important: This feature is deprecated.] Web Services Distributed Management (WSDM) facilitates the exchange of systems status, applications, and devices on a network information.

A manageability resource can provide information about its status by using a web services interface. For example, a WSDM-enabled computer system might provide information about its CPU usage, memory usage and disk space. A manageability consumer is an entity that actively retrieves status information from the resource, or passively receives notifications from it.

For more information about WSDM, see the WSDM home page at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm`.

WSDM support

Netcool/Impact can act as a consumer of status information from one or more manageability resources. Netcool/Impact can retrieve status information, update status information, and actively exchange other kinds of information with a resource. Netcool/Impact cannot passively receive asynchronous notifications that originate with another WSDM entity.

Setting Up WSDM support

WSDM support is a built-in feature of the Netcool/Impact web services DSA. You are not required to complete any setup or configuration steps to use WSDM features in Netcool/Impact. You do not need to compile a WSDL file and install it into Netcool/Impact before you use the WSDM support.

WSDM support

You can use the WSDM functions, provided by the Netcool/Impact policy language (IPL), to retrieve status information, update status information, and actively exchange other kinds of information with a manageability resource.

Writing WSDM policies

[Important: This feature is deprecated.] The Netcool/Impact policy language (IPL) provides functions to retrieve property, update property, and exchange other messages.

Retrieving property values

The IPL provides a function named `WSDMGetResourceProperty` that you can use to retrieve property values from a Web Services Distributed Management (WSDM)

© Copyright IBM Corp. 2006, 2014

resource. When this function is encountered in a policy, a web services request is issued to the specified WSDM endpoint and the results of the request is returned to the policy in the form of an array. You can handle the results of this function in the same way you handle any other type of array.

Updating property values

The IPL provides a function named `WSDMUpdateResourceProperty` that you can use to set property values that are managed by a WSDM resource. When this function is encountered in a policy, a web services request is issued to the specified endpoint and the new value or values are included as a message parameter.

Exchanging other messages

The IPL provides a function named `WSDMInvoke` that you can use to make other calls to a web services API at the specified endpoint. When this function is encountered in a policy, a web services request is issued to that endpoint and the results of the request is returned to the policy in the form of an array. The structure and returned data for the message is defined by the WSDM resource.

WSDMGetResourceProperty

[Important: This feature is deprecated.] The `WSDMGetResourceProperty` function retrieves the value of a management property that is associated with a Web Services Distributed Management (WSDM) managed resource.

You can use this function to retrieve information about the status of a WSDM-enabled system, application, or device.

To retrieve the property value, you call `WSDMGetResourceProperty` and pass the URI of the WSDM endpoint reference and a flattened XML Qname that specifies which property to retrieve.

Syntax

```
Array = WSDMGetResourceProperty(endPointRef, methodName, propQName)
```

Parameters

The `WSDMGetResourceProperty` function has the following parameters.

Table 26. WSDMGetResourceProperty function parameters

Parameter	**Format**	**Description**
endPointRef	String	URI that specifies the endpoint where the WSDM resource is located.
UserName	String	Optional: a user name is required by the web service for SOAP authentication, if any. If no user name is required, omit this parameter.
Password	String	Optional: a password is required by the web service for SOAP authentication, if any. If no password is required, omit this parameter.

Table 26. WSDMGetResourceProperty function parameters (continued)

Parameter	Format	Description
propQName	String	Flattened XML Qname that specifies the management property to retrieve. The format for the flattened Qname is *namespace*:*localname* [*URI*], where *namespace* is the XML namespace where the property is defined, *localname* is the name of the XML element that contains the property and *URI* is the endpoint where the WSDM resource is located. For more information about QNames, see the XML specifications at http://www.w3.org.

Return Value

The WSDMGetResourceProperty function returns the property value to the policy as an array. For properties that consist of a single value, the value is stored in the first array element. For properties that consist of more than one value, the values are stored in the array in the order that they are retrieved from the WSDM resource. In most cases, this function returns an array that contains a single property value.

Example

The following example shows how to use WSDMGetResourceProperty to retrieve a management property named MemoryInUse from the endpoint http://www.example.com/wsdm-endpoint.

```
// Specify endpoint URI and flattened QName

MyEndPoint = "http://www.example.com/wsdm-endpoint";
MyQName = "wsrl:MyProperty [http://www.example.com/wsdm-endpoint]";

// Call WSDMGetResourceProperty and pass the endpoint
// and QName and runtime parameters

MyResult = WSDMGetResourceProperty(MyEndPoint, MyQName);

// Print the value of the property to the policy log

Log("Value of MyProperty is " + MyResult[0]);
```

WSDMUpdateResourceProperty

[Important: This feature is deprecated.] The WSDMUpdatetResourceProperty function updates the value or values of a management property that is associated with a Web Services Distributed Management (WSDM) managed resource.

You can use this function to set information about the state of a WSDM-enabled system, application, or device.

To update the property value, call WSDMUpdateResourceProperty and pass the URI of the WSDM endpoint reference, a flattened XML Qname that specifies the property and an array of new property values.

Syntax

WSDMUpdateResourceProperty(*endPointRef*, *propQName*, *params*)

Parameters

The `WSDMUpdateResourceProperty` function has the following parameters.

Table 27. WSDMUpdateResourceProperty function parameters

Parameter	Format	Description
endPointRef	String	URI that specifies the endpoint where the WSDM resource is located.
propQName	String	Flattened XML Qname that specifies the management property to update. The format for the flattened Qname is *namespace*:*localname* [*URI*], where *namespace* is the XML namespace where the property is defined, *localname* is the name of the XML element that contains the property and *URI* is the endpoint where the WSDM resource is located. For more information about QNames, see the XML specifications at `http://www.w3.org`.
ArrayOfValues	Array	An array that contains the value or values of the property. For properties that consist of a single value, you must store the value in the first array element. For properties that consist of more than one value, you must store the values in the array in the order that they are managed by the WSDM resource. In most cases, the property consists of a single value.
UserName	String	Optional: a user name is required by the web service for SOAP authentication, if any. If no user name is required, omit this parameter.
Password	String	Optional: a password is required by the web service for SOAP authentication, if any. If no password is required, omit this parameter.

Example

The following example shows how to use `WSDMUpdateResourceProperty` to update a management property named `MemoryInUse` from the endpoint `http://www.example.com/wsdm-endpoint`.

This example uses IPL.

```
// Specify endpoint URI, flattened QName and property value

MyEndPoint = "http://www.example.com/wsdm-endpoint";
MyQName = "wsrl:MyProperty [http://www.example.com/wsdm-endpoint]";
Params = {"256"};

// Call WSDMUpdateResourceProperty and pass the endpoint
// and QName and property value

WSDMUpdateResourceProperty(MyEndPoint, MyQName, Params);
```

This example uses JavaScript.

```
// Specify endpoint URI, flattened QName and property value

MyEndPoint = "http://www.example.com/wsdm-endpoint";
MyQName = "wsrl:MyProperty [http://www.example.com/wsdm-endpoint]";
Params = ["256"];

// Call WSDMUpdateResourceProperty and pass the endpoint
// and QName and property value

WSDMUpdateResourceProperty(MyEndPoint, MyQName, Params);
```

WSDMInvoke

[Important: This feature is deprecated.] The `WSDMInvoke` function sends a web services message to a Web Services Distributed Management (WSDM) managed resource.

The structure and content of this message is defined by the receiving WSDM entity. You can use this function to send messages other than messages that retrieve or update a management property to a WSDM resource.

To retrieve the property value, you call `WSDMInvoke` and pass the URI of the WSDM endpoint reference, the method name, and a Java Qname object that specifies which property to retrieve.

Syntax

`*Array* = WSDMInvoke(*endPointRef*, *methodName*, *propQName*)`

Parameters

The `WSDMInvoke` function has the following parameters.

Table 28. WSDMInvoke function parameters

Parameter	Format	Description
endPointRef	String	URI that specifies the endpoint where the WSDM resource is located.
Method	String	Name of the method displayed by the API at the WSDM resource endpoint.
propQName	Object	Java Qname object that specifies the management property to retrieve. You can create an instance of this object in the policy that uses a call to the `NewJavaObject` function provided by the Java DSA.
UserName	String	Optional: a user name is required by the web service for SOAP authentication, if any. If no user name is required, omit this parameter.
Password	String	Optional: a password is required by the web service for SOAP authentication, if any. If no password is required, omit this parameter.

Return Value

The `WSDMInvoke` function returns any values that were sent in the WSDM reply as an array. For properties that consist of a single value, the value is stored in the first array element. For properties that consist of more than one value, the values are stored in the array in the order that they are retrieved from the WSDM resource. In most cases, this function returns an array that contains a single property value.

Example

The following example shows how to use `WSDMInvoke` to remotely start a web services method named `GetResourceProperty`. This method is displayed by the API at the specified WSDM endpoint.

This example uses IPL.

```
// Specify endpoint URI, method name and QName

MyEndPoint = "http://www.example.com/wsdm-endpoint";
MyMethodName = "GetResourceProperty";
MyQNameParams = {"http://docs.oasis-open.org/wsrf/rl-2", "CurrentTime", "wsrl"};
MyQName = NewJavaObject("javax.xml.namespace.QName", qnameParams);

// Call WSDMInvoke and pass the endpoint, the method name
// and the QName object

MyResult = WSDMInvoke(MyEndPoint, MyMethodName, MyQName);

// Print the value of the property to the policy log

Log("Value of MyProperty is " + MyResult[0]);
```

This example uses JavaScript.

```
// Specify endpoint URI, method name and QName
MyEndPoint = "http://www.example.com/wsdm-endpoint";
MyMethodName = "GetResourceProperty";
MyQNameParams = ["http://docs.oasis-open.org/wsrf/rl-2", "CurrentTime", "wsrl"];
MyQName = NewJavaObject("javax.xml.namespace.QName", qnameParams);

// Call WSDMInvoke and pass the endpoint, the method name
// and the QName object

MyResult = WSDMInvoke(MyEndPoint, MyMethodName, MyQName);

// Print the value of the property to the policy log

Log("Value of MyProperty is " + MyResult[0]);
```

Chapter 8. Working with the JMS DSA

You can use the Java Message Service (JMS) data source adapter (DSA) to send and receive JMS messages from within a policy.

The JMS DSA is installed automatically when you install Netcool/Impact.

For detailed information about connecting WebSphere MQ and JMS DSA, see the *Netcool/Impact Integrations Guide*.

Supported JMS providers

Before you can use the Java Message Service (JMS) data source adapter (DSA) to send and retrieve JMS messages, you must obtain the correct set of JMS client libraries.

The JMS client libraries are third-party software components that provide the function that is required to connect to the JMS and JNDI providers in your environment. These libraries are Java JAR files that are distributed with each JMS application. The JMS DSA is compatible with JMS providers that fully support the JMS 1.1 specification.

For more information about JMS 1.1, see the Sun Microsystems Java website at http://java.sun.com/products/jms. Some supported JMS providers include OpenJMS 0.7.7; BEA WebLogic 8.1; Sun Java System Application Server 8 and later; and WebSphere MQ. For information about connecting WebSphere MQ to a JMS DSA, see the *Integrations Guide*.

Configuring JMS DSAs to send and receive JMS messages

You must complete the configuration steps before you can use the Java Message Service (JMS) data source adapter (DSA) to send and retrieve JMS messages.

Procedure

1. Obtain and install the required JMS client libraries.
2. Copy the client JAR files from the JMS client installation directory to the `$IMPACT_HOME/dsalib` directory.
3. Restart the embedded version of WebSphere Application Server.
4. Create a JMS data source, and configure it for the JMS source.

   For more information about creating a JMS data source, see “JMS data source” on page 70.
5. Handle the incoming JMS messages.

   You can handle the incoming JMS messages by using any of these approaches:
   - Write JMS policies that use the JMS data source, and the JMS functions.

     For more information, see “Writing JMS DSA policies” on page 73.
   - Configure the JMSListener service to send JMS events to a policy.

     If you use the JMSListener to send JMS messages to your policy, you do not have to use the ReceiveJMSMessage function to receive them. For more information, see “Handling incoming messages from a JMS message listener” on page 79.

© Copyright IBM Corp. 2006, 2014

Setting up OpenJMS as the JMS provider

You can set up OpenJMS as the Java Message Service (JMS) provider for Netcool/Impact.

Procedure

1. Obtain the OpenJMS libraries from the OpenJMS website (http://openjms.sourceforge.net/.
2. To install OpenJMS, follow the procedure in the getting started information that is available on the OpenJMS website (http://openjms.sourceforge.net/).
3. Copy the OpenJMS client JAR files to the `$IMPACT_HOME/dsalib` directory.

   You can find the OpenJMS client JAR files in the `lib` subdirectory in the OpenJMS installation directory.
4. To restart the embedded version of WebSphere Application Server, use the `ewasImpactStartStop` script that is in the `$IMPACT_HOME/bin` directory.
5. To start the OpenJMS server, use the `startup` script that is in the `bin` subdirectory in the OpenJMS installation directory.
6. Create a JMS data source, and configure it for OpenJMS.

   For more information, see "JMS data source"

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.

JMS data source configuration properties

You can configure the properties for the Java Message Service (JMS) data source.

Table 29. General settings for the JMS data source window

Window element	Description
Data Source Name	Enter a unique name to identify the data source. You can use only letters, numbers, and the underscore character in the data source name. If you use UTF-8 characters, make sure that the locale on the Impact Server where the data source is saved is set to the UTF-8 character encoding.

Table 30. Source settings for the JMS data source window

Window element	Description
JNDI Factory Initial	Enter the name of the JNDI initial context factory. The JNDI initial context factory is a Java object that is managed by the JNDI provider in your environment. The JNDI provider is the component that manages the connections and destinations for JMS. OpenJMS, BEA WebLogic, and Sun Java Application Server distribute a JNDI provider as part of their JMS implementations. The required value for this field varies by JMS implementation. For OpenJMS, the value of this property is **`org.exolab.jms.jndi.InitialContextFactory`**. For other JMS implementations, see the related product documentation.
JNDI Provider URL	Enter the JNDI provider URL. The JNDI provider URL is the network location of the JNDI provider. The required value for this field varies by JMS implementation. For OpenJMS, the default value of this property is **`tcp://hostname:3035`**, where host name is the name of the system on which OpenJMS is running. The network protocol (TCP or, RMI,) must be specified in the URL string. For other JMS implementations, see the related product documentation.
JNDI URL Packages	Enter the Java package prefix for the JNDI context factory class. For OpenJMS, BEA WebLogic, and Sun Java Application Server, you are not required to enter a value in this field.
JMS Connection Factory Name	Enter the name of the JMS connection factory object. The JMS connection factory object is a Java object that is responsible for creating new connections to the messaging system. The connection factory is a managed object that is administered by the JMS provider. For example, if the provider is BEA WebLogic, the connection factory object is defined, instantiated, and controlled by that application. For the name of the connection factory object for your JMS implementation, see the related product documentation.
JMS Destination Name	Enter the name of a JMS topic or queue, which is the name of the remote topic or queue where the JMS message listener listens for new messages.
JMS Connection User Name	Enter a JMS user name. If the JMS provider requires a user name to listen to remote destinations for messages, enter the user name in this field. JMS user accounts are controlled by the JMS provider.
JMS Connection Password	If the JMS provider requires a password to listen to remote destinations for messages, enter the password in this field.
Test Connection	Test the connection to the JMS Implementation. If the test is successful, the system shows the following message: `JMS: Connection OK`

Specifying more JNDI properties for the JMS data source

You can specify more Java Naming and Directory Interface (JNDI) properties by editing the Java Message Service (JMS) data source.

Procedure

1. Open the JMS data source for editing in a text editor of your choice. You can find all data sources in the `$IMPACT_HOME/etc/` directory. The data source file name is **`<servername>_<datasourceName>.ds`**. **`<servername>`** is the name of the Impact Server instance, and **`<datasourceName>`** is the name of your JMS data source as displayed in the data source editor in GUI.
2. Add your JNDI properties in the following format:

   ```
   <datasourceName>.JMS.DSPROPERTY.#.NAME=<property>
   <datasourceName>.JMS.DSPROPERTY.#.VALUE=<property value>
   ```

   # is the property number in a sequence of properties, the starting number is 1, for example:

   ```
   <datasourceName>.JMS.DSPROPERTY.1.NAME=java.naming.factory.initial
    <datasourceName>.JMS.DSPROPERTY.1.VALUE=org.exolab.jms.jndi.
   InitialContextFactory
   <datasourceName>.JMS.DSPROPERTY.2.NAME=java.naming.provider.url
   <datasourceName>.JMS.DSPROPERTY.2.VALUE=tcp://jndi_host:3035
   <datasourceName>.JMS.DSPROPERTY.3.NAME=java.naming.security.principal
   <datasourceName>.JMS.DSPROPERTY.3.VALUE=User1
   <datasourceName>.JMS.DSPROPERTY.4.NAME=java.naming.security.credentials
   <datasourceName>.JMS.DSPROPERTY.4.VALUE=password
   <datasourceName>.JMS.NUMDSPROPERTIES=4
   ```

   The `<datasourceName>.JMS.NUMDSPROPERTIES=<number of properties>` property specifies the number of more properties, 4 in the previous example.

   Note: Use the **`$IMPACT_HOME/bin/nci_crypt`** utility to encrypt the value of the **`java.naming.security.credentials`** property.
3. Save the changes in the data source, and restart the Impact Server to apply the changes.

JMS message listener

The Java Message Service (JMS) message listener service runs a policy in response to incoming messages that are sent by external JMS message providers.

The message provider can be any other system or application that can send JMS messages. Each JMS message listener listens to a single JMS topic or queue. There is one default JMS message listener named JMSMessageListener. You can create as many listener services as you need, each of which listens to a different topic or queue.

A JMS message listener is only required when you want Netcool/Impact to listen passively for incoming messages that originate with JMS message producers in your environment. You can actively send and retrieve messages from within a policy without using a JMS message listener.

JMS message listener service configuration properties

You can configure the properties for the Java Message Service (JMS) listener service.

Table 31. JMSMessageListener Service configuration window

Window element	Description
Service name	Enter a unique name to identify the service.
Policy To Execute	Select the policy that you created to run in response to incoming messages from the JMS service.

Table 31. JMSMessageListener Service configuration window (continued)

Window element	Description
JMS Data Source	JMS data source to use with the service. You need an existing and valid JMS data source for the JMS Message Listener service to establish a connection with the JMS implementation and to receive messages. For more information about creating JMS data sources, see "JMS data source configuration properties" on page 70.
Message Selector	The message selector is a filter string that defines which messages cause Netcool/Impact to run the policy specified in the service configuration. You must use the JMS message selector syntax to specify this string. Message selector strings are similar in syntax to the contents of an SQL WHERE clause, where message properties replace the field names that you might use in an SQL statement. The content of the message selector depends on the types and content of messages that you anticipate receiving with the JMS message listener. For more information about message selectors, see the JMS specification or the documentation distributed with your JMS implementation. The message selector is an optional property.
Durable Subscription: Enable	You can configure the JMS message listener service to use durable subscriptions for topics that allow the service to receive messages when it does not have an active connection to the JMS implementation. A durable subscription can have only one active subscriber at a time. Only a JMS topic can have durable subscriptions.
Client ID	Client ID for durable subscription. It defines the client identifier value for the connection. It must be unique in the JMS Implementation.
Subscription Name	Subscription Name for durable subscription. Uniquely identifies the subscription made from the JMS message listener to the JMS Implementation. If this property is not set, the name of JMS DSA listener service itself is used as its durable subscription name, which is JMSMessageListener by default.
Clear Queue: Clear	Clear the message waiting in the JMSMessageListener queue that has not yet been picked by the EventProcessor service. It is recommended not to do this while the Service is running.
Service: 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.

Writing JMS DSA policies

Java Message Service (JMS) policies send or retrieve JMS messages.

JMS policies use the SendJMSMessage and ReceiveJMSMessage functions, or work with the JMS message listener service.

In a policy, you use the JMS DSA to perform the following tasks:
- Send messages to a JMS topic or queue
- Retrieve messages from a JMS topic
- Queue or handle incoming messages from a JMS message listener

Sending messages to a JMS topic or queue

You can send messages to a Java Message Service (JMS) topic or queue from within a policy.

Procedure

1. Create and configure a JMS data source

   For more information, see "JMS data source" on page 70.
2. Create a message properties context.

   For more information, see "Message properties context" on page 75.
3. Create a message body string or context.

   For more information, see "Creating a message body string or context" on page 76.
4. Call the SendJMSMessage function and pass the values the JMS data source, the message properties context, and the specified message body as runtime parameters.

   For more information about the syntax of the SendJMSMessage function, see "SendJMSMessage."

SendJMSMessage

The SendJMSMessage function sends a message to the specified destination by using the Java Message Service (JMS) DSA.

To send the message, you call the SendJMSMessage function and pass the JMSDataSource, a message properties context, and the message body as runtime parameters.

Syntax

The SendJMSMessage function has the following syntax:

```
SendJMSMessage(DataSource, MethodCallProperties, Message)
```

Parameters

The SendJMSMessage function has the following parameters.

Table 32. SendJMSMessage function parameters

Parameter	Format	Description
`DataSource`	String	Valid, and existing JMS data source.
`MethodCallProperties`	Context	Context that contains message header, and other JMS properties for the message. Custom message properties are supported.
`Message`	String | Context	String or context that contains the body of the message.

Message properties context

The message properties context specifies runtime parameters for the underlying Java Message Service (JMS) client method call that retrieves the message when you call the ReceiveJMSMessage function.

You pass this context as a runtime parameter when you call the SendJMSMessage function in a policy. This message properties context specifies the message header, custom message properties, and the message selector. The table shows the valid JMS message header values.

Table 33. JMS Message Header Values

Property	Description
`DeliveryMode`	Optional. Specifies the JMS delivery mode for the method. Possible values are `1` for non-persistent and `2` for persistent.
`DisableMessageId`	Optional. Specifies whether JMS message IDs are disabled.
`DisableMessageTimeStamp`	Optional. Specifies whether JMS message time stamps are disabled.
`JMSCorrelationID`	Optional. Specifies a JMS correlation ID for the message.
`JMSCorrelationIDAsBytes`	Optional. Specifies a JMS correlation ID for the message as an array of bytes.
`JMSDeliveryMode`	Optional. Specifies a JMS delivery mode. Possible values are `0` for persistent mode and `1` for non-persistent mode.
`JMSDestination`	Optional. Specifies a destination for the message in the form of a JMS-administered object.
`JMSExpiration`	Optional. Specifies an expiration value in milliseconds for the message. If not specified, value is set by the JMS provider.
`JMSMessageID`	Optional. Specifies a JMS message ID for the message.
`JMSPriority`	Optional. Specifies a JMS priority level for the message. JMS supports priority levels from `0` to `9`, with `9` as the highest.
`JMSRedelivered`	Optional. Specifies whether the message is being redelivered. Possible values are `True` or `False`.
`JMSReplyTo`	Optional. Specifies the name of a JMS destination where replies to this message are sent.
`JMSTimeStamp`	Optional. Specifies a time stamp for the message in seconds since the beginning of the UNIX epoch. If not specified, the value is set by the JMS provider.
`JMSType`	Optional. Specifies a JMS message type for the message. Some JMS implementations use a message repository to store defined types of messages. You can use this header value to associate a particular message with a message type.
`Priority`	Same as `JMSPriority`.
`TimeToLive`	Optional. Specifies the length of time that a message is retained by the JMS delivery system before it expires. The default value is `0`, which indicates an unlimited message lifetime.

For more information about the JMS message header, see the documentation that was provided with your JMS application.

Optionally, you can also specify custom message properties. These properties are user-defined and can contain any value. Generally, these properties are used to send meta information about messages that is not otherwise described in the message header.

The following example shows how to create and populate a message properties context:

```
// Call NewObject to create the new context
MsgProps = NewObject();

// Assign message header values as member variables
MsgProps.TimeToLive = 0;
MsgProps.Priority = 5;
MsgProps.DeliveryMode = "PERSISTENT";
MsgProps.ReplyTo = "jms/Topic";

// Assign custom message properties as member variables
MsgProps.Custom1 = "First custom property";
MsgProps.Custom2 = "Second custom property";
```

Creating a message body string or context

You specify the message body by using a string value or a context, depending on whether you want to send a text message or a map message.

To specify the body of a text message, you use a string assignment statement in the policy. When you call SendJMSMessage, you pass this string to the function as a runtime parameter. This example shows how to assign the body of a text message to a string:

```
MsgTextBody = "Body content of text message";
```

To specify the body of a map message, you create a context by using the NewObject function. You assign one member variable for each name-value pair in the map, where the name of the variable corresponds to the name for the pair. When you call SendJMSMessage, you pass this context to the function as a runtime parameter.

This example shows how to create a message body context for a map message. In this example, the names of values in the map are `name`, `location`, and `email`.

```
MsgMapBody = NewObject();

MsgMapBody.name = "John Smith";
MsgMapBody.location = "New York City";
MsgMapBody.email = "jsmith@example.com";
```

Example of sending a map message to a JMS destination

The following example shows how to send a map message to a Java Message Service (JMS) destination by using the SendJMSMessage function.

```
// Set JMSDataSource to a valid and existing JMSDataSource in Impact.
// The destination where the message is sent is obtained from the JMSDataSource.
JMSDataSource = "JMSDS1";

// Create a message properties object and populate its
// member variables with message header properties and custom properties
MsgProps = NewObject();
MsgProps.TimeToLive = 0;
MsgProps.color = "green";
MsgProps.Expiration = 2000;
MsgProps.DeliveryMode = "PERSISTENT";
MsgProps.ReplyTo="queue2";

// Specify custom message properties
MsgProps.Custom1 = "Value 1";
MsgProps.Custom2 = "Value 2";

// Create a map message content and populate its member
// variables where each variable and value represent a name/
// value pair for the resulting map
MsgMapBody = NewObject();
MsgMapBody.name = "sanjay";
MsgMapBody.location = "New York City";
MsgMapBody.facility = "Wall St.";

// Call SendJMSMessage and pass the JNDI properties
// context, the message properties context, the message
// map context and other parameters
SendJMSMessage(JMSDataSource, MsgProps, MsgMapBody);
```

Example of sending a text message to a JMS destination

This example shows how to send a text message to a Java Message Service (JMS) destination by using the SendJMSMessage function.

```
// Set JMSDataSource to a valid and existing JMSDataSource in Impact.
// The destination where the message is sent is obtained from the JMSDataSource.
JMSDataSource = "JMSDS1";

// Create a message properties object and populate its
// member variables with message header properties and custom properties
MsgProps = NewObject();
MsgProps.TimeToLive = 0;
MsgProps.color = "green";
MsgProps.Expiration = 2000;
MsgProps.DeliveryMode = "PERSISTENT";
MsgProps.ReplyTo="queue2";

// Specify custom message properties
MsgProps.Custom1 = "Value 1";
MsgProps.Custom2 = "Value 2";

// Create a text message content
MsgTextBody = "This is the message body";

// Call SendJMSMessage and pass the JNDI properties
// context, the message properties context, the message
// map context and other parameters
SendJMSMessage(JMSDataSource, MsgProps, MsgTextBody);
```

Retrieving JMS messages from a topic or queue

You can retrieve messages from a Java Message Service (JMS) topic or queue from within a policy.

Procedure

1. Create and configure a JMS data source

   For more information, see "JMS data source" on page 70.
2. Create a message properties context.

   For more information, see "Creating a message properties context" on page 78.
3. Call the ReceiveJMSMessage function and pass the values of the JMS data source, and the message properties context as parameters.

For examples of the ReceiveJMSMessage function usage, see "ReceiveJMSMessage."

4. Handle the retrieved message

   For more information, see "Handling a retrieved message" on page 79.

ReceiveJMSMessage

The ReceiveJMSMessage function retrieves a message from the specified Java Message Service (JMS) destination.

To retrieve the message, you call this function and pass a JMSDataSource, and a message properties context as runtime parameters.

Syntax

The ReceiveJMSMessage function has the following syntax:

```
ReceiveJMSMessage(DataSource, MethodCallProperties)
```

Parameters

The ReceiveJMSMessage function has the following parameters:

Table 34. ReceiveJMSMessage function parameters

Parameter	Format	Description
`DataSource`	String	Existing, and valid JMS data source.
`MethodCallProperties`	Context	Context that contains optional `MessageSelector` and `Timeout`.

Creating a message properties context

The message properties context specifies connection information for the underlying Java Message Service (JMS) client method call that retrieves the message when you call the ReceiveJMSMessage function.

You pass this context as a parameter when you call the ReceiveJMSMessage function in a policy. The following table shows the properties that you can set in the message properties context:

Table 35. Message Properties Context

Property	Description
`MessageSelector`	String expression that specifies which message in the topic or queue you want to retrieve. The message selector syntax is similar to the contents of an SQL `WHERE` clause and is defined in the JMS specification.
`Timeout`	Specifies the length of time that a message is retained by the JMS delivery system before it expires. Default value is `0`, which indicates an unlimited message lifetime.

You can create an empty message properties context by passing the NewObject function to the ReceiveJMSMessage as a parameter.

The following example shows how to create a message properties context.

```
// Call NewObject to create the next context
MsgProps = NewObject();
```

```
// Assign a message selector that filters the message to
// retrieve

MsgProps.MessageSelector = "color = 'green' AND custom2 = '1234543'";
```

Handling a retrieved message

The ReceiveJMSMessage function uses three variables to store message information that is retrieved from a Java Message Service (JMS) topic or queue.

Table 1 shows the built-in variables that store the message information:

Table 36. Built-in Message Variables

Variable	Description
`JMSMessage`	JMS message body. If the message is a text message, the value of this variable is a string. If the message is a map message, the value of this variable is a context where each member variable in the context corresponds to a name-value pair in the message map.
`MessageType`	If the message is a text message, the value of this variable is a string "Text". If the message is a map message, the value of this variable is a string "Map".
`JMSProperties`	Custom JMS message properties that are attached to the message.

This example shows how to handle a retrieved message:

```
// Call ReceiveJMSMessage and pass the JNDI properties,
// message properties and other information as runtime parameters
ReceiveJMSMessage(JMSDataSource, MsgProps);

// Print the contents of the message to the policy log
Log("Message type: " + MessageType);
Log("Message properties: " + JMSProperties.Custom1);
Log("Message properties: " + JMSProperties.Custom2);

If (MessageType == "Text") {
    Log("Message body: " + JMSMessage);
} Else {
    Log("Message map value 1: " + JMSMessage.MyValue1);
    Log("Message map value 2: " + JMSMessage.MyValue2);
}
```

Handling incoming messages from a JMS message listener

When a Java Message Service (JMS) message listener receives a message from a JMS destination, it compares the contents of the message to message selectors specified in its configuration.

If the message matches the message selector, or if no selector is specified, the JMS message listener puts the message in its queue. The EventProcessor service picks up the message, and sends it to the policy as an EventContainer.

The JMS message listener uses the message variables that are used when you use the ReceiveJMSMessage function - JMSMessage, MessageType, and JMSProperties - to retrieve a policy. For more information about these variables, see “Handling a retrieved message.” When you handle these variables as set by a JMS message listener, you must reference them by using the @ notation in an IPL policy, or the dot notation in a JavaScript policy, for example, `EventContainer.MessageType`.

This example shows how to handle an incoming message from a JMS message listener by using the @ notation.

```
// Print the contents of the message to the policy log
Log("Message type: " + @MessageType);
Log("Message properties: " + @JMSProperties.Custom1);
Log("Message properties: " + @JMSProperties.Custom2);

If (MessageType == "Text") {
    Log("Message body: " + @JMSMessage);
} Else {
    Log("Message map value 1: " + @JMSMessage.MyValue1);
    Log("Message map value 2: " + @JMSMessage.MyValue2);
}
```

Example of receiving a map message

This example shows how to use the ReceiveJMSMessage function to receive a map message.

The example uses the map message that was used in "Example of sending a map message to a JMS destination" on page 76.

```
/// Use a existing and valid JMSDataSource
JMSDataSource = "JMSDS1";

// Create a message properties object and populate its
// member variables with optional parameters like MessageSelector and Timeout
MsgProps = NewObject();
// MessageSelector is used for filtering incoming messages so that messages
// with properties matching the MessageSelector expression are delivered.
MsgProps.MessageSelector = "color = 'green' AND Custom2 = 'Value 2'";

// Timeout must be specified in milliseconds. This parameter specifies how long the
// MessageConsumer blocks to receive a Message. A Timeout of zero makes the
// MessageConsumer wait indefinitely to receive a message.
MsgProps.Timeout = 6000;

// Call ReceiveJMSMessage and pass the JMSDataSource and message properties
ReceiveJMSMessage(JMSDataSource, MsgProps);

// Print the contents of the message to the policy log
Log("Message type: " +MessageType);
Log("Message prop.Custom1: " + JMSProperties.Custom1);
Log("Message prop.Custom2: " + JMSProperties.Custom2);
If (MessageType == "Text") {
   Log("Message body: " + JMSMessage);
} Else {
   Log("Message map.name: " + JMSMessage.name);
   Log("Message map.location: " + JMSMessage.location);
}
```

The `If (MessageType == "Text")` statement also checks whether the message is a text message, and prints the message to the log, if it is.

Chapter 9. Working with the XML DSA

The XML DSA is a data source adaptor that is used to read and to extract data from any well-formed XML document.

XML DSA overview

The XML DSA is used to read and extract data from any XML document.

The XML DSA can read XML data from files, strings, and HTTP servers by way of the network (XML over HTTP). The Xerces DOMParser 2.6 parser is used for the XML DSA.

The XML DSA is installed with Netcool/Impact so you do not need to complete any additional installation or configuration steps.

Before you can use the XML DSA, you must complete the following tasks:
- Create a set of XML data types that corresponds to the structure of the XML document you want to read with Netcool/Impact. For more information about creating XML data types, see "Creating XML data types" on page 83.
- Set up XML data type mappings that show the relationship between an XML data source, an XML document, and XML data types.
- Write one or more XML DSA policies that read XML data from a file, a string or from an HTTP server over a network.

XML documents

The DSA considers an XML document to be any well-formed set of XML data that descends from a single root element.

This document can be in a string, a text file, or on an HTTP server.

XML DTD and XSD files

XML DTD and XSD files contain a document type description.

You must provide an XML DTD or XSD file for each type of XML document that you want the DSA to read.

XML data types

XML data types are Netcool/Impact data types that represent XML documents and their contents.

The DSA uses the following XML data types:

Super data types
: Super data types represent types of XML documents. The DSA uses one data type for each type of document that it reads.

Element data types
: Element data types represent the elements in an XML document. The DSA requires one such data type for each type of XML element.

© Copyright IBM Corp. 2006, 2014

Super data types

Super data types represent types of XML documents. The DSA uses one data type for each type of document that it reads.

A super data type contains a single data item, called the document data item. This data item represents the instance of the document that the DSA uses. The display name of the document data item is the same as the name of the super data type. The document data item contains a static link to the element data item that represents the root level element of the document.

Element data types

Element data types represent the elements in an XML document. The DSA requires one such data type for each type of XML element.

An element data type contains one field for each attribute in the corresponding XML element. In addition, the data type contains a field that corresponds to the PCDATA value of the element, if any.

Element data types contain one or more data items, called element data items. Each such data item represents an instance of the element in the document.

The hierarchical relationship between XML elements is represented at the data item level by static links. Element data items are statically linked in such a way that each data item contains links to other data items. The other data items represent the child elements of the corresponding element in the XML document.

Element namespaces are a convention that is used by the DSA to show that a set of element data types is related to a single XML document. The DSA uses element namespaces to avoid ambiguity in cases where more than one type of XML document that is used by the DSA has an element of the same name.

XML configuration files

XML configuration files are text files that store mapping information for XML data types.

The DSA reads the configuration files at startup and uses the information during run time to locate the DTD or XSD file and data source for each XML document. For more information about XML configuration files, see “Data type mappings” on page 85.

XML document and data type mapping

The XML DSA provides mapping between an XML document and a set of data types.

The DSA uses the information in an XML DTD or XSD file to understand the structure of the XML data and to map the data to the corresponding data types.

One aspect of the structure of the XML data is the hierarchical relationship between XML elements. The DSA uses static links to map this relationship to the XML data types. Each element data item is linked to its logical child data item by a static link. When you read the XML data in a policy, you use the `GetByLinks` function to traverse the resulting structure. You can also use the embedded linking syntax to traverse the structure.

This example shows a partial XML document, and the linking relationship between the corresponding element data items.

```
<XML_alert id="0123456789">
    <XML_head>
        <XML_sender>IBM</XML_sender>
        <XML_subject>Alert</XML_subject>
    </XML_head>
    <XML_body>
        <XML_node>NodeXYZ</XML_node>
        <XML_summary>Node not responding</XML_summary>
    </XML_body>
</XML_alert>
```

This figure shows the linking relationship between the corresponding element data items:

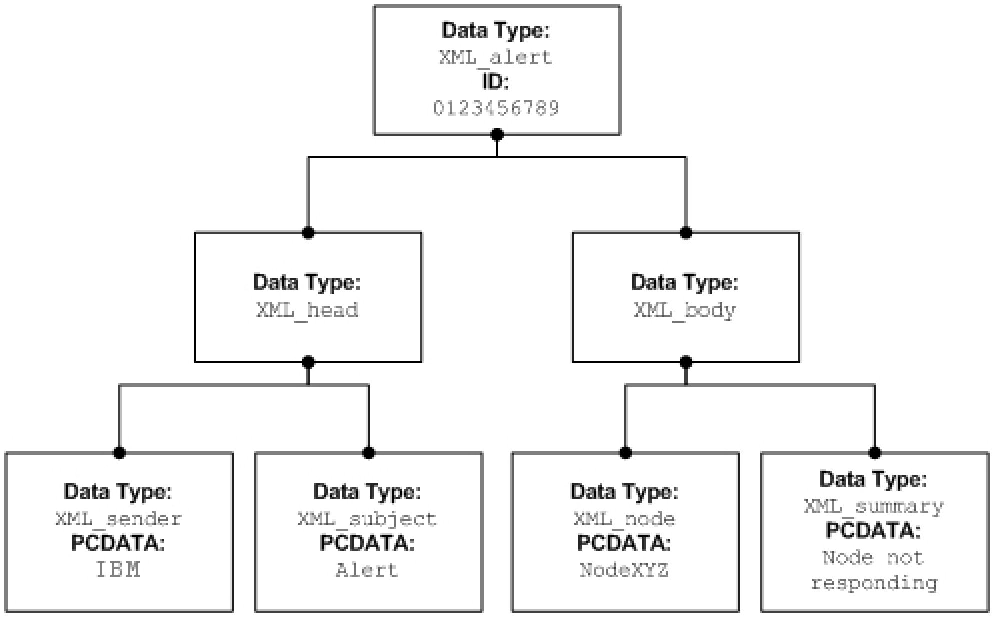


Figure 1. Linking relationships between corresponding element data items

Creating XML data types

You must create XML data types to represent the structure of the XML document that you want to read with Netcool/Impact.

To create the XML data types, you run either the create DTD types script or the create XSD types script, depending on which type of schema you are using. The create types script creates a super data type and then reads the XML DTD or XSD file. The create types script creates one element data type for each type of element that is defined in the file, including the root level element. The script uses the names of the elements in the DTD or XSD file as the names of the element data types. If you specify an element namespace, add a prefix to the name of each element data type. The script then uses the command-line service to insert data types into Netcool/Impact. For more information, see “Create data types scripts” on page 84.

You can also use the XML DSA wizard to automate creating XML data types. For more information about XML DSA wizards, see *Policy wizards* in the *User Interface Guide*.

Important: If you create an XML data type in a server cluster, either by using the wizard or the script, cluster members are updated with the new `.type` files. The following configuration files are not updated:

- `XmlHttpTypes`
- `XmlFileTypes`

The `$IMPACT_HOME/dsa/XmlDsa` will be replicated during startup of the secondary cluster members from the primary server. If you are using XML DSA wizard, or using the scripts provide, the changes will replicate in real time.

Create data types scripts

The XML DSA provides two scripts that you can run from the command line to create XML data types.

You can find these scripts in the `$IMPACT_HOME/dsa/XmlDsa/bin` directory. You use the `CreateDtdTypes` script to create data types from an XML DTD. The script has the following syntax:

```
CreateDtdTypes server_name user password dtdFile type_name namespace_prefix
```

You use the `CreateXsdTypes` script to create data types from an XML XSD. The script has the following syntax:

```
CreateXsdTypes server_name user password xsdFile type_name namespace_prefix
```

Table 37 explains the options that are used with the scripts.

Table 37. Create data type scripts options

Option	Description
server_name	The name of the Impact Server.
user	The name of the Impact Server user.
password	The Impact Server user's password.
dtdFile	The path and file name of the XML DTD file that describes the XML document. Relative to the `$IMPACT_HOME/dsa/XmlDsa/bin` directory.
xsdFile	The path and file name of the XML XSD file that describes the XML document. Relative to the `$IMPACT_HOME/dsa/XmlDsa/bin` directory.
type_name	The name of the resulting super data type.
namespace_prefix	The optional prefix added to the names of element data types. This string is not prefixed to the name of the super data type.

The *CreateDtdTypes* and *CreateXsdTypes* scripts replace any colon character in XML element names with an underscore when you create the data types. For example, if a DTD file contains an element named `netcool:alert`, the create DTD types script creates a corresponding element data type named `netcool_alert`.

Important: The Impact Server must be up for these scripts to run successfully.

Here is an example of the `CreateDtdTypes` script usage on UNIX:

```
./CreateDtdTypes.sh NCI tipadmin netcool ../TOC.dtd XmlStringTOC STEST_
```

Data type mappings

After you create the XML data types, you must set up data type mappings.

A data type mapping is a set of information that shows the relationship between an XML data source, an XML document, and XML data types. The DSA uses this information to map the contents of an XML document to the data types in Netcool/Impact. You must set up one data type mapping for each type of XML document you want Netcool/Impact to read.

Data type mapping information is stored in XML configuration files. The DSA uses the following XML configuration files:

- `XmlFileTypes`
- `XmlHttpTypes`

These files are in the `$IMPACT_HOME/dsa/XmlDsa` directory.

Note: After you edit the data types, you must restart the Impact Server.

Setting up mappings for XML files and strings

For each XML string or file that you want the DSA to read, you must add the mapping information to the `XmlFileTypes` file.

Add the following mapping information:

- Name of the super data type
- Path and file name of the corresponding XML DTD/XSD file
- Path and file name of the corresponding XML file (XML files only)
- Namespace prefix that is used for the element data types (optional)

You use the following format to specify mapping information:

```
XmlDsa.fileTypes.n.property=value
```

where *n* is a numeric value that identifies the mapping, *property* is the name of the mapping property, and *value* is the value.

Table 1 shows the mapping properties in the `XmlFileTypes` file.

Table 38. XmlFileTypes mapping properties

Property	Description
`typeName`	Specifies the name of the corresponding super data type.
`dtdFile`	Specifies the path and file name of a corresponding XML DTD or XSD file. The path can be an absolute path or a path relative to the `$IMPACT_HOME` directory.
`isXsd`	Boolean variable that specifies whether the schema is defined in XSD or DTD format. If it is not specified, the default is DTD format. If it is not specified, the default is DTD format.
`xmlFile`	Specifies the path and file name of the corresponding file for XML files. The path is relative to the `$IMPACT_HOME` directory. For XML strings, use the hyphen character as a placeholder.
`prefix`	Specifies the namespace prefix that is used to identify the corresponding element data types. This property is optional.

This example shows a set of mapping properties for an XML document that is contained in a file.

```
XmlDsa.fileTypes.1.typeName XML_file_superType
XmlDsa.fileTypes.1.dtdFile dsa/XmlDsa/file.dtd
XmlDsa.fileTypes.1.xmlFile dsa/XmlDsa/file.xml
XmlDsa.fileTypes.1.prefix XML_
```

This example shows a set of mapping properties for an XML document that is contained in a string.

```
XmlDsa.fileTypes.2.typeName XML_string_superType
XmlDsa.fileTypes.2.dtdFile dsa/XmlDsa/string.dtd
XmlDsa.fileTypes.2.xmlFile -
XmlDsa.fileTypes.2.prefix XML_
```

Note: this example uses the hyphen character (-) for the `xmlFile` property.

The following example shows an expression that uses the `XmlFileTOC` data type with `isXsd` set to `true`. The name space prefix is `FTEST`. This prefix must be added to all data types that are a part of the XML file.

```
XmlDsa.fileTypes.1.typeName XmlFileTOC
XmlDsa.fileTypes.1.dtdFile dsa/XmlDsa/TOC.xsd
XmlDsa.fileTypes.1.xmlFile dsa/XmlDsa/TOC.xml
XmlDsa.fileTypes.1.prefix FTEST_
XmlDsa.fileTypes.1.isXsd true
```

Setting up mappings for XML over HTTP

For each XML document that you want the DSA to read over HTTP, you must add the mapping information to the `XmlHttpTypes` file.

Add the following mapping information:

- Name of the super data type.
- Base URL for the HTTP server.
- User name, password, and authentication realm (optional). This information is only required if the XML document is in a password-protected area of the HTTP server.
- Namespace prefix that is used for the element data types (optional).

You use the following format to specify mapping:

```
XmlDsa.httpTypes.n.property=value
```

where *n* is a numeric value that identifies the mapping, *property* is the name of the mapping property, and *value* is the value.

Table 1 shows the mapping properties in the `XmlHttpTypes` file.

Table 39. XmlHttpTypes mapping properties

Property	**Description**
`typeName`	Name of the corresponding super data type.
`dtdFile`	Path and file name of the corresponding XML DTD file. Can be an absolute path, or relative to the `$IMPACT_HOME` directory.
`xsdFile`	Path and file name of a corresponding XML XSD file. Can be an absolute path, or relative to the `$IMPACT_HOME` directory. Used only if the XML schema is an XSD.

Table 39. XmlHttpTypes mapping properties (continued)

Property	Description
`isXsd`	This Boolean variable specifies whether the schema is defined in XSD or DTD format. Default is DTD, if not specified.
`url`	Base URL for the HTTP server. The base URL includes the server host name, and the path where the script or executable file that provides the XML data is located. You do not need to specify the trailing backslash in the base URL. This URL is combined with the contents of the `FilePath` parameter to form the complete URL when you retrieve the XML data in a policy.
`user`	User name valid under HTTP server authentication (optional).
`password`	Password valid under HTTP server authentication (optional).
`realm`	Authentication realm on the HTTP server (optional).
`prefix`	Namespace prefix that is used to identify the corresponding element data types (optional).
`connectionsPerHost`	Number of connections per host. The default is 2. (Optional)

This example shows a set of mapping properties for XML data that is provided by an HTTP server.

```
XmlDsa.httpTypes.1.typeName XML_http_superType
XmlDsa.httpTypes.1.dtdFile dsa/XmlDsa/http.dtd
XmlDsa.httpTypes.1.url http://localhost:9080/cgi-bin
XmlDsa.httpTypes.1.user jsmith
XmlDsa.httpTypes.1.password pwd
XmlDsa.httpTypes.1.realm primary
XmlDsa.httpTypes.1.connectionsPerHost 5
```

Reading XML documents

You can read XML documents from within a policy.

Procedure

1. Retrieve the document data item.

   You retrieve the data item by calling the `GetByFilter` function and passing the name of the super data type and a filter string.
2. Retrieve the root level element data item.

   To retrieve the root level element data item, use the `GetByLinks` function.
3. Retrieve the child element data item.

   To retrieve child element data items, you can use successive calls to the `GetByLinks` function or you can use the embedded linking syntax.
4. Access attribute values.

   To access an element data item's attribute values, reference the corresponding data type fields.

Retrieving the document data item

You retrieve the data item by calling the `GetByFilter` function and passing the name of the super data type and a filter string.

The content of the filter string varies depending on whether the data source is an XML string, XML file, or XML data that is located on an HTTP server.

For XML strings, the filter is the entire XML string that you want to read. For XML files, the filter is an empty string. For XML over HTTP, the filter string is an expression that specifies the method to use in retrieving the XML data and the path to a script or executable file that provides the data on the HTTP server. For more information, see "XML over HTTP."

This example shows how to retrieve the document data item that is associated with an XML string, where the corresponding super type is named `XML_string_SuperType`:

```
// Call GetByFilter and pass the name of the super type
// and the filter string

Type = "XML_string_superType";
Filter = "<alert><node>Node1234</node><summary>
Node not responding</summary></alert>";
CountOnly = False;
DocDataItem = GetByFilter(Type, Filter, CountOnly);
```

This example shows how to retrieve the document data item that is associated with an XML file, where the corresponding super type is named `XML_file_superType`:

```
// Call GetByFilter and pass the name of the super type// and the filter
stringType = "XML_file_superType";Filter = "";CountOnly = False;DocDataItem =
GetByFilter(Type, Filter, CountOnly);
```

XML over HTTP

For XML over HTTP, the filter string is an expression that specifies the method to use in retrieving the XML data and the path to a script or executable file that provides the data on the HTTP server.

The XML DSA uses either the `GET` or `POST` method to retrieve the XML data. For example::

```
Operation = 'method' AND FilePath = 'path'
```

Where `method` is either `GET` or `POST` and `path` is the location of the script or executable relative to the base URL. You specify the base URL when you set the mapping information for the document in the `XmlHttpTypes` file.

Note: The `FilePath` specification can include query string values. You can retrieve XML documents from the HTTP server that are dynamically created depending on values that are sent by Netcool/Impact as part of the HTTP request.

This example shows how to use an HTTP `GET` request to retrieve the document data item that is associated with XML data. In this example, the name of the super data type is `XML_http_superType` and the location of the script that provides the XML data is `getXMLdoc.pl`.

```
// Call GetByFilter and pass the name of the super type// and the filter
stringType = "XML_http_superType";Filter = "Operation = 'GET' AND
FilePath = 'getXMLdoc.pl?node=NodeXYZ'";CountOnly = False;DocDataItem =
GetByFilter(Type, Filter, CountOnly);
```

Retrieving the root level element data item

To retrieve the root level element data item, use the `GetByLinks` function.

When you call `GetByLinks`, you must pass the name of the root level element data type, an empty filter string, and the document data item.

This example shows how to use GetByLinks to retrieve the root level element data item.

```
// Call GetByLinks and pass the name of theDataTypes = {"XML_alert"};Filter = "";
MaxNum = "10000";DataItems = DocDataItem;RootDataItem =
GetByLinks(DataTypes, Filter, MaxNum, DataItems);
```

Retrieving child element data items

To retrieve child element data items, you can use successive calls to the GetByLinks function or you can use the embedded linking syntax.

This example shows how to use the linking syntax to retrieve the first child element data item that is linked to the root level element data item, where the data type of the child data item is XML_body.

```
ChildNode = RootDataItem[0].links.XML_body.first;
```

This example shows how to retrieve an array that contains all child element data items that are linked to the root level element data item.

```
ChildNodes = RootDataItem[0].links.XML_body.array;
```

Accessing attribute values

To access an element data item's attribute values, reference the corresponding data type fields.

This example shows how to log the value of the ID attribute that is associated with the current element data item:

```
Log("The message ID is: " + DataItem.id);
```

This example shows how to log the PCDATA value that is associated with the current element data item:

```
Log(DataItem.PCDATA);
```

Sample policies

The DSA provides four sample policies.

- XmlStringTestPolicy
- XmlFileTestPolicy
- XmlHttpTestPolicy
- XmlXsdFileTestPolicy

These policies are configured to use the TOC.dtd, TOC.xsd and TOC.xml files in the $IMPACT_HOME/impact/dsa/XmlDsa directory.

XmlStringTestPolicy

The XmlStringTestPolicy shows how to use the XML DSA to read data from an XML string.

The policy reads the contents of an XML-formatted string and then prints the data to the policy log. Before you use this policy, you must run the create DTD types script as follows:

```
./CreateDtdTypes.sh NCI admin netcool ../TOC.dtd XmlStringTOC STEST_
```

You do not need to edit the contents of the XmlFileTypes configuration file. By default, this file contains the necessary data source mappings. The data type mappings are defined as follows:

```
XmlDsa.fileTypes.2.typeName XMLStringTOC
XmlDsa.fileTypes.2.dtdFile dsa/XmlDsa/TOC.dtd
XmlDsa.fileTypes.2.xmlFile -
XmlDsa.fileTypes.2.prefix STEST_
```

XmlFileTestPolicy

The XmlFileTestPolicy shows how to use the XML DSA to read data from an XML file.

This policy reads the contents of the TOC.xml file and then prints the data to the policy log. Before you use this policy, you must run the create DTD types script as follows:

```
./CreateDtdTypes.sh NCI admin netcool ../TOC.dtd XmlFileTOC FTEST_
```

You do not need to edit the contents of the XmlFileTypes configuration file. By default, this file contains the necessary data source mappings. The following are the data type mappings:

```
XmlDsa.fileTypes.1.typeName XmlFileTOC
XmlDsa.fileTypes.1.dtdFile dsa/XmlDsa/TOC.dtd
XmlDsa.fileTypes.1.xmlFile dsa/XmlDsa/TOC.xml
XmlDsa.fileTypes.1.prefix FTEST_
```

XmlHttpTestPolicy

The XmlHttpTestPolicy shows how to use the XML DSA to read data from a location on an HTTP server.

This policy reads the XML data from an HTTP server and then prints it to the policy log. Before you use this policy, you must run the CreateDtdTypes script as follows:

```
./CreateDtdTypes.sh NCI admin netcool ../TOC.dtd XmlHttpTOC HTEST_
```

You must also install a Perl CGI script on the HTTP server that generates the XML output that is requested by the DSA. This script is named xml.cgi and is located in $IMPACT_HOME/dsa/XmlDsa. You must check to make sure that the first line of the script references the actual location of the Perl executable file on the system before you install the script. To install, copy the script into the cgi-bin directory that is used by the HTTP server.

After you install the script, modify the XmlHttpTypes configuration file to reflect the location of the script and to include a valid user name and password for the authentication realm, if any.

The following example shows the data type mappings:

```
XmlDsa.httpTypes.1.typeName XmlHttpTOC
XmlDsa.httpTypes.1.dtdFile dsa/XmlDsa/TOC.dtd
XmlDsa.httpTypes.1.prefix HTEST_
XmlDsa.httpTypes.1.url http://localhost:9080
XmlDsa.httpTypes.1.user John
XmlDsa.httpTypes.1.password Smith
XmlDsa.httpTypes.1.realm basicrealm
```

XmlXsdFileTestPolicy

The XmlXsdFileTestPolicy shows how to use the XML DSA to read data from an XML file.

This policy reads XML data returned from a URL and then prints the data to the policy log. Before you use this policy, you must run the create XSD types script as follows:

```
./CreateXsdTypes.sh NCI admin netcool ../TOC.xsd XmlXsdFileTOC XSDFTEST_
```

where `filename` is the name and path of an XML file stored on the file system.

You do not need to edit the contents of the `XmlFileTypes` configuration file. By default, this file contains the necessary data source mappings. The following are the data type mappings:

```
XmlDsa.fileTypes.2.typeName XmlXsdFileTOC XmlDsa.fileTypes.2.xsdFile
dsa/XmlDsa/TOC.xsdXmlDsa.fileTypes.2.xmlFile
dsa/XmlDsa/TOC.xmlXmlDsa.fileTypes.2.prefix
XSDFTEST_XmlDsa.fileTypes.2.isXsd    true
```

Chapter 10. Working with IPL to XML functions

Data source adapters allow for access to data from a wide variety of sources. In many cases, the data is retrieved from SQL data sources and delivered as Impact Policy Language (IPL) objects (contexts). One challenge for policy writers has been converting these IPL contexts into XML strings for applications with interfaces that expect data in XML format. To facilitate this task, a set of IPL to XML functions has been developed that can be used to generate an XML string from an IPL object.

IPL to XML functions overview

You use IPL to XML functions to generate an XML string from an IPL object. What effectively happens is a top-level IPL object, referred to as the XML document object, is transformed into XML. IPL objects nested within the document object, referred to as element objects, become XML elements. The functions are used to create the XML document and element objects and to set XML attributes, content, and comments.

XML document object

The XML document object is the base object. XML element objects, attributes, content, and comments are added to the XML document object. It is the XML document object that is converted into an XML string.

For information about how to create the XML document object, see "Creating the XML document object" on page 94.

XML element objects

XML element objects are added to the XML document object or to each other (for nested XML elements). After you created and added an XML element object you can use other functions to add XML attributes, content, and comments to it. For more information about the IPL to XML functions used to create and add XML element objects, see "Adding a sub element" on page 94 and "Creating an unassociated element" on page 95.

Adding XML attributes to element objects

You can use on of three methods to add attributes to XML element objects. The simplest method is to pass an element object, attribute name, and attribute value to the `IPLtoXML.addAttribute()` function. For more information about this method, see "Adding XML attributes to element objects, simple approach" on page 95.

The second method requires creating a separate XML attribute object and adding that XML attribute object to the XML element object using the `IPLtoXML.addAttributeObject()` function. For more information about how to use this function, see "Adding XML attributes to element objects that use Attribute objects" on page 96. This method is useful in cases where you expect to use the same attribute and value in many places in your XML.

You use the third method to add several attributes at once using the `IPLtoXML.addOrgNodeAttributes()` function. Use this function to quickly take data from a data type lookup and add all the fields to an XML element object as attributes. For more information about this method, see "Adding XML attributes to element objects adding attributes from an OrgNode" on page 97.

Adding XML content to element objects

You can add content to any element and append additional content to it later. For more information about how to add and append content to element objects, see "Adding the content to an XML element object" on page 97 and "Appending content to XML element objects" on page 98.

XML comments

Using the `addCommentToElement` function you can add comments to any XML element. The function also puts in the XML commenting code (`<!-- -->`) for you so do not have to add it manually.

For more information about using this function, see "Adding XML comments to element objects" on page 98.

Nesting XML elements

In cases where you created a stand-alone XML element object using the `newElement` function you can still nest that XML element object using the `addElement` function. In most cases you, will not be creating stand-alone objects using the `newElement` function but instead will be creating and nesting XML element objects at the same time using the `newSubElement` function.

For more information about adding XML element objects to each other, see "Adding XML element objects to each other (nesting)" on page 99.

Creating the XML document object

Procedure

Use this function to create the XML document object:

```
IPLtoXML.newDocument(myXMLDocumentObject)
```

The `myXMLDocumentObject` variable becomes the new XML document object.

Example

```
newDocument(REM_Album);
```

Adding a sub element

This function creates an XML element object and nests it within the parent element object in one step.

Procedure

To add a sub element, use this function:

```
IPLtoXML.newSubElement(parentElement, myElement, elementType)
```

where

parentElement
This element or document object must exist.

myElement
This variable becomes the new XML element of type `elementType` nested within the `parentElement`.

elementType
The type of XML element to create.

Example

IPL:

```
newDocument(myCars);
newSubElement(myCars, myElement, "Honda");
newSubElement(myElement, driver1, "susi");
```

Generated XML:

```
<Honda><susi/></Honda>
```

Creating an unassociated element

Rather than using this function to create an unassociated element you can use the `newSubElement` function to create and nest an XML element object in one step.

Procedure

To create an unassociated element, use this function:

```
IPLtoXML.newElement(myElement, elementType)
```

where

myElement
This variable becomes the new XML element of type `elementType`.

elementType
The type of XML element to create.

Example

IPL:

```
newElement(myElement, "Honda");
```

Generated XML:

```
<Honda/>
```

Adding XML attributes to element objects, simple approach

Procedure

To add XML attributes to an element object, use this function:

```
IPLtoXML.addAttribute(myElementObject, attributeName, attributeValue)
```

where

myElementObject
The element to add the attribute to.

attributeName
The string name of the attribute to add.

attributeValue
The value of the attribute.

Example

IPL:

```
newElement(myForester, "Subaru");
addAttribute(myForester, "year", 2003);
```

Generated XML:

```
<Subaru year="2003"/>
```

Note: Use `addAttribute` when you want to avoid creating the attribute object.

Adding XML attributes to element objects that use Attribute objects

To add XML attributes to an element object that Attribute objects follow this procedure.

Procedure

1. Create the attribute object. Use the following function:

   ```
   IPLtoXML.newAttributeObject(attributeObject, attributeName, attributeValue)
   ```

   where

   attributeObject
   This variable becomes the new XML attribute object.

   attributeName
   The name of the attribute.

   attributeValue
   The value for the attribute.

   IPL:

   ```
   newAttributeObject(carYear, "year", 2003);
   ```

   Generated XML:

   ```
   year="2003"
   ```

   Note: Attribute objects are useful because they can then be reused and added to multiple elements in your code.
2. Add the attribute object to an element object. Use the following function:

   ```
   IPLtoXML.addAttributeObject(myElementObject, attributeObject)
   ```

   where

   myElementObject
   The XML element object to which the attribute is added

   attributeObject
   The XML attribute object that is added to `myElementObject`.

   IPL:

   ```
   newElement(myForester, "Subaru");
   newAttribute(carYear, "year", 2003);
   addAttributeObject(myForester, carYear);
   ```

Generated XML:

```
<Subaru year="2003"/>
```

Adding XML attributes to element objects adding attributes from an OrgNode

Procedure

To add attributes from an OrgNode, use this function:

```
IPLtoXML.addOrgNodeAttributes(elementObject, OrgNode)
```

where

elementObject
The element object to add the attributes to.

OrgNode
An IPL object whose fields you want to add to element as attributes. The object could have come from a lookup or could have been built using `newobject();`.

Example

IPL:

```
newElement(myForester, "Subaru");
og=newobject();
og.color="blue";
og.doors=4;
og.turbo="no";
addOrgNodeAttributes(myForester, og);
```

Generated XML:

```
<Subaru color="blue" doors="4" turbo="no"/>
```

Adding the content to an XML element object

Content can be added to any element.

Procedure

To add the content to an XML element object, use this function:

```
IPLtoXML.setContent(myElementObject, content)
```

where

myElementObject
The XML element object to add the content to.

content
The text string to add to `myElementObject` as content.

Example

IPL:

```
newElement(myForester, "Subaru");
setContent(myForester, "Extra Car");
```

Generated XML:

```
<Subaru>Extra Car</Subaru>
```

Appending content to XML element objects

Additional content can be appended to any element.

Procedure

To append the content to an XML element object, use this function:

```
IPLtoXML.appendContent(myElementObject, content)
```

where

myElementObject
: The XML element object to append the content to.

content
: The text string to append to the existing content.

Example

IPL:

```
newElement(myForester, "Subaru");
setContent(myForester, "Extra Car");
appendContent(myForester, " driven by Dad");
```

Generated XML:

```
<Subaru>Extra Car driven by Dad</Subaru>
```

Adding XML comments to element objects

Procedure

To add XML comments to element objects, use this function:

```
IPLtoXML.addCommentToElement(myElementObject, comment)
```

where

myElementObject
: The XML element object to add the XML comment to.

comment
: The text string that becomes the XML comment. XML remarking code `<!-- -- >` is added by the function. Do not add it yourself.

Example

IPL:

```
newElement(myForester, "Subaru");
addCommentToElement(myForester, "mom drives the element");
```

Generated XML:

```
<Subaru><!-- mom drives the element --></Subaru>
```

Adding XML element objects to each other (nesting)

Procedure

To nest an XML element object use this function:

```
IPLtoXML.addElement(dore, myElementObject)
```

where

dore The XML document or element object to add `myElementObject` to.

myElementObject
The XML element object that is added to the `dore`.

Example

IPL:

```
newElement(myCars, "Cars");
newElement(myForester, "Subaru");
addElement(myCars, myForester);
```

Generated XML:

```
<Cars><Subaru/></Cars>
```

Generating XML strings from document objects

Procedure

To generate an XML string from the document object, use this function:

```
IPLtoXML.generateXML(xmlPiece, XML)
```

where

xmlPiece
Either an entire XML document object or just one XML element object.

XML The variable to hold the XML string that is generated from `xmlPiece`.

Example

IPL:

```
STACK=NewJavaObject("java.util.Stack", {}); //
this global variable is required for the IPLtoXML conversion
newDocument(carXML);
newElement(myForester, "Subaru");
addElement(carXML, myForester);
addCommentToElement(myForester, "mom drives the element");
generateXML(carXML, Output);
```

This XML is generated in the Output variable:

```
<?xml version="1.0" encoding="UTF- 8"?><Subaru><!-- mom drives
the element--></Subaru>
```

Replacement of default XML entities

The function `replaceEntities()` replaces the following default XML entities in XML content and attributes:

- `&` ampersand, replaced with `&amp;`
- < less than, replaced with `&lt;`
- > greater than, replaced with `&gt;`
- `'` apostrophe, replaced with `&apos;`
- `"` quotation mark, replaces with `&quot;`

If you have additional entities that need to be replaced then edit the function `replaceEntities(x)` within the IPL to XML policy.

Element ordering in XML

The order in which elements at any particular nesting depth are added to the generated XML is based on the element object variable name.

Let us take the following policy for example:

```
IPLtoXML.newDocument(theWeather);
IPLtoXML.newSubElement(theWeather, system, "weatherSystem");
IPLtoXML.newSubElement(system, sub01, "tornado");
IPLtoXML.newSubElement(system, sub05, "thunderstorm");
IPLtoXML.newSubElement(system, SUB, "hail");
IPLtoXML.addComment(sub05, "bad thunderstorms and a tornado
where I live today");
```

The resulting XML looks like this example:

```
<weatherSystem><hail/><tornado/><thunderstorm><!-- bad
thunderstorms and a tornado where I live today
--></thunderstorm></weatherSystem>
```

The hail, tornado, and thunderstorm elements are all at the same depth in the XML nesting. The hail element was added first, element object name `SUB`. The tornado element was added second, element object name `sub01`. The thunderstorm element was added last `sub05`. If the element order in the XML is important then name the element objects carefully.

Examples of IPLtoXML functions usage

This section contains three examples of usage of IPLtoXML functions.

A simple example

This example shows how the different `IPLtoXML` functions can be used to generate a simple XML structure.

```
STACK=NewJavaObject("java.util.Stack", {});//required for
recursion in generateXML function
IPLtoXML.newDocument(docObj);
IPLtoXML.newElement(cars, "cars");
IPLtoXML.addElement(docObj, cars);
IPLtoXML.addAttribute(cars, "familyName", "Daniel");
IPLtoXML.setContent(cars, "the cars owned by a family");
IPLtoXML.newElement(subaru, "car");
IPLtoXML.newElement(honda, "car");
IPLtoXML.addElement(cars, subaru);
IPLtoXML.addElement(cars, honda);
IPLtoXML.newAttributeObject(carLocation, "location", "in the
garage");
IPLtoXML.addAttributeObject(subaru, carLocation);
IPLtoXML.addAttributeObject(honda, carLocation);
IPLtoXML.addAttribute(honda, "color", "black");
IPLtoXML.addAttribute(honda, "doors", 4);
IPLtoXML.addAttribute(honda, "driver", "susi");
IPLtoXML.addAttribute(honda, "model", "honda element");
foresterDetails=newobject();
foresterDetails.color="blue";
foresterDetails.doors="4";
foresterDetails.driver="tom";
foresterDetails.model="subaru forester";
IPLtoXML.addOrgNodeAttributes(subaru, foresterDetails);
IPLtoXML.newElement(hondaDriver, "Driver");
IPLtoXML.addAttribute(hondaDriver, "Name", "Susan Daniel");
IPLtoXML.addAttribute(hondaDriver, "Age", 33);
IPLtoXML.addCommentToElement(hondaDriver, "mother of
twins");
IPLtoXML.addElement(honda, hondaDriver);
IPLtoXML.generateXML(docObj, xml);
log(xml);
```

The results of logging the variable XML:

```
07 Oct 2007 17:11:06,888: SynchronousMessageProcessor:
Parser log: <?xml version="1.0" encoding="UTF-8"?><cars
familyName="Daniel">the cars owned by a family<car
color="blue" doors="4" driver="tom" location="in the garage"
model="subaru forester"/><car color="black" doors="4"
driver="susi" location="in the garage" model="honda
element"><Driver Age="33" Name="Susan Daniel"/><!-- mother
of twins -- ></car></cars>
```

ObjectServer event example

This example shows how `IPLtoXML` is run on an event from the Netcool/OMNIbus ObjectServer. Let us assume that we start with an Netcool/OMNIbusevent generated from the Netcool/OMNIbus Simnet probe.

1. Create an XML document object:
   ```
   IPLtoXML.newDocument(eventXML);
   ```
2. Create an XML element object and add it to the XML document object:
   ```
   IPLtoXML.newSubElement(eventXML, theEvent, "omniEvent");
   ```
3. Add the fields in the Event Container to the theEvent element object as attributes:
   ```
   IPLtoXML.addOrgNodeAttributes(theEvent, EventContainer);
   ```
4. Use the Java DSA to create a Java memory stack in a global variable called STACK:
   ```
   STACK=NewJavaObject("java.util.Stack", {});
   ```

   Note: `STACK` is required every time the `generateXML()` function is called and must be a global variable (defined outside of any function). `IPLtoXML` uses recursion and (v4.x) you cannot do recursive functions in the Impact parser without a separate memory stack.
5. Call the `GenerateXML()` function itself. Pass the function the XML document object, `eventXML`, and a results variable to put the output into:
   ```
   IPLtoXML.GenerateXML(eventXML, results);
   ```

The generated XML looks like this example:

```
<?xml version="1.0" encoding="UTF-8"?><omniEvent
Acknowledged="0" Agent="LinkMon" AlertGroup="Link"
AlertKey="" Class="3300" Customer="" EventId=""
EventReaderName="OMNIbusEventReader" ExpireTime="0"
FirstOccurrence="1188488450" Flash="0" Grade="0"
Identifier="link6LinkMon1Link" InternalLast="1188488841"
KeyField="8689" LastOccurrence="1188488841"
LocalNodeAlias="" LocalPriObj="" LocalRootObj=""
LocalSecObj="" Location="" Manager="Simnet Probe"
NmosCauseType="0" NmosObjInst="0" NmosSerial="" Node="link6"
NodeAlias="" OwnerGID="0" OwnerUID="65534" PhysicalCard=""
PhysicalPort="0" PhysicalSlot="0" Poll="0" ProcessReq="0"
ReceivedWhileImpactDown="1" RemoteNodeAlias=""
RemotePriObj="" RemoteRootObj="" RemoteSecObj=""
Serial="8689" ServerName="NCOMS" ServerSerial="8689"
Service="" Severity="0" StateChange="1188488841"
Summary="Link Up on port" SuppressEscl="0" Tally="8"
TaskList="0" Type="2" URL="" X733CorrNotif=""
X733EventType="0" X733ProbableCause="0" X733SpecificProb=""/
>
```

There is a single XML Element called `omniEvent`. Each of the fields in the original Event Container are XML attributes of `omniEvent`.

Example of generating WAAPI XML using IPL

This example shows how IPL to XML can be used to create an XML string that can be passed to the Netcool/Webtop API (WAAPI).

```
//create the XML Document object
IPLtoXML.newDocument(waapiXML);
//first element: the methodCall
IPLtoXML.newSubElement(waapiXML, myMethodCall,
"methodCall");
//a method element nested in the methodCall element
IPLtoXML.newSubElement(meMethodCall, method1, "method");
//attributes for method1
IPLtoXML.addAttribute(method1, "methodName",
"entity.createOrReplaceEntity");
//entitygroup element nested in method element
IPLtoXML.newSubElement(method1, entityGroup01,
"entitygroup");
//attributes for entitygroup
IPLtoXML.addAttribute(entityGroup01, "name",
"support");//this group was added to NGF
//entity element nested in entitygroup
IPLtoXML.newSubElement(entityGroup01, entity01, "entity");
//entity attributes
IPLtoXML.addAttribute(entity01, "name", "ncientity01");
IPLtoXML.addAttribute(entity01, "filter", "Severity > 4");
IPLtoXML.addAttribute(entity01, "metriclabel", "tom");
IPLtoXML.addAttribute(entity01, "metricshow", "Average");
IPLtoXML.addAttribute(entity01, "metricof", "Severity");
//entitylist nested in entitygroup
IPLtoXML.newSubElement(entityGroup01, entitylist01,
"entitylist");
//entitylist attributes
IPLtoXML.addAttribute(entitylist01, "name",
"ncientitylist01");
IPLtoXML.addAttribute(entitylist01, "list", "AllEvents");
IPLtoXML.addAttribute(entitylist01, "view", "basic");
IPLtoXML.addAttribute(entitylist01, "metriclabel", "tom");
```

The resulting XML string:

```
<?xml version="1.0" encoding="UTF-8"?><methodCall><method
methodName="entity.createOrReplaceEntity"><entitygroup
name="support "><entity filter="Severity &gt; 4"
metriclabel="tom" metricof="Severity" metricshow="Average"
name="ncientity01"/><entitylist list="AllEvents"
metriclabel="tom" name="ncientitylist01"
view="basic"/></entitygroup></method></methodCall>
```

This string can be passed to WAAPI via JRExec or Command/Response to create the Webtop entity.

Chapter 11. Working with the SNMP DSA

The SNMP DSA is a data source adaptor that is used set and retrieve management information stored by SNMP agents.

SNMP DSA overview

The SNMP DSA is a data source adaptor that is used to set and retrieve management information stored by SNMP agents. It is also used to send SNMP traps and notifications to SNMP managers.

The SNMP DSA is installed automatically when you install Tivoli Netcool/Impact. You must make sure that any MIB files that are to be used by the DSA are located in the `$IMPACT_HOME/impact/dsa/snmpdsa/mibs` directory when you start the Impact Server. For more information about installing MIB files, see “Installing MIB files” on page 107. You are not required to perform any additional installation or configuration steps.

You must perform the following tasks when you use the SNMP DSA:
- Create one data source for each SNMP agent that you want to access using the DSA, or create a single data source and use it to access all agents. For more information about working with SNMP data sources, see “Working with SNMP data sources” on page 108.
- Create data types that you will use to access variables and tables managed by SNMP agents. For more information about working with SNMP data types, see “Working with SNMP data types” on page 110.
- Write one or more policies that set or retrieve variables and tables managed by SNMP agents, or that send SNMP traps and notifications. For more information about SNMP policies, see “SNMP policies” on page 112.

SNMP data model

An SNMP data model is an abstract representation of SNMP data managed by agents in your environment.

SNMP data models have the following elements:
- SNMP data sources
- SNMP data types

SNMP data sources

SNMP data sources represent an agent in the environment.

The data source configuration specifies the host name and port where the agent is running, and the version of SNMP that it supports. For SNMP v3, the configuration also optionally specifies authentication properties.

You can either create one data source for each SNMP agent that you want to access using the DSA, or you can create a single data source and use it to access all agents. You can create and configure data sources using the GUI. After you create a data source, you can create one or more data types that represent the OIDs of variables managed by the corresponding agent.

SNMP data types

SNMP data types are Netcool/Impact data types that specify the structure and content of data associated with an agent.

The identity of the agent is determined by the data source that is associated with the data type. Each data type specifies one or more object IDs (OIDs) that reference variables managed by the agent.

The SNMP DSA supports the following categories of data types:
- Packed OID data types
- Table data types

Previous versions of this DSA supported another category of data type called discrete OID data types. This category was used to reference single variable OIDs. In this version of the DSA, you access single variables in the exact same way that you access the sets of variables represented by packed OID data types.

For more information about OIDs and SNMP variables, see the reference documentation for the agent you want to access using the SNMP DSA.

Packed OID data types

Packed OID data types are data types that reference the OIDs of one or more variables managed by a single agent. You use this category of data type when you want to access single variables or sets of related variables. When you create a packed OID data type, you specify the name of the associated data source, the OID for each variable and options that determine the behavior of the DSA when connecting to the agent.

For more information about creating packed OID data types, see “Working with SNMP data types” on page 110.

Table data types

Table data types are data types that reference the OIDs of one or more SNMP tables managed by a single agent. When you create a table data type, you specify the name of the associated data source, the OID for the table and options that determine the behavior of the DSA when connecting to the agent.

For more information about creating data types, see “Creating SNMP data types” on page 110.

SNMP DSA process

The SNMP DSA process has the following phases:
- Sending Data to Agents
- Retrieving Data from Agents
- Sending Traps and Notifications to Managers
- Handling Error Conditions
- Handling Timeouts

Sending data to agents

The DSA supports two functions in the Netcool/Impact policy language (IPL) that allow you to send data to an SNMP agent. These functions are the standard function `AddDataItem` and the SNMP function `SnmpSetAction`.

When Netcool/Impact encounters a call to one of these functions in a Netcool/Impact policy, it assembles an SNMP SET command using the information specified in the function parameters and passes this command to the DSA for processing. The DSA then sends the command to the agent.

If the SET command is successful, the agent sends a confirmation message to the DSA and Netcool/Impact continues processing the policy.

Retrieving data from agents

The DSA supports three functions that allow you to retrieve data from an agent. These functions are the standard function GetByFilter and the SNMP functions SnmpGetAction and SnmpGetNextAction.

When Netcool/Impact encounters a call to one of these functions in a Netcool/Impact policy, it assembles an SNMP GET or GETNEXT command using the information specified in the function parameters. It then passes this command to the DSA for processing. The DSA then sends the command to the agent.

If the GET or GETNEXT command is successful, the agent sends the requested data back to the DSA. The DSA returns the information to Netcool/Impact, which then stores the information in a policy-level variable that you can access in subsequent parts of the policy.

Sending traps and notifications to managers

The DSA supports an SNMP function named SNMPTrapAction that you use to send traps or notifications to an SNMP manager.

When the Netcool/Impact encounters a call to SNMPTrapAction, it assembles an SNMP TRAP command using the information specified in the function parameters. It then passes this command to the DSA for processing. The DSA then sends the command to the manager.

If the TRAP command is successful, the manager sends a confirmation message to the DSA and the policy is processed.

Handling error conditions

If a SET, GET, GETNEXT, or TRAP command sent to an agent or manager is unsuccessful, the DSA returns an error string to Netcool/Impact that can be printed to the policy log or otherwise handled in the body of the policy.

Handling timeouts

If an agent or manager does not respond to a SET, GET, GETNEXT, or TRAP command sent by the DSA within the timeout period specified in the function call or the related SNMP data type, the DSA sets a timeout message in the error string and returns it to Netcool/Impact. This error string can be handled in the body of the policy in the same was as any other error message.

Installing MIB files

You must make sure that any MIB files that are to be used by the DSA are located in the $IMPACT_HOME/impact/dsa/snmpdsa/mibs directory when you start the Netcool/Impact server. By default, this directory contains the RFC1213-MIB and RFC1271-MIB files. Other commonly used MIB files are installed with Netcool/Impact and are located in the $IMPACT_HOME/impact/dsa/snmpdsa/addMibs directory. You must copy these or other MIB files that you provide to the `$IMPACT_HOME/impact/dsa/snmpdsa/mibs` directory before you can use them with the DSA. After you copy a new file to this directory, you must stop and restart the Netcool/Impact server.

Working with SNMP data sources

You use the GUI to perform the following tasks with SNMP data sources:

- Create new data sources
- Edit data sources
- Delete data sources

Creating SNMP data sources

About this task

You can either create one data source for each SNMP agent that you want to access using the DSA, or you can create a single data source and use it to access all agents.

If you plan to use the standard data-handling functions `AddDataItem` and `GetByFilter` to access SNMP data, you must create a separate data source for each agent. In this scenario, the host name, port, and other connection information for the agent is encapsulated as part of the data source configuration. When you make a call to the `AddDataItem` or `GetByFilter` function, you pass the name of a data type associated with the data source and Netcool/Impact uses this information to derive the identity and location of the agent in the environment.

If you plan to use the SNMP functions that are provided with this release of the DSA, you can create a single data source and use it to access all agents. In this scenario, the host name and port are passed as runtime parameters when you call each function. You can dynamically specify the agent during policy runtime that is based on host name information from incoming ObjectServer events or derived from other external data sources.

This version of the DSA provides additional support for SNMP v3 authentication. If you are creating a data source for use with SNMP v3, you must perform additional configuration tasks.

Creating SNMP v1 and v2 data sources

Use this procedure to create an SNMP v1 or v2 data source.

Procedure

1. Log in to the Netcool/Impact GUI using a web browser.
2. Click the **Data Sources** tab and select **SNMP** from the **Source** list.
3. Click the **New Data Source** button.

   The New Data Source dialog box opens.
4. Type a unique name for the data source in the **Data Source Name** field.
5. If you are creating this data source for use with the standard data-handling functions `AddDataItem` and `GetByFilter`, type the host name or IP address where the agent resides in the **Host Name** field and the port in the **Port** field. If you are creating this data source for use with the new SNMP functions, you can accept the default values with no changes.

6. Type the name of the SNMP read-community in the **Read Community** field. The default is `public`.
7. Type the name of the SNMP write-community in the **Write Community** field. The default is `public`.
8. Type a timeout value in seconds in the **Timeout** field. When the DSA connects to an agent associated with this data source, it waits for the specified timeout period before returning an error to Netcool/Impact.
9. Select `1` or `2` from the **Version** list.
10. Click **OK**.

Creating SNMP v3 data sources

About this task

To create a data source with SNMP v3 authentication, you specify the configuration properties and then provide the information required for the agent to authenticate the DSA as an SNMP user. The authentication parameters can be overridden by calls to the SNMP functions in the Impact Policy Language.

For information about authentication parameters, see the documentation provided by the SNMP agent and manager.

To create an SNMP v3 data source:

Procedure

1. Log in to the GUI using a web browser.
2. Click the **Data Sources** tab and select **SNMP** from the **Source** list.
3. Click the **New Data Source** button.

   The New Data Source dialog box opens.
4. Type a data source name, the host name and IP address of the SNMP agent, community strings and timeout values as specified in the previous section.
5. Select `3` from the **Version** list.
6. Type the name of an SNMP v3 authentication user in the **User** field.
7. Select a protocol from the **Authentication Protocol** list. The default is `MD5`.
8. Type the password for the authentication user in the **Password** field.
9. Select a protocol from the **Privacy Protocol** field.
10. Type a privacy password in the **Privacy Password** field.
11. Type a context ID in the **Context ID** field.
12. Type a context name in the **Context Name** field.
13. Click **OK**.

Editing SNMP data sources

You can edit the configuration for a data source after you create it. To edit an SNMP data source:

Procedure

1. Log in to the GUI using a web browser.
2. Click the name of the data source in the **Data Sources** tab. The **Edit Data Source** window opens.
3. Set the configuration properties for the data source as described in the previous sections.

4. Click **OK**.

Results

Any changes to the configuration take effect immediately after you finish editing the data source. There is no need to restart the Impact Server after making a change.

Deleting an SNMP data source

About this task

To delete an SNMP data source:

Procedure

1. Log in to the Netcool/Impact GUI using a web browser.
2. In the **Data Sources** tab, click the **Delete Data Source** icon next to the name of the data source you want to delete.

Working with SNMP data types

You use the GUI to perform the following tasks with SNMP data types:

- Create new data types
- Edit data types
- Delete data types

Creating SNMP data types

About this task

If you plan to use the standard data-handling functions `AddDataItem` and `GetByFilter` to access SNMP data, you must create a separate data type for each set of variables (packed OID data types) or each set of tables (table data types) that you want to access. In this scenario, the object IDs (OIDs) for the variables or tables are encapsulated as part of the data type configuration. When you make a call to the `AddDataItem` or `GetByFilter` function, you pass the name of a data type and this information is used to determine the identity of the variables or table.

If you plan to use the SNMP functions that are provided with this release of the DSA, you can create a single data type for each data source and use it to access all the variables and tables associated with the agent. In this scenario, the variable or table OIDs are passed as runtime parameters when you call each function. You can dynamically specify the OIDs during policy runtime that is based on information from an external data source.

Creating packed OID data types

Packed OID data types are data types that reference the OIDs of one or more variables managed by a single agent. You use this category of data type when you want to access single variables or sets of related variables. When you create a packed OID data type, you specify the name of the associated data source, the OID for each variable and options that determine the behavior of the DSA when connecting to the agent.

To create a packed OID data type:

1. Log in to the Netcool/Impact GUI using a web browser.

2. Click the **Data Types** tab and select an SNMP data source from the **Data Source** list.
3. Click the **New Data Type** icon. The **New Data Type** editor opens.
4. Type a name for the data type in the **Data Type Name** field.
5. Select an SNMP data source from the **Data Source Name** field. By default, the data source you chose in step 2 is selected.
6. Select `Packed` from the **OID Configuration** list.
7. If you are creating this data type for use with the standard data-handling functions `AddDataItem` and `GetByFilter`, you must create an attribute on the data type for each variable you want to access. To create an attribute, click the **New Attribute** button and specify an attribute name and the OID for the variable.

   If you are creating this data source for use with the new SNMP functions, you do not need to explicitly create attributes for each variable. In this scenario, you pass the variable OIDs when you make each function call in the Netcool/Impact policy.
8. Click **Save**.

Creating table data types

Use this procedure to create a table data type.

Procedure

1. In the data types tab, select an SNMP data source from the list.
2. Click the **New Data Type** button to open the New Data Type editor.
3. Type a name for the data type in the **Data Type Name** field.

   Important:

   The data type name must match the table name that will be queried, for example, `ifTable`, or `ipRouteTable`.
4. Select an SNMP data source from the **Data Source Name** field. By default, the data source you chose in step 2 is selected.
5. Select `Table` from the **OID Configuration** list.
6. If you are creating this data type for use with the standard data-handling functions AddDataItem and GetByFilter, you must create a new attribute on the data type for each table you want to access. To create an attribute, click the **New Attribute** button and specify an attribute name and the OID for the table.

   Important:

   The attributes are the column names in each table. For example, in the following ifTable, the attributes will be `ifIndex`, `ifDescr` and other column names:

   ```
   Column Names          OID
   ifIndex             .1.3.6.1.2.1.2.2.1.1
   ifDescr             .1.3.6.1.2.1.2.2.1.2
   ...               ...
   ```

   If you are creating this data source for use with the new SNMP functions, you do not need to explicitly create attributes for each table. In this scenario, you pass the table OIDs when you make each function call in the Netcool/Impact policy.
7. If you want the DSA to retrieve table data from the agent using the SNMP `GETBULK` command instead of an SNMP `GET`, select **Get Bulk**.

The `GETBULK` command retrieves table data using a continuous `GETNEXT` command. This option is suitable for retrieving data from very large tables.

8. If you have selected **Get Bulk**, you can control the number of variables in the table for which the `GETNEXT` operation is performed using the specified **Non-Repeaters** and **Max Repetitions** values.

   The **Non-Repeaters** value specifies the first number of non-repeating variables and **Max Repetitions** specifies the number of repetitions for each of the remaining variables in the operation.
9. Click **Save**.

Editing SNMP data types

You can edit the configuration for a data type after you create it. To edit an SNMP data types:

Procedure

1. Log in to the GUI using a web browser.
2. Click the name of the data type in the **Data Types** tab.

   The Edit Data Type window opens.
3. Set the configuration properties for the data type as described in the previous sections.
4. Click **OK**.

Results

Any changes to the configuration take effect immediately after you finish editing the data type. There is no need to restart the Impact Server after making a change.

Deleting SNMP data types

About this task

To delete an SNMP data type:

Procedure

1. Log in to the Netcool/Impact GUI using a web browser.
2. In the **Data Types** tab, click the **Delete Data Type** button next to the name of the data type you want to delete.

SNMP policies

You can perform the following tasks related to the SNMP DSA in a policy:

- Set packed OID data on SNMP agents using standard data-handling functions
- Set packed OID data on SNMP agents using SNMP functions
- Set table data on SNMP agents using standard data-handling functions
- Set table data on SNMP agents using SNMP functions
- Retrieve packed OID data on SNMP agents using standard data-handling functions
- Retrieve packed OID data on SNMP agents using SNMP functions
- Send SNMP traps and notifications

Setting packed OID data with standard data-handling functions

About this task

You can use the standard data-handling function AddDataItem to set the value of a single variable managed by an agent or to set the value of multiple variables.

Setting the value of a single variable

To set the value of a single variable, you create a context, and populate its Oid and Value member variables. You can also populate optional HostId and Port members variables. After you populate the context variables, you call AddDataItem and pass the name of an SNMP data type and the context as runtime parameters. If you specified values for the HostId and Port variables in the context, these override the host and port information as defined in the data type.

To create a context, you call the NewObject function as shown in the following example.

```
// Call the NewObject function

MyContext = NewObject();
```

After you create the context, you can set the Oid and Value variables, as shown in the following example. All member variables of the context must be set as strings.

```
// Populate the context variables

MyContext.Oid = ".1.3.6.1.2.1.1.4.0";
MyContext.Value = "MyValue";
```

Oid and Value represent the OID of the variable managed by the agent and its corresponding value.

After you populate the context variables, you can call AddDataItem and pass the name of an SNMP data type and the context as runtime parameters, as shown in the following example.

```
// Call AddDataItem and pass the name of an SNMP data type and the context

AddDataItem("MySnmpType", MyContext);
```

In this example, the host name, and port where the agent is located is specified by the MySnmpType data type.

If the DSA is unable to successfully send the data to the agent, it stores an error message in the policy-level variable ErrorString. The following example shows how to print the error message to the policy log.

```
// Print any error message to the policy log

Log("Errors: " + ErrorString);
```

The following example shows how to set the value of a variable managed by an agent, where the host name and port are specified by the MySnmpType data type. In this example, the variable OID is .1.3.6.1.2.1.1.4.0 and the value is MyValue.

```
// Create a new context with the NewObject function

MyContext = NewObject();

// Populate the context variables
```

```
MyContext.Oid = ".1.3.6.1.2.1.1.4.0";
MyContext.Value = "MyValue";

// Call AddDataItem and pass the name of an SNMP data type and the context

AddDataItem("MySnmpType", MyContext);

// Print any error message to the policy log

Log("Errors: " + ErrorString);
```

The following example shows how to set the value of a variable managed by an agent, where the host name and port specified by the data type are overridden by context variables set in the policy. In this example, the host is `192.168.1.1` and the port is 161.

```
// Create a new context with the NewObject function

MyContext = NewObject();

// Populate the context variables

MyContext.Oid = ".1.3.6.1.2.1.1.4.0";
MyContext.Value = "MyValue";
MyContext.HostId = "192.168.1.1";
MyContext.Port = "161";

// Call AddDataItem and pass the name of an SNMP data type and the context

AddDataItem("MySnmpType", MyContext);

// Print any error message to the policy log

Log("Errors: " + ErrorString);
```

Setting the value of multiple variables

To set the value of multiple variables, you create a context and populate member variables that correspond to the attributes you configured when you created the corresponding SNMP data type. You can also populate optional `HostId` and `Port` members variables.

After you populate the context variables, you call `AddDataItem` and pass the name of the SNMP data type and the context as runtime parameters. If you specified values for the `HostId` and `Port` variables in the context, these override the host and port information as defined in the data type.

To create a context, you call the `NewObject` function as shown in the following example.

```
// Call the NewObject function

MyContext = NewObject();
```

After you create the context, you can set the member variables, and the optional variables, as shown in the following example. All member variables of the context must be set as strings.

```
// Populate the context variables

MyContext.SysLocation  = "New York";
MyContext.SysName = "SYS01";
```

Here, `SysLocation`, and `SysName` are attributes that you defined in the configuration for the corresponding SNMP DSA data source.

After you populate the context variables, you can call `AddDataItem` and pass the name of an SNMP data type and the context as runtime parameters, as shown in the following example.

```
// Call AddDataItem and pass the name of an SNMP data type and the context

AddDataItem("MySnmpType", MyContext);
```

In this example, the host name, and port where the agent is located is specified in the data type configuration.

If the DSA is unable to successfully send the data to the agent, it stores an error message in the policy-level variable `ErrorString`. The following example shows how to print the error message to the policy log.

```
// Print any error message to the policy log

Log("Errors: " + ErrorString);
```

The following example shows how to set the value of variables managed by an agent, where the host name and port is specified by the `MySnmpType` data type.

```
// Create a new context with the NewObject function

MyContext = NewObject();

// Populate the context variables

MyContext.SysLocation  = "New York";
MyContext.SysName = "SYS01";

// Call AddDataItem and pass the name of an SNMP data type and the context

AddDataItem("MySnmpType", MyContext);

// Print any error message to the policy log

Log("Errors: " + ErrorString);
```

The following example shows how to set the value of a variable managed by an agent, where the host name and port specified by the data type are overridden by context variables set in the policy. In this example, the host is `192.168.1.1` and the port is 161.

```
// Create a new context with the NewObject function

MyContext = NewObject();

// Populate the context variables

MyContext.SysLocation  = "New York";
MyContext.SysName = "SYS01";
MyContext.HostId = "192.168.1.1";
MyContext.Port = "161";

// Call AddDataItem and pass the name of an SNMP data type and the context

AddDataItem("MySnmpType", MyContext);

// Print any error message to the policy log

Log("Errors: " + ErrorString);
```

Setting packed OID data with SNMP functions

Procedure

You can use the SNMP function `SnmpSetAction` to set the value of a single or multiple variables managed by an agent.
When you call `SnmpSetAction`, you pass an SNMP data type, the host name and port of the agent, an array of OIDs, and the array of values that you want to set. If you are using SNMP v3, you can also specify the information required to authenticate as an SNMP user.
For more information about `SnmpSetAction`, see "SnmpSetAction" on page 128.

Example

The following example shows how to set SNMP variables by calling `SnmpSetAction` and passing the name of an SNMP data type, an array of OIDs, and an array of values as runtime parameters. In this example, the SNMP data type is named `SNMP_PACKED`.

```
// Call SnmpSetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP SET

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = {" .1.3.6.1.2.1.1.4.0", "  .1.3.6.1.2.1.1.5.0"};
ValueList = {"Value_01", "Value_02"};

SnmpSetAction(TypeName, HostId, Port, VarIdList, ValueList, NULL, NULL, \
    NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

For more examples, see "SnmpSetAction" on page 128.

Retrieving packed OID data from SNMP agents

About this task

Packed OID data types reference the OIDs of one or more variables managed by a single agent. You use this category of data type when you want to access single variables or sets of related variables.

You can retrieve packed OID data from SNMP agents using one of the following functions:

Procedure

- Standard data-handling functions
- SNMP functions

Retrieving packed OID data with standard data-Handling functions

You can use the standard data-handling function `GetByFilter` to retrieve packed OID data managed by an agent.

To retrieve the packed OID data, you call `GetByFilter` and specify the name of an SNMP data type as a runtime parameter. The data type configuration contains a list of OIDs for the variables whose value you want to retrieve and attribute names that you can use to reference the values. The data source associated with the data type specifies the host name and port where the agent is located.

The `GetByFilter` function returns an array of data items whose first element stores a context where the member variables represent values retrieved from the agent. You can reference the returned values using the attribute names that you defined when you created the data type.

If the DSA is unable to successfully retrieve the data, it stores an error message in a member variable on the context called `ErrorString`.

The following example shows how to call `GetByFilter` and specify the name of an SNMP data type. You can set the `Filter` parameter to an empty string and `CountOnly` to False.

```
// Call GetByFilter and pass the name of an SNMP data type

TypeName = "MySnmpType";
Filter = "";
CountOnly = False;

MySNMPValues = GetByFilter(TypeName, Filter, CountOnly);
```

The following example shows how to access values returned by the function. In this example, `MySnmpType` defines attributes named `HostId`, `SysContact`, `SysName`, and `SysLocation`.

```
// Access the member variables of the context returned by GetByFilter

Log("HostId: " + MySNMPValues[0].HostId);
Log("SysContact: " + MySNMPValues[0].SysContact);
Log("SysName: " + MySNMPValues[0].SysName);
Log("SysLocation: " + MySNMPValues[0].SysLocation);
```

The following example shows how to access an error message returned by the call to `GetByFilter`.

```
Log("Errors: " + MySNMPValues[0].ErrorString);
```

The following complete example shows how to use `GetByFilter` and handle the values it returns.

```
// Call GetByFilter and pass the name of an SNMP data type

TypeName = "MySnmpType";
Filter = "";
CountOnly = False;

MySNMPValues = GetByFilter(TypeName, Filter, CountOnly);

// Access the member variables of the context returned by GetByFilter

Log("HostId: " + MySNMPValues[0].HostId);
Log("SysContact: " + MySNMPValues[0].SysContact);
Log("SysName: " + MySNMPValues[0].SysName);
Log("SysLocation: " + MySNMPValues[0].SysLocation);

Log("Errors: " + MySNMPValues[0].ErrorString);
```

Retrieving packed OID data with SNMP functions

You can use the SNMP function `SnmpGetAction` to retrieve packed OID data managed by an agent.

When you call `SnmpGetAction`, you pass an SNMP data type, the host name and port of the agent, and other parameters. If you are using SNMP v3, you can also specify the information required to authenticate as an SNMP user.

For more information about SnmpGetAction, see "SnmpGetAction" on page 121.

The following example shows how to use SnmpGetAction. In this example, the variable OIDs are specified by the SNMP_PACKED data type configuration.

```
// Call SnmpGetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GET

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";

SnmpGetAction(TypeName, HostId, Port, NULL, NULL, NULL, NULL, NULL, NULL, NULL, \
    NULL, NULL, NULL);

// Print the results of the SNMP GET to the policy log

Count = 0;

While (Count < Length(ValueList)) {
    Log(ValueList[Index]);
    Count = Count + 1;
}
```

Traversing SNMP trees

You can use the SnmpGetNextAction function to retrieve the value of the next SNMP variables in the variable tree from an agent. This function is useful in situations where you want to traverse an entire tree or in situations where you do not know the OID of subsequent variables in a tree that you want to retrieve.

When you call SnmpGetNextAction, you pass an SNMP data type and the host name and port where the agent is located. If you are using SNMP v3, you can also specify the information required to authenticate as an SNMP user. You can also optionally pass a list of OIDs and other information needed to retrieve the data.

For more information about the SnmpGetNextAction function, see "SnmpGetNextAction" on page 125.

The following example shows how to use SnmpGetNextAction.

```
// Call SnmpGetNextAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GETNEXT

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";

SnmpGetNextAction(TypeName, HostId, Port, NULL, NULL, NULL, NULL, NULL, \
    NULL, NULL, NULL, NULL, NULL);

// Print the results of the SNMP GETNEXT to the policy log

Count = 0;

While (Count < Length(ValueList)) {
    Log(VarIdList + ": " + ValueList[Index]);
    Count = Count + 1;
}
```

Retrieving table data from SNMP agents

About this task

Table data types reference the OIDs of one or more tables managed by a single agent. You use this category of data type when you want to access SNMP tables.

You can retrieve table data from SNMP agents using:

Procedure

- Standard data-handling functions
- SNMP functions

Retrieving table data with standard data-handling functions

You can use the standard data-handling function `GetByFilter` to retrieve table data managed by an agent.

To retrieve the table data, you call `GetByFilter` and specify the name of an SNMP data type as a runtime parameter. The data type configuration contains a list of OIDs for the tables whose value you want to retrieve and attribute names that you can use to reference the tables. The data source associated with the data type specifies the host name and port where the agent is located.

The `GetByFilter` function returns an array of data items whose first element stores a context where the member variables represent values retrieved from the agent. You can reference the returned values using the attribute names that you defined when you created the data type.

If the DSA is unable to successfully retrieve the data, it stores an error message in a member variable on the context called `ErrorString`.

The following example shows how to call `GetByFilter` and specify the name of an SNMP data type. You can set the `Filter` parameter to an empty string and `CountOnly` to False.

```
// Call GetByFilter and pass the name of an SNMP data type

TypeName = "MySnmpType";
Filter = "";
CountOnly = False;

MySNMPValues = GetByFilter(TypeName, Filter, CountOnly);
```

The following example shows how to access values returned by the function. In this example, `MySnmpType` defines attributes named `HostId`, `SysContact`, `SysName`, and `SysLocation`.

```
// Access the member variables of the context returned by GetByFilter

Log("HostId: " + MySNMPValues[0].HostId);
Log("SysContact: " + MySNMPValues[0].SysContact);
Log("SysName: " + MySNMPValues[0].SysName);
Log("SysLocation: " + MySNMPValues[0].SysLocation);
```

The following example shows how to access an error message returned by the call to `GetByFilter`.

```
Log("Errors: " + MySNMPValues[0].ErrorString);
```

The following complete example shows how to use GetByFilter and handle the values it returns.

```
// Call GetByFilter and pass the name of an SNMP data type

TypeName = "MySnmpType";
Filter = "";
CountOnly = False;

MySNMPValues = GetByFilter(TypeName, Filter, CountOnly);

// Access the member variables of the context returned by GetByFilter

Log("HostId: " + MySNMPValues[0].HostId);
Log("SysContact: " + MySNMPValues[0].SysContact);
Log("SysName: " + MySNMPValues[0].SysName);
Log("SysLocation: " + MySNMPValues[0].SysLocation);

Log("Errors: " + MySNMPValues[0].ErrorString);
```

Sending SNMP traps and notifications

About this task

You use the SnmpTrapAction function to send a trap (for SNMP v1) or a notification (for SNMP v2) to an SNMP manager.

To send the trap or notification, you call the function and pass the host name and port where the manager is located, a list of OIDs and corresponding values for the trap, and other related information. If the trap or notification is not successful, the function stores an error message in the policy-level ErrorString variable. You can handle the contents of ErrorString in subsequent parts of the policy.

For more information about the SnmpTrapAction function, see “SnmpTrapAction” on page 131.

Example

The following example shows how to send a trap using the SnmpTrapAction function.

```
// Call SnmpTrapAction and pass the host name, port, OID list, OID values
// and other required parameters

HostId = "192.168.1.1";
Port = "162";
Version = "1";
Community = "public";

SysUpTime = "1001";

Enterprise = ".1.3.6.1.2.1.11";
GenericTrap = 3;
SpecificTrap = 0;

VarIdList = {".1.3.6.1.2.1.2.2.1.1.0", "sysDescr"};
ValueList = {"2", "My system"};

SnmpTrapAction(HostId, Port, VarIdList, ValueList, Community, Version, \
    SysUpTime, Enterprise, GenericTrap, SpecificTrap, NULL);

// Print any errors to the policy log

Log("Error: " + ErrorList);
```

The following example shows how to send a notification using the SnmpTrapAction function. In this example, you set a value for the SnmpTrapOid parameter.

```
// Call SnmpTrapAction and pass the host name, port, OID list, OID values
// and other required parameters

HostId = "192.168.1.1";
Port = "162";
Version = "1";
Community = "public";

SysUpTime = "1001";

Enterprise = ".1.3.6.1.2.1.11";
GenericTrap = 3;
SpecificTrap = 0;

VarIdList = {".1.3.6.1.2.1.2.2.1.1.0", "sysDescr"};
ValueList = {"2", "My system"};

SnmpTrapOid = ".1.3.6.1.2.4.1.11";

SnmpTrapAction(HostId, Port, VarIdList, ValueList, Community, Version, \
    SysUpTime, Enterprise, GenericTrap, SpecificTrap, SnmpTrapOid);

// Print any errors to the policy log

Log("Error: " + ErrorList);
```

SNMP functions

The SNMP DSA supports a special set of functions that you can use to send data to and retrieve data from SNMP agents. You can also use the SNMP functions to send SNMP traps and notifications to SNMP managers.

The SNMP DSA supports the following functions:

- SnmpGetAction
- SnmpGetNextAction
- SnmpSetAction
- SnmpTrapAction

The SNMP DSA also supports the use of standard data-handling functions as described in “SNMP policies” on page 112.

SnmpGetAction

The SnmpGetAction function retrieves a set of SNMP variables from the specified agent

The values are then stored in a variable named ValueList and any error messages in a variable named ErrorString. This function operates by sending an SNMP GET command to the specified agent.

When you call SnmpGetAction, you pass an SNMP data type and, for SNMP v3, any authorization parameters that are required. To override the agent and variable information specified in the SNMP data type, you can also optionally pass a host name, a port number, a list of OIDs, and other information needed to retrieve the data.

Syntax

The following is the syntax for `SnmpGetAction`:

```
SnmpGetAction(TypeName, [HostId], [Port], [VarIdList], [Community], [Timeout],
[Version], [UserId], [AuthProtocol], [AuthPassword], [PrivPassword], [ContextId],
[ContextName])
```

Parameters

The `SnmpGetAction` function has the following parameters.

Table 40. SnmpGetAction function parameters

Parameter	Format	Description
TypeName	String	Name of the SNMP data type that specifies the host name, port, OIDs, and other information needed to retrieve the SNMP data.
HostId	String	Optional. Host name or IP address of the system where the SNMP agent is running. Overrides value specified in the SNMP data type.
Port	Integer	Optional. Port where the SNMP agent is running. Overrides value specified in the SNMP data type.
VarIdList	Array	Optional. Array of strings containing the OIDs of SNMP variables to retrieve from the agent. Overrides values specified in the SNMP data type.
Community	String	Optional. Name of the SNMP write community string. Default is `public`.
Timeout	Integer	Optional. Number of seconds to wait for a response from the SNMP agent before timing out.
Version	Integer	Optional. SNMP version number. Possible values are `1`, `2` and `3`. Default is `1`.
UserId	String	Required for SNMP v3 authentication. If using SNMP v1 or v2, or using v3 without authentication, pass a `null` value for this parameter.
AuthProtocol	String	Optional. For use with SNMP v3 authentication only. Possible values are. `MD5_AUTH`, `NO_AUTH`, `SHA_AUTH`. `NO_AUTH` is the default.
AuthPassword	String	Optional. For use with SNMP v3 authentication only. Authentication password associated with the specified SNMP User ID.
PrivPassword	String	Optional. For use with SNMP v3 authentication only. Privacy password associated with the specified SNMP User ID.
ContextId	String	Optional. For use with SNMP v3 authentication only. Authentication context ID.
ContextName	String	Optional. For use with SNMP v3 authentication only. Authentication context name.

Return Values

When you call `SnmpGetAction`, it sets the following variables in the policy context: `ValueList` and `ErrorString`.

The ValueList variable is an array of strings, each of which stores the value of one variable retrieved from the SNMP agent. The strings in the array are assigned in the order that the variable OIDs are specified in the SNMP data type or the VarIdList parameter.

ErrorString is a string variable that contains any error messages generated while attempting to perform the SNMP GET command.

Example 1

The following example shows how to retrieve a set of SNMP variables by calling SnmpGetAction and passing the name of an SNMP data type as a runtime parameter. In this example, the SNMP data type is named SNMP_PACKED. The data type configuration specifies the host name and port where the SNMP agent is running and the OIDs of the variables you want to retrieve.

```
// Call SnmpGetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GET

TypeName = "SNMP_PACKED";

SnmpGetAction(TypeName, "192.168.1.1", 161, null, null, null, \
    null, null, null, null, null, null, null);

// Print the results of the SNMP GET to the policy log

Count = 0;

While (Count < Length(ValueList)) {
 Log(ValueList[Index]);
 Count = Count + 1;
}
```

Example 2

The following example shows how to retrieve a set of SNMP variables by calling SnmpGetAction and explicitly overriding the default host name, port, and other configuration values set in the SNMP data type.

Example 2 using IPL.

```
// Call SnmpGetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GET

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList =  {".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"};
Community = "private";
Timeout = 15;

SnmpGetAction(TypeName, HostId, Port, VarIdList, Community, \
    Timeout, null, null, null, null, null, null,null);

// Print the results of the SNMP GET to the policy log

Count = 0;

While (Count < Length(ValueList)) {
 Log(ValueList[Index]);
 Count = Count + 1;
}
```

Example 2 using JavaScript.

```
// Call SnmpGetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GET
TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = [".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"];
Community = "private";
Timeout = 15;
SnmpGetAction(TypeName, HostId, Port, VarIdList, Community, \
Timeout, null, null, null, null, null, null,null);
// Print the results of the SNMP GET to the policy log
Count = 0;
While (Count < Length(ValueList)) {
Log(ValueList[Index]);
Count = Count + 1;
}
```

Example 3

The following example shows how to retrieve a set of SNMP variables using SNMP v3 authentication.

Example 3 using IPL.

```
// Call SnmpGetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GET

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList =  {".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"};
Community = "private";
Timeout = 15;
Version = 3;
UserId = "snmpusr";
AuthProtocol = "MD5_AUTH";
AuthPassword = "snmppwd";
ContextId = "ctx";

SnmpGetAction(TypeName, HostId, Port, VarIdList, Community, \
    Timeout, Version, UserId, AuthProtocol, AuthPassword, null, ContextId, null);

// Print the results of the SNMP GET to the policy log

Count = 0;

While (Count < Length(ValueList)) {
 Log(ValueList[Index]);
 Count = Count + 1;
}
```

Example 3 using JavaScript.

```
// Call SnmpGetAction and pass the name of the SNMP data type that contains
// configuration information required to perform the SNMP GET
TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = [".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"];
Community = "private";
Timeout = 15;
Version = 3;
UserId = "snmpusr";
AuthProtocol = "MD5_AUTH";
AuthPassword = "snmppwd";
```

```
ContextId = "ctx";
SnmpGetAction(TypeName, HostId, Port, VarIdList, Community, \
Timeout, Version, UserId, AuthProtocol, AuthPassword, null, ContextId, null);
// Print the results of the SNMP GET to the policy log
Count = 0;
While (Count < Length(ValueList)) {
Log(ValueList[Index]);
Count = Count + 1;
}
```

SnmpGetNextAction

The `SnmpGetNextAction` function retrieves the next SNMP variables in the variable tree from the specified agent.

It stores the resulting OIDs in a variable named `VarIdList`, the resulting values in a variable named `ValueList`, and any error messages in a variable named `ErrorString`. The function sends a series of SNMP `GETNEXT` commands to the specified agent where each command specifies a single OID for which the next variable in the tree is to be retrieved.

When you call `SnmpGetNextAction`, you pass an SNMP data type and, for SNMP v3, any authorization parameters that are required. To override the agent and variable information specified in the SNMP data type, you can also optionally pass a host name, a port number, a list of OIDs, and other information needed to retrieve the data.

Syntax

The following is the syntax for `SnmpGetNextAction`:

```
SnmpGetNextAction(TypeName, [HostId], [Port], [VarIdList], [Community],
    [Timeout], [Version], [UserId], [AuthProtocol], [AuthPassword],
    [PrivPassword], [ContextId], [ContextName])
```

Parameters

The `SnmpGetNextAction` function has the following parameters.

Table 41. SnmpGetNextAction function parameters

Parameter	Format	Description
TypeName	String	Name of the SNMP data type that specifies the host name, port, OIDs, and other information needed to retrieve the SNMP data.
HostId	String	Optional. Host name or IP address of the system where the SNMP agent is running. Overrides value specified in the SNMP data type.
Port	Integer	Optional. Port where the SNMP agent is running. Overrides value specified in the SNMP data type.
VarIdList	Array	Optional. Array of strings containing the OIDs of SNMP variables to retrieve from the agent. Overrides values specified in the SNMP data type.
Community	String	Optional. Name of the SNMP write community string. Default is `public`.
Timeout	Integer	Optional. Number of seconds to wait for a response from the SNMP agent before timing out.

Table 41. SnmpGetNextAction function parameters (continued)

Parameter	Format	Description
Version	Integer	Optional. SNMP version number. Possible values are 1, 2 and 3. Default is 1.
UserId	String	Required for SNMP v3 authentication. If using SNMP v1 or v2, or v3 without authentication, pass a `null` value for this parameter.
AuthProtocol	String	Optional. For use with SNMP v3 authentication only. Possible values are. `MD5_AUTH`, `NO_AUTH`, `SHA_AUTH`. `NO_AUTH` is the default.
AuthPassword	String	Optional. For use with SNMP v3 authentication only. Authentication password associated with the specified SNMP User ID.
PrivPassword	String	Optional. For use with SNMP v3 authentication only. Privacy password associated with the specified SNMP User ID.
ContextId	String	Optional. For use with SNMP v3 authentication only. Authentication context ID.
ContextName	String	Optional. For use with SNMP v3 authentication only. Authentication context name.

Example 1

The following example shows how to retrieve SNMP variables in the variable tree by calling `SnmpGetNextAction` and passing the name of an SNMP data type as a runtime parameter. In this example, the SNMP data type is named `SNMP_PACKED`. The data type configuration specifies the host name and port where the SNMP agent is running and the OIDs of the variables whose subsequent values in the tree you want to retrieve.

```
// Call SnmpGetNextAction and pass the name of the SNMP
// data type that contains configuration information required
// to perform the SNMP GETNEXT

TypeName = "SNMP_PACKED";

SnmpGetNextAction(TypeName, "192.168.1.1", 161, null, null, \
    null, null, null, null, null, null, null, null);

// Print the results of the SNMP GETNEXT to the policy log

Count = 0;

While (Count < Length(ValueList)) {
 Log(VarIdList + ": " + ValueList[Index]);
 Count = Count + 1;
}
```

Example 2

The following example shows how to retrieve SNMP variables in the variable tree by calling `SnmpGetNextAction` and explicitly overriding the default host name, port, and other configuration values set in the SNMP data type.

Example 2 using IPL.

```
// Call SnmpGetNextAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP GETNEXT

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList =  {".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"};
Community = "private";
Timeout = 15;

SnmpGetNextAction(TypeName, HostId, Port, VarIdList, Community, \
    Timeout, null, null, null, null, null, null, null);

// Print the results of the SNMP GETNEXT to the policy log

Count = 0;

While (Count < Length(ValueList)) {
 Log(VarIdList + ": " + ValueList[Index]);
 Count = Count + 1;
}
```

Example 2 using JavaScript.

```
// Call SnmpGetNextAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP GETNEXT
TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = [".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"];
Community = "private";
Timeout = 15;
SnmpGetNextAction(TypeName, HostId, Port, VarIdList, Community, \
Timeout, null, null, null, null, null, null, null);
// Print the results of the SNMP GETNEXT to the policy log
Count = 0;
While (Count < Length(ValueList)) {
Log(VarIdList + ": " + ValueList[Index]);
Count = Count + 1;
}
```

Example 3

The following example shows how to retrieve subsequent SNMP variables in the variable tree using SNMP v3 authentication.

Example 3 using IPL.

```
// Call SnmpGetNextAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP GETNEXT

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList =  {".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"};
Community = "private";
Timeout = 15;
Version = 3;
UserId = "snmpusr";
AuthProtocol = "MD5_AUTH";
AuthPassword = "snmppwd";
ContextId = "ctx";

SnmpGetNextAction(TypeName, HostId, Port, VarIdList, Community, \
    Timeout, Version, UserId, AuthProtocol, AuthPassword, null, \
    ContextId, null);

// Print the results of the SNMP GET to the policy log

Count = 0;

While (Count < Length(ValueList)) {
 Log(VarIdList + ": " + ValueList[Index]);
 Count = Count + 1;
}
```

Example 3 using JavaScript.

```
// Call SnmpGetNextAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP GETNEXT
TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = [".1.3.6.1.2.1.1.5.0", ".1.3.6.1.2.1.1.6.0"];
Community = "private";
Timeout = 15;
Version = 3;
UserId = "snmpusr";
AuthProtocol = "MD5_AUTH";
AuthPassword = "snmppwd";
ContextId = "ctx";
SnmpGetNextAction(TypeName, HostId, Port, VarIdList, Community, \
Timeout, Version, UserId, AuthProtocol, AuthPassword, null, \
ContextId, null);
// Print the results of the SNMP GET to the policy log
Count = 0;
While (Count < Length(ValueList)) {
Log(VarIdList + ": " + ValueList[Index]);
Count = Count + 1;
}
```

SnmpSetAction

The SnmpSetAction function sets variable values on the specified SNMP agent.

If the attempt to set variable fails, it stores the resulting error message in a variable named `ErrorString`. This function operates by sending an SNMP `SET` command to the specified agent.

When you call `SnmpSetAction`, you pass an SNMP data type, the host name, and port of the agent, an array of OIDs, and the array of values that you want to set. If you are using SNMP v3, you can also include information required to authenticate as an SNMP user.

Syntax

The following is the syntax for `SnmpSetAction`:

```
SnmpSetAction(TypeName, [HostId], [Port], [VarIdList], \
ValueList, [Community], [Timeout], [Version], [UserId], [AuthProtocol],\
[AuthPassword], [PrivPassword], [ContextId], [ContextName])
```

Parameters

The `SnmpSetAction` function has the following parameters.

Table 42. SnmpSetAction function parameters

Parameter	Format	Description
TypeName	String	Name of the SNMP data type that specifies the host name, port, OIDs, and other information needed to set the SNMP data.
HostId	String	Optional. Host name or IP address of the system where the SNMP agent is running. Overrides value specified in the SNMP data type.
Port	Integer	Optional. Port where the SNMP agent is running. Overrides value specified in the SNMP data type.
VarIdList	Array	Array of strings containing the OIDs of SNMP variables to set on the agent. Overrides values specified in the SNMP data type.
ValueList	Array	Array of strings containing the values you want to set. You must specify these values in the same order that the OIDs appear either in the SNMP data type or in the *VarIdList* variable.
Community	String	Optional. Name of the SNMP write community string. Default is `public`.
Timeout	Integer	Optional. Number of seconds to wait for a response from the SNMP agent before timing out.
Version	Integer	Optional. SNMP version number. Possible values are `1`, `2` and `3`. Default is `1`.
UserId	String	Required for SNMP v3 authentication. If using SNMP v1 or v2, or using v3 without authentication, pass a `null` value for this parameter.
AuthProtocol	String	Optional. For use with SNMP v3 authentication only. Possible values are. `MD5_AUTH`, `NO_AUTH`, `SHA_AUTH`. `NO_AUTH` is the default.
AuthPassword	String	Optional. For use with SNMP v3 authentication only. Authentication password associated with the specified SNMP User ID.
PrivPassword	String	Optional. For use with SNMP v3 authentication only. Privacy password associated with the specified SNMP User ID.
ContextId	String	Optional. For use with SNMP v3 authentication only. Authentication context ID.
ContextName	String	Optional. For use with SNMP v3 authentication only. Authentication context name.

Example 1

The following example shows how to set SNMP variables by calling `SnmpSetAction` and passing the name of an SNMP data type, an array of OIDs, and an array of values as runtime parameters. In this example, the SNMP data type is named `SNMP_PACKED`.

Example 1 using IPL.

```
// Call SnmpSetAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP SET

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = {" .1.3.6.1.2.1.1.4.0", "  .1.3.6.1.2.1.1.5.0"};
ValueList = {"Value_01", "Value_02"};

SnmpSetAction(TypeName, HostId, Port, VarIdList, ValueList, \
    null, null, null, null, null, null, null, null, null);
```

Example 1 using JavaScript.

```
// Call SnmpSetAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP SET
TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = [" .1.3.6.1.2.1.1.4.0", " .1.3.6.1.2.1.1.5.0"];
ValueList = ["Value_01", "Value_02"];
SnmpSetAction(TypeName, HostId, Port, VarIdList, ValueList, \
null, null, null, null, null, null, null, null, null);
```

Example 2

The following example shows how to set SNMP variables using SNMP v3 authentication.

Example 2 using IPL.

```
// Call SnmpSetAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP SET

TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = { ".1.3.6.1.2.1.1.4.0",  " .1.3.6.1.2.1.1.5.0"};
ValueList = {"Value_01", "Value_02"};
Community = "private";
Timeout = 15;
Version = 3;
UserId = "snmpusr";
AuthProtocol = "MD5_AUTH";
AuthPassword = "snmppwd";
ContextId = "ctx";

SnmpSetAction(TypeName, HostId, Port, VarIdList, ValueList, \
    Community, Timeout, Version, UserId, AuthProtocol, \
    AuthPassword, null, ContextId, null);
```

Example 2 using JavaScript.

```
// Call SnmpSetAction and pass the name of the
// SNMP data type that contains configuration information
// required to perform the SNMP SET
TypeName = "SNMP_PACKED";
HostId = "192.168.1.1";
Port = "161";
VarIdList = [ ".1.3.6.1.2.1.1.4.0", " .1.3.6.1.2.1.1.5.0"];
ValueList = ["Value_01", "Value_02"];
Community = "private";
Timeout = 15;
Version = 3;
```

```
UserId = "snmpusr";
AuthProtocol = "MD5_AUTH";
AuthPassword = "snmppwd";
ContextId = "ctx";
SnmpSetAction(TypeName, HostId, Port, VarIdList, ValueList, \
Community, Timeout, Version, UserId, AuthProtocol, \
AuthPassword, null, ContextId, null);
```

SnmpTrapAction

The SnmpTrapAction function sends a trap (for SNMP v1) or a notification (for SNMP v2) to an SNMP manager.

Sending traps or notifications is not supported for SNMP v3.

Syntax

The following is the syntax for `SnmpTrapAction`:

```
SnmpTrapAction(HostId, Port, [VarIdList], [ValueList], \
[Community], [Timeout], [Version], [SysUpTime], [Enterprise], \
[GenericTrap], [SpecificTrap], [SnmpTrapOid])
```

Parameters

The `SnmpTrapAction` function has the following parameters.

Table 43. SnmpTrapAction function parameters

Parameter	Format	Description
HostId	String	Host name or IP address of the system where the SNMP manager is running.
Port	Integer	Port where the SNMP manager is running.
VarIdList	Array	Optional. Array of strings containing the OIDs of SNMP variables to send to the manager.
ValueList	Array	Optional. Array of strings containing the values you want to send to the manager. You must specify these values in the same order that the OIDs appear in the *VarIdList* variable.
Community	String	Optional. Name of the SNMP write community string. Default is `public`.
Timeout	Integer	Optional. Number of seconds to wait for a response from the SNMP agent before timing out.
Version	Integer	Optional. SNMP version number. Possible values are 1 and 2. Default is 1.
SysUpTime	Integer	Optional. Number of milliseconds since the system started. Default is the current system time in milliseconds.
Enterprise	String	Required for SNMP v1 only. Enterprise ID.
GenericTrap	String	Required for SNMP v1 only. Generic trap ID.
SpecificTrap	String	Required for SNMP v1 only. Specific trap ID.
SnmpTrapOid	String	Optional for SNMP v1. Required for SNMP v2. SNMP trap OID.

Example 1

The following example shows how to send an SNMP v1 trap to a manager using SnmpTrapAction.

Example 1 using IPL.

```
// Call SnmpTrapAction

HostId = "localhost";
Port = 162;
Version = 1;
Community = "public";
SysUpTime = 1001;
Enterprise = ".1.3.6.1.2.1.11";
GenericTrap = 3;
SpecificTrap = 0;
VarIdList = {".1.3.6.1.2.1.2.2.1.1.0", "sysDescr"};
ValueList = {"2", "My system"};

SnmpTrapAction(HostId, Port, VarIdList, ValueList, \
    Community, 15, Version, SysUpTime, Enterprise, GenericTrap, \
    SpecificTrap, null);
```

Example 1 using JavaScript.

```
// Call SnmpTrapAction
HostId = "localhost";
Port = 162;
Version = 1;
Community = "public";
SysUpTime = 1001;
Enterprise = ".1.3.6.1.2.1.11";
GenericTrap = 3;
SpecificTrap = 0;
VarIdList = [".1.3.6.1.2.1.2.2.1.1.0", "sysDescr"];
ValueList = ["2", "My system"];
SnmpTrapAction(HostId, Port, VarIdList, ValueList, \
Community, 15, Version, SysUpTime, Enterprise, GenericTrap, \
SpecificTrap, null);
```

Example 2

The following example shows how to send an SNMP v2 notification to a manager using SnmpTrapAction. SNMP v2 requires that you specify an SNMP trap OID when you call this function.

Example 2 using IPL.

```
// Call SnmpTrapAction

HostId = "localhost";
Port = 162;
Version = 1;
Community = "public";
SysUpTime = 1001;
Enterprise = ".1.3.6.1.2.1.11";
GenericTrap = 3;
SpecificTrap = 0;
VarIdList = {".1.3.6.1.2.1.2.2.1.1.0", "sysDescr"};
ValueList = {"2", "My system"};
SnmpTrapOid = ".1.3.6.1.2.4.1.11";

SnmpTrapAction(HostId, Port, VarIdList, ValueList, \
    Community, 15, Version, SysUpTime, Enterprise, \
    GenericTrap, SpecificTrap, SnmpTrapOid);
```

Example 2 using JavaScript.

```
// Call SnmpTrapAction
HostId = "localhost";
Port = 162;
Version = 1;
Community = "public";
SysUpTime = 1001;
Enterprise = ".1.3.6.1.2.1.11";
GenericTrap = 3;
SpecificTrap = 0;
VarIdList = [".1.3.6.1.2.1.2.2.1.1.0", "sysDescr"];
ValueList = ["2", "My system"];
SnmpTrapOid = ".1.3.6.1.2.4.1.11";
SnmpTrapAction(HostId, Port, VarIdList, ValueList, \
Community, 15, Version, SysUpTime, Enterprise, \
GenericTrap, SpecificTrap, SnmpTrapOid);
```

Chapter 12. Working with the ITNM DSA

The ITNM DSA is a Direct Mode, bi-directional DSA that is used to send queries to the Netcool/Impact ITNM application and get the results of those queries.

ITNM DSA overview

The ITNM DSA is a Direct Mode, bi-directional DSA that is used to send queries to the ITNM application and get the results of those queries.

After you set up Netcool/Impact and install the DSA, you can read the data in a policy using the `GetByFilter` function. The DSA can also receive asynchronous messages from ITNM regarding alerts.

The ITNM DSA requires ITNM version 3.8 or higher running on AIX®, Linux, Windows, or Solaris. For more information about ITNM hardware and software requirements, see the *Tivoli Network Manager IP Edition Version 3.8 and 3.9* Information Center at http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.tivoli.namnmip.doc/welcome_nmip.htm.

Setting up the DSA

The drivers required to connect Netcool/Impact to ITNM version 3.8 and 3.9 are available in `NCHOME/impact/integrations/itnm` in your Netcool/Impact installation.

- To connect to ITNM 3.8 use, `ncp_j_api-3.8.0.50.jar`.
- To connect to ITNM 3.9 use, `ncp_j_api-3.9.0.32.jar`.

For the version of ITNM you want to receive events from, complete the following steps:

1. Copy the appropriate jar file from `NCHOME/impact/integrations/itnm` and place it in `NCHOME/impact/dsalib` folder.
2. Restart the Netcool/Impact server.
3. If you are running in a clustered mode, repeat this step for each server in the cluster.

To set up the ITNM DSA, complete following tasks:

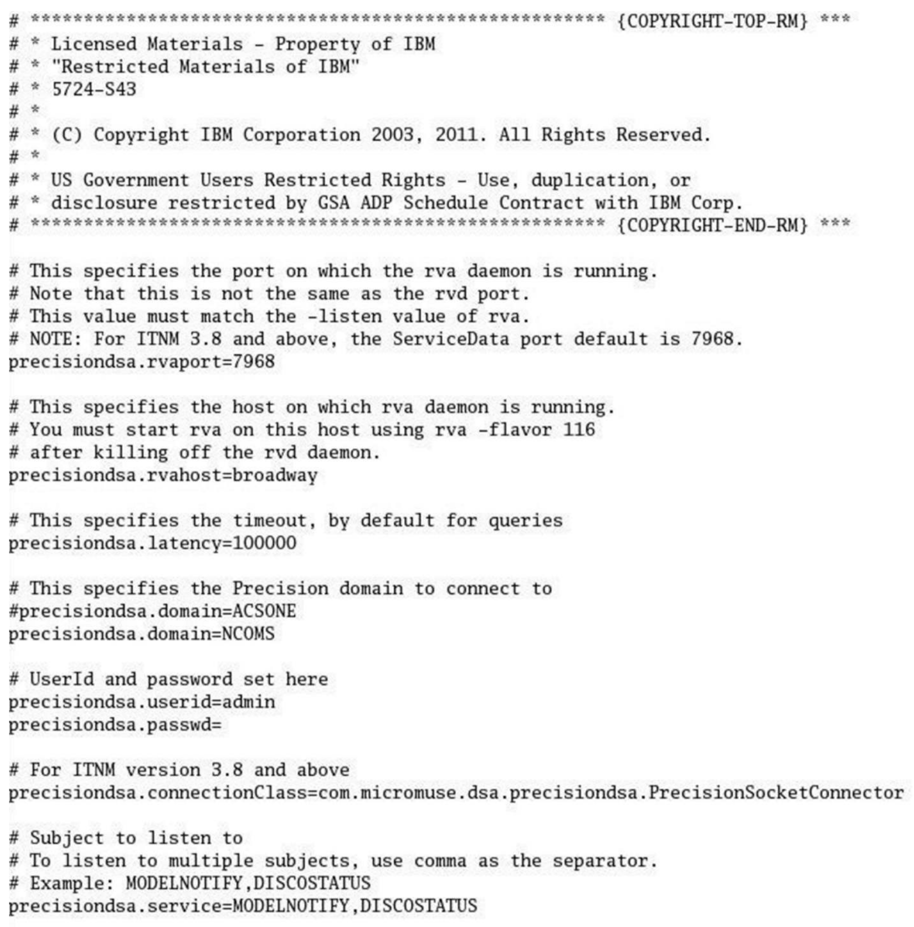
1. Edit the `precisiondsa.properties` file. For more information about this task, see "Editing the DSA properties file" on page 136.
2. Configure the **ITNM Event Listener**service for the DSA (optional). For more information about this task, see "Running the ITNM event listener service for the DSA" on page 136.
3. If you plan to receive asynchronous events from ITNM, start the ITNM Event Listener Service.

A preconfigured data type, data source, and two sample policies are included in Netcool/Impact.

Editing the DSA properties file

Procedure

1. After you set up the DSA and restarted the server, you must edit the `precisiondsa.properties` file, which you can find in the directory `$NCHOME/impact/dsa/precisiondsa`.
2. The following image shows an example of the ITNM DSA properties file. Edit the information as required to connect to the ITNM Listener Daemon, following the instructions in the file.



```
# ******************************************************** {COPYRIGHT-TOP-RM} ***
# * Licensed Materials - Property of IBM
# * "Restricted Materials of IBM"
# * 5724-S43
# *
# * (C) Copyright IBM Corporation 2003, 2011. All Rights Reserved.
# *
# * US Government Users Restricted Rights - Use, duplication, or
# * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# ******************************************************** {COPYRIGHT-END-RM} ***

# This specifies the port on which the rva daemon is running.
# Note that this is not the same as the rvd port.
# This value must match the -listen value of rva.
# NOTE: For ITNM 3.8 and above, the ServiceData port default is 7968.
precisiondsa.rvaport=7968

# This specifies the host on which rva daemon is running.
# You must start rva on this host using rva -flavor 116
# after killing off the rvd daemon.
precisiondsa.rvahost=broadway

# This specifies the timeout, by default for queries
precisiondsa.latency=100000

# This specifies the Precision domain to connect to
#precisiondsa.domain=ACSONE
precisiondsa.domain=NCOMS

# UserId and password set here
precisiondsa.userid=admin
precisiondsa.passwd=

# For ITNM version 3.8 and above
precisiondsa.connectionClass=com.micromuse.dsa.precisiondsa.PrecisionSocketConnector

# Subject to listen to
# To listen to multiple subjects, use comma as the separator.
# Example: MODELNOTIFY,DISCOSTATUS
precisiondsa.service=MODELNOTIFY,DISCOSTATUS
```

Running the ITNM event listener service for the DSA

The ITNM event listener service is preconfigured in Netcool/Impact. When the INTNM DSA is set up you can log in to Tivoli Netcool/Impact and, run the **ITNMEventListener** service available in the**Services** node for the **ITNM** project. This step is optional. It is only necessary to set up an event listener service if you want to listen for events asynchronously from IBM Tivoli Network Manager.

About this task

The **ITNMEvent Listener** service monitors a non-ObjectServer event source for events. They typically work with DSAs that allow bidirectional communication with a data source.

To run the **ITNMEvent Listener** service:

Procedure

1. From the **Project** selection list, select the **ITNM** project.
2. Select **Event Automation** > **Services**.
3. The **ITNMEvent Listener** service is displayed.
4. Enter the required information in new the Event Listener configuration window.
5. If you want to view the preconfigured settings, right click the service and click **Edit**.
   - **Listener Filter** Leave this field blank
   - **Policy To Execute** shows the `ITNMSampleListenerPolicy` that runs when an event is received from the IBM Tivoli Network Manager application.
   - **Direct Mode Class Name** this field is prepopulated

     ```
     con.micromuse.dsa.precisiondsa.
     PrecisionEventFeedSource
     ```
   - **Direct Mode Source Name** this field is prepopulated with a unique name that identifies the data source, for example, `ITNMServer`
6. Close the **ITNMEvent Listener** service tab.
7. To run the service, in the **Services** tab, select the **ITNMEvent Listener** service and click the **Start Service** icon to receive events from IBM Tivoli Network Manager. For information about IBM Tivoli Network Manager, see the IBM Tivoli Network Manager documentation available from the following link, http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.tivoli.namnmip.doc/welcome_nmip.htm.

ITNM DSA data type

The ITNM data type is the only one that works with the ITNM DSA.

You cannot rename an ITNM data type.

When the DSA queries the ITNM database, the records are returned as data items of the ITNM data type. Each field in the records is turned into an attribute of the corresponding data item.

For example, a record can contain fields such as:

- `ObjectId`
- `EntityName`
- `Address`
- `Description`
- `ExtraInfo`

To access the values, you can directly access the attributes just like any other data items using the following command:

```
log("Description is " + DataItem.Description);
```

This command prints out the `Description` field string that was on the ITNM record returned by the query.

ExtraInfo field

The `ExtraInfo` field is a special case.

All the other fields contain data which can be printed directly. The ExtraInfo field has a hierarchy and has subfields of its own, such as m_BaseName and DNSName. If you tried to print out the contents of the m_BaseName field as follows:

```
log("ExtraInfo->m_BaseName is " + DataItem.ExtraInfo.m_BaseName);
```

it will not work because DataItem.ExtraInfo is a string itself, not an DataItem that can be dereferenced. To get the value of m_BaseName, you must perform a more specific query as follows:

```
select ExtraInfo->m_BaseName,ExtraInfo->DNSName from master.entityByName;
```

This query selects the embedded fields from the ExtraInfo fields, and puts their values on the data item that is returned from the query.

Writing policies using the ITNM DSA

The ITNM DSA supports only the GetByFilter function. This function has three components for the filter argument for this DSA, as described in Table 44.

Table 44. ITNM DSA Filter Arguments

Argument	Description
Subject	This argument specifies on what service the OQL query has been sent to. For MODEL, the value is RIVERSOFT.3.0.MODEL.QUERY.
Query	This is the actual query to be sent to the subject described in the previous row. If this component exists, than all the records from the subject will be retrieved. Make sure that the OQL query contains **NO** " ' " characters.
Timeout	This is the timeout value for getting the results back. It uses the value in the precisiondsa properties file if you do not specify the timeout value in the filter.

GetByFilter

The GetByFilter function retrieves data items from a data type using a filter as the query condition.

To retrieve data items using a filter condition, you call GetByFilter and pass the data type name and the filter string as runtime parameters. The syntax for the filter string varies depending on whether the data type is an internal, SQL database, LDAP, or Mediator data type.

GetByFilter returns an array of references to the retrieved data items. If you do not assign the returned array to a variable, the function assigns it to the built-in DataItems variable and sets the value of the Num variable to the number of data items in the array.

You can use GetByFilter with internal, SQL database, and LDAP data types. You can also use GetByFilter with some Mediator data types.

Important: When data items are assigned to the built-in DataItem variable, they are not immediately updated but are stored in a queue to optimize the number of calls to the database. So, for example, if you update multiple fields in the DataItems variable there will only be one call to update the underlying database, when a function call is made. To force all queued updates, call the CommitChanges() function in your policy. The CommitChanges() function does not take any arguments.

Syntax

The `GetByFilter` function has the following syntax:

```
[Array =] GetByFilter(DataType, Filter, [CountOnly])
```

Parameters

The `GetByFilter` function has the following parameters.

Table 45. GetByFilter function parameters

Parameter	**Format**	**Description**
DataType	String	Name of the data type.
Filter	String	Filter expression that specifies which data items to retrieve from the data type.
CountOnly	Boolean	Pass a `false` value for this parameter. Provided for compatibility with earlier versions only.

Return value

Array of references to the retrieved data items. Optional.

Examples

The following example shows how to retrieve data items from an internal or SQL database data type.

```
// Call GetByFilter and pass the name of the data type
// and an SQL database filter expression

DataType = "Admin";
Filter = "Level = 'Supervisor' AND Location LIKE 'NYC.*'";
CountOnly = false;

MyAdmins = GetByFilter(DataType, Filter, CountOnly);
```

The following example shows how to retrieve data items from an LDAP data type.

```
// Call GetByFilter and pass the name of the data type
// and an LDAP filter expression

DataType = "Customer";
Filter = "(|(facility=NYC)(facility=NNJ))";
CountOnly = false;

MyCustomers = GetByFilter(DataType, Filter, CountOnly);
```

The following example shows how to retrieve data items from a Mediator data type.

```
// Call GetByFilter and pass the name of the data type
// and the Mediator filter exprssion

DataType = "SWNetworkElement";
Filter = "ne_name = 'DSX1 PNL-01 (ORP)'";
```

```
CountOnly = false;

MyElements = GetByFilter(DataType, Filter, CountOnly);
```

Writing policies to receive events from ITNM

The **ITNM Event Listener** Service that you optionally configured after installing the DSA is similar to the OMNIbusEventReader, with the exception that it can asynchronously receive events from ITNM.

Policy Variables

After an event is received, the policy assigned to it is invoked with the variables described in Table 46. The variables are stored in the `EventContainer` and must be referenced in the policy using the `EventContainer` or `@ notation`. See the `ITNMSampleListenerPolicy` for an example.

Table 46. Variables Returned by a Policy after Event Received from ITNM

Variable	Description
`ActionName`	This variable describes the type of action that is in the update. The possible values are: • `"REC_DELETE"` • `"REC_UPDATE"` • `"REC_NEW"` • `"DontKnow"`
`FieldNames`	This variable gives the names of the fields that are in the CRIV_Record that is received from ITNM. Since the field names returned in this record are not known before the policy is executed, a string concatenation of all these fieldNames, with a delimiter of "##", is used. This is a sample value in the *`FieldNames`* variable: `##Field1##Field2##Field3##field4` and so on.
`Field1`	One of the fields in the record returned by ITNM.
`Field2`	One of the fields in the record returned by ITNM.
`Field3`	One of the fields in the record returned by ITNM.
`Field4`	One of the fields in the record returned by ITNM.

Sample policies

The DSA provides the following sample policies:

- `ITNMSampleListenerPolicy`
- `ITNMSamplePolicy`

ITNMSampleListenerPolicy

`ITNMSampleListenerPolicy.ipl` shows how to use the ITNM DSA to read data from an ITNM Listener. The policy reads the contents of an ITNM formatted string and then prints the data to the Policy log.

ITNMSamplePolicy

`ITNMSamplePolicy.ipl` shows how to use the ITNM DSA to read data from an ITNM database. The policy reads the contents of an ITNM formatted string and then prints the data to the Policy log.

Chapter 13. Working with the socket DSA

[**Important:** This feature is deprecated.] The socket DSA is a data source adaptor that provides an interface between Tivoli Netcool/Impact and a socket server.

Socket DSA overview

The socket DSA is a data source adapter that provides an interface between Netcool/Impact and a socket server. You can use the socket DSA as a generic connector between Netcool/Impact and third-party entities where dedicated DSAs do not exist. These third-party entities can be any sort of device, application, or system that provides an interface accessible by a scripting or programming language that supports network sockets. Such languages include C/C++, Java, and Perl.

Socket server

A socket server is a program that acts as a mediator between a third-party entity and the socket DSA. You can implement a custom socket server or you can expand and use the sample socket server that is provided with the DSA. The socket server uses the Berkeley socket protocol to communicate with Netcool/Impact via a network. The socket server is a required part of a socket DSA solution. For more information about implementing a custom socket server, see “Implementing a custom socket server” on page 149.

Data model

The socket DSA data model consists of a data source and set of data types that you define. You must define one data type for each type of data that you plan to exchange between the socket DSA and the socket server. For more information, see "Socket DSA data model" on page 135.

Process

At run time, Netcool/Impact uses the socket DSA to send queries to the socket server for information that is stored or provided by a corresponding third-party entity. The socket server then makes a request for to the entity for the data. When the socket server receives a reply, it forwards the information back to the DSA. The DSA then populates the socket DSA data types with the data.

Setting up the socket DSA

The socket DSA is installed automatically when you install Netcool/Impact. You are not required to perform any additional installation or configuration steps.

Writing socket DSA policies

You use the standard Netcool/Impact function `AddDataItem` to send data to the socket server from within a policy. You use `GetByFilter`, `GetByKey`, and `GetByLinks` to retrieve data from the server.

© Copyright IBM Corp. 2006, 2014

Using the sample socket server

The socket DSA provides a sample socket server written in Perl that you can use to test and explore the function of the DSA. You can also customize the sample server and use it as part of your real world socket DSA solution.

Note: The best practice is to use the sample socket server to learn about the DSA function before you attempt to implement a custom server using any other development tools. The sample socket server is the best way to get started using the socket DSA.

For more information about working with the sample socket server, see "Working with the sample socket server" on page 146.

Implementing a custom socket server

If you do not want to use the sample socket server that is distributed with the DSA, you can implement your own using any scripting or programming language that provides access to network sockets. These languages include Java, C/C++, and Perl. For more information about implementing a custom socket server, see "Implementing a custom socket server" on page 149.

Socket DSA data model

The socket DSA data model consists of the following elements:

- Socket DSA data source
- Socket DSA data types

Socket DSA data source

[**Important:** This feature is deprecated.] The socket DSA data source is named SocketMediatorDataSource.

Netcool/Impact automatically generates a socket DSA named SocketMediatorDataSource at installation time. Configuration properties for this data source should never be changed.

Socket DSA data types

The socket DSA data model consists of a set of data types that you use to represent logical types of data passed between the DSA and the socket server.

The DSA does not provide a predefined set of data types. Instead, you must analyze the types of data that you plan to exchange between the DSA and the socket server and define one new data type for each required type of data.

For example, if you are using the socket server to mediate between Netcool/Impact and a network inventory system, you might define one data type for each network element whose information you want to access in a policy. These data types might be named `Location`, `Facility`, `Rack`, `Port` or `Card`. If you are using the socket server as an interface between Netcool/Impact and a messaging system, you might define one data type for each type of message that is to be passed between the DSA and the socket server. These data types might be named `ProblemRequest`, `ProblemReply`, `EnhancementRequest` and `EnhancementReply`.

When you customize the sample socket server or implement your own, you specify how the socket server handles requests from Netcool/Impact that are related to a particular data type.

Configuring the socket DSA

Procedure

To configure the socket DSA, you must manually set the properties in the DSA properties file.
The DSA properties file is named `socketdsa.properties` and is located in the `$IMPACT_HOME/impact/dsa/socketdsa` directory. Property values must not contain any trailing space characters.

Note: You must stop and restart the Netcool/Impact server after you change the properties file.
This table shows the properties in the DSA properties file.

Table 47. Socket DSA Configuration Properties

Property	Description
`SocketHost`	Host name of the system where the socket server is running.
`SocketPort`	Port number used by the socket server to listen to incoming requests from the socket DSA.
`verboseMode`	Specifies whether the DSA prints debug information to the Netcool/Impact server log.

Writing socket DSA policies

You can perform the following tasks with the socket DSA from within a Netcool/Impact policy:

- Retrieve data from the socket server by filter
- Retrieve data from the socket server by key
- Retrieve data from the socket server by links
- Send new data to the socket server

The results of these tasks are dictated in large part by how the socket server is implemented. To understand how a socket server handles these operations, you can review the `UserDataInterface.pm` file in the sample socket server. This file demonstrates how a simple socket server responds to requests to retrieve or add data.

A sample is automatically imported into the Netcool/Impact server during installation. This policy is named `TestSocketDsa` and demonstrates how to perform all of the function supported by the DSA. The sample policy works with the sample socket server that is included in the DSA tar file. For more information about the sample server, see “Working with the sample socket server” on page 146.

Retrieving data by filter

To retrieve data by filter from the socket server, you call the `GetByFilter` function and pass it the name of a socket DSA data type and the filter expression as runtime parameters.

The structure and content of the filter expression used in GetByFilter are specified when you customize or implement the socket server.

When Netcool/Impact encounters the call to GetByFilter in the policy, it passes the request to the socket DSA, which in turn passes the name of the data type and the full filter expression to the socket server. The socket server then analyzes the request and returns a nested array of sets of name/value pairs to the DSA that fulfill the terms of the specified filter. The DSA uses this nested array to populate the data items returned by the function in the policy.

The following example shows how to retrieve data by filter from the sample socket server distributed with the DSA. The code that handles requests to retrieve data by filter is located in the UserDataInterface.pm file.

```
Seconds = GetDate();log ("Starting TestSocketDSA with type JimmyExample at " + \
    LocalTime(Seconds, "HH:mm::ss"));Type="JimmyExample";Types={"JimmyExample"};
// GetByFilter testing
// First, no filter (should get all the items)
log ("Testing GetByFilter -- finding all");
Filter = "";
CountOnly = false;
All = GetByFilter(Type, Filter, CountOnly);
i = 0;
while (i < Num) {
    i = i + 1;
    log("Jimmy[" + i + "] => " + All[i-1].FirstName);
    // Save Jimmy's DataItem for later on when we test the Links.
    if (All[i-1].FirstName == "Jimmy") {
    JimmyDataItem = All[i-1];
    }
}
// Provide a filter this time.
log ("Testing GetByFilter -- finding Carl");
Filter = "FirstName = 'Carl'";
CountOnly = false;
Carl = GetByFilter(Type, Filter, CountOnly);
if (Num == 1) {
    log("Found Carl!  => " + Carl[0].FirstName + " " + Carl[0].LastName);
} else {
    log("Error: Didn't find Carl!");
}
log ("Testing GetByFilter -- finding Nick (bogus entry)");
Filter = "FirstName = 'Nick'";
CountOnly = false;
Nick = GetByFilter(Type, Filter, CountOnly);
if (Num == 1) {
    log("Yikes We found something!  => " + Nick[0]);
} else {
    log("Great!  We didn't find Nick!");
}
```

Retrieving data by key

To retrieve data by key from the socket server, you call the GetByKey function and pass the name of a socket DSA data type and a key expression as runtime parameters.

When Netcool/Impact encounters the call to GetByKey in the policy, it passes the request to the socket DSA, which in turn passes the name of the data type and the full key expression to the socket server. The socket server then analyzes the request and returns a set of name/value pairs to the DSA that fulfills the terms of the specified key expression. The DSA uses this nested array to populate the data items returned by the function in the policy.

The structure and content of the key expression used in GetByKey are specified when you customize or implement the socket server.

On the part of the socket server, retrieving data by key is different from retrieving data by filter in that each set of name/value pairs that it returns to the socket DSA can contain a name called KEY and a corresponding value. The KEY attribute is then used by the socket DSA to populate the key field in the corresponding Netcool/Impact data types. If there is no attribute named KEY, the socket DSA considers the first name/value pair returned to represent the key field for a data item.

The following example shows how to retrieve data by key from the sample socket server distributed with the DSA. The code that handles requests to retrieve data by key is located in the UserDataInterface.pm file.

```
Seconds = GetDate();log ("Starting TestSocketDSA with type JimmyExample at " + \
    LocalTime(Seconds, "HH:mm::ss"));Type="JimmyExample";Types={"JimmyExample"};
// Test GetByKey

log ("Testing GetByKey with existing key == Cindy");

Key = "Cindy";MaxNum = 1;
Cindy = GetByKey(Type, Key, MaxNum);

if (Cindy == NULL) {
    log("Error:  Didn't find Cindy!  Num is " + Num);} else {
    log("Found Cindy! => " + Cindy[0].FirstName + "  Num is " + Num);}log
 ("Testing GetByKey with nonexistent key == Judy");

Key = "Judy";Judy = GetByKey(Type, Key, MaxNum);

if (Num == 0) {
    log("Great!  Didn't find Judy!  Num is " + Num);} else {
    log("Yikes!  Found Judy! => " + Judy[0].FirstName + "  Num is " + Num);}
```

Retrieving data By links

To retrieve data by links from the socket server, you call GetByLinks and pass the name of a socket DSA data type, an optional link filter, the maximum number of data items to return and a data item that the returned data items are linked to.

When Netcool/Impact encounters the call to GetByLinks in the policy, it passes the request to the socket DSA. The socket DSA determines the key field in the linked data item and then passes that value along with the data type name, the link filter, and the maximum number of data items to the socket server. The socket server then analyzes the request and returns a nested array of sets of name/value pairs to the DSA that are linked to the specified data item and fulfill the terms of the specified key expression. The DSA uses this nested array to populate the data items returned by the function in the policy.

The structure and content of the link filter used in GetByLinks are determined when you customize or implement the socket server.

The following example shows how to retrieve data by links from the sample socket server distributed with the DSA. The code that handles requests to retrieve data by links is located in the UserDataInterface.pm file.

Note: Where a code line exceeds the width of page margins, the line is broken by a space and back slash: " \".

```
Seconds = GetDate();log ("Starting TestSocketDSA with type JimmyExample at " + \
    LocalTime(Seconds, "HH:mm::ss"));Type="JimmyExample";Types={"JimmyExample"};

// GetByLinks testing// First, no filter (should get Judy and Brobot)log
("Testing GetByLinks -- finding all links from Jimmy");Filter = "";
CountOnly = false;JimmyDataItems = {};
JimmyDataItems = JimmyDataItems + JimmyDataItem;
JimmyLinks = GetByLinks(Types, Filter, null, JimmyDataItems);
i = 0;while (i < Num) {    i = i + 1;
    log("JimmyLinks[" + i + "] => " + JimmyLinks[i-1].FirstName);}//
 Provide a filter this time.log
 ("Testing GetByLinks -- finding Judy"); \
 Filter = "FirstName = 'Judy'";CountOnly = false;
 Judy = GetByLinks(Types, Filter, null, JimmyDataItems);if (Num == 1) {
    log("Found Judy!  => " + Judy[0].FirstName + " " + Judy[0].LastName);} else {
    log("Error: Didn't find Judy!");}
```

Sending data

To send data to the socket server, you call `AddDataItem` and pass the name of a socket DSA data type and a context that contains a set of name/value pairs.

When Netcool/Impact encounters the call to `AddDataItem`, it passes the data type name and the set of name/value pairs to the socket DSA. The socket DSA sends these to the socket server. The socket server then analyzes the request and uses the data in the name/value pairs to perform an operation such as adding a new row to a database or sending a message to a messaging system.

The following example shows how to send data to the sample socket server distributed with the DSA. The code that handles requests to send data is located in the `UserDataInterface.pm` file.

```
// Test AddDataItemlog("Testing AddDataItem -- Adding Hugh Example")
;Jimmy=NewObject();
Jimmy.FirstName = "Hugh";
Jimmy.LastName = "Example";
Jimmy.Hobby = "Ducks";
ObjectToCopy=Jimmy;
AddDataItem(Type, ObjectToCopy);
```

Working with the sample socket server

The socket DSA provides a sample socket server written in Perl that you can use to test and explore the function of the DSA. You can also customize the sample server and use it as part of your real world socket DSA solution.

Note: The best practice is to become familiar with the sample socket server before you attempt to implement a custom socket server using any other development tools. The sample socket server is the best way to get started using the Socket DSA.

Setting up the sample socket server

Procedure

The DSA properties file contains settings for the host name and port of the socket server. This file is named `socketdsa.properties` and is located in the `$IMPACT_HOME/impact/dsa/socketdsa` directory. You must make sure that the properties in this file reflect the actual location of the server.

Sample socket server components

The sample socket server consists of the following components:

- `Server.pl`, which contains the main server framework and the function required to communicate with the socket DSA
- `UserDataInterface.pm`, which contains the data source-facing function of the sample server

Server.pl

`Server.pl` contains the main server framework and the function required to communicate with the socket DSA. You can run `Server.pl` with version 5.8 and later of the Perl interpreter. The `Server.pl` script is designed to work as provided. No additional customization is required. You can, however, rewrite this script to better suit your needs. To customize the sample socket server, change the `UserDataInterface.pm` module.

`Server.pl` uses the `Net::Server` module to communicate across a network with the DSA. `Net::Server` is a freely available Perl module that provides the core function required to build a server that communicates with other applications using Internet protocols. For more information about `Net::Server`, see `http://seamons.com/net_server.html`.

`Server.pl` contains the Netcool/Impact-facing function of the sample server. To handle requests from the socket DSA to return data from or add new data to a data source, it uses calls to functions defined in the `UserDataInterface.pm` module.

At initialization, `Server.pl` binds to the port address specified by the `$portnum` variable. The default port address is `22180`.

After initialization, `Server.pl` waits for incoming messages from the socket DSA on the specified port. The socket DSA initiates each message exchange by sending the string `hi` to the port where the server is running. When the server receives the string, it replies with an identical `hi` message.

The server then waits to receive a request from the socket DSA. Each request starts with a message that contains the name of the operation to perform. The operation names correspond directly to the function names `GetByFilter`, `GetByKey`, `GetByLinks`, and `AddDataItem`. `Server.pl` responds to this initial message by requesting additional information from the socket DSA based on the parameters that are required to perform the operation. The parameters correspond to the parameters passed to the function from within a Netcool/Impact policy.

For example, when brokering a request for the `GetByFilter` operation, `Server.pl` asks the socket DSA for the name of the data type and the filter string. `Server.pl` assigns the contents of the replies from the DSA to the `$typename` and `$filter` variables.

When `Server.pl` has received the parameters required by a particular operation, it calls the corresponding function defined in `UserDataInterface.pm` and passes the parameter data that it received from the socket DSA. `UserDataInterface.pm` assembles the result set for the request and returns it to `Server.pl`, which in turn sends the results back to the DSA.

`Server.pl` sends the results back to the socket DSA as sets of name/value pairs, where each set represents a data item and each name/value pair represents a data item field. The format of the results is a series of messages, where each name and value is sent as a distinct message and an empty string is sent to signify the end of a data item.

UserDataInterface.pm

`UserDataInterface.pm` is a Perl module that contains the data source-facing function of the sample socket server. This module is responsible for acquiring the information requested by the socket DSA from the underlying vendor software, device or system, and for passing on new information that originated with Tivoli Netcool/Impact.

By default, `UserDataInterface.pm` uses sample data hard-coded into the Perl module file. This data is suitable for use when learning about socket servers and when running the sample policies that are distributed with the DSA. When you create a custom solution based on the sample socket server, you modify `UserDataInterface.pm` so that it works with data sources specific to your environment.

`UserDataInterface.pm` contains one function for each of the operations supported by the socket DSA. These functions are `GetByFilter`, `GetByKey`, `GetByLinks`, and `AddDataItem`. `Server.pl` calls these functions when it brokers requests from the socket DSA. For example, when the socket server receives a request from the DSA to perform a `GetByFilter` operation, it calls the `GetByFilter` function defined in `UserDataInterface.pm`.

In the case of `GetByFilter`, `GetByKey`, and `GetByLinks`, `UserDataInterface` receives parameters that specify the terms of the operation from `Server.pl` and then returns either a single hash (in the case of `GetByKey`) or an array of hashes (in the case of `GetByFilter` and `GetByLinks`). In all cases, each hash represents a single data item, where the name/value pairs that it contains represent data item fields.

In the case of `AddDataItem`, the function receives parameters that specify the contents of the new data element and returns a single hash that represents the new data that has been passed to the vendor software, device, or system.

Note that all these functions require `Server.pl` to pass the name of an underlying data type. In the default functions provided with `UserDataInterface.pm`, the type name is used with select statements to determine the appropriate information to perform for each type of data. The data type name can also be used in a more general way to specify different types of operations that you want the socket server to perform that are not necessarily associated with underlying data sets.

When you customize the sample socket server, you rewrite one or more of these functions to either return the appropriate sets of data or send new data to the third-party data source. You can use these functions to specify any manner of operations, such as calls to Perl database drivers or calls to custom interfaces that you have written to work with third-party systems.

Running the sample socket server

Procedure

1. Before you run `Server.pl`, you must modify the first line of the file so that it specifies the location on the file system where Perl is installed. If you do not modify the first line, you must explicitly invoke the Perl interpreter when you run the script.

2. You must also set the PERL5LIB environment variable so that it includes the directory where you installed the sample server.

   For example, if you installed the server in /usr/local/socketdsa/SocketDSAServer/Server.pl, you can set this variable in bash or sh by entering the following command at a command prompt:

   ```
   PERL5LIB=/usr/local/socketdsa; export PERL5LIB
   ```

   The directory that you specify must be two levels up from Server.pl.
3. To run Server.pl, enter the following command at a command line prompt:

   ```
   Server.pl -port port_number
   ```

   where *port_number* is the port where you want the sample socket server to run. If you do not specify a port, the server uses 22180, which is the default.

   You can also run Server.pl by explicitly invoking the Perl compiler as follows:

   ```
   perl Server.pl -port port_number
   ```

Testing the socket server

The socket DSA provides a command line client that you can use to test the availability of socket servers, including the sample server provided in the DSA tar file. You use this client to send messages to a socket server using a simple command line input. The client is named TestClient and is located in the DSA jar file.

About this task

Procedure

- To test the socket server, enter the following command at a command-line prompt on the system where you are running Tivoli Netcool/Impact:

  ```
  java com.micromuse.dsa.socketdsa.TestClient hostname
      port
  ```

  where *hostname* is the name of the system where the socket server is running and *port* is the port number used by the server.
- To test the availability of a socket server, enter the following string at the command line:

  ```
  hi
  ```

  The test client sends this string to the socket server and prints the response. If the socket server is running correctly, the response will be a hi string identical to the one sent from the command line.

What to do next

You can perform additional testing by entering additional strings at the command line, following the command sequence documented in the code comments in Server.pl.

Implementing a custom socket server

If you do not want to use the sample socket server that is distributed with the DSA, you can implement your own using any scripting or programming language that provides access to network sockets. These languages include Java, C/C++, and Perl.

A custom socket server must perform the following tasks:

- Create a receiver socket and bind to a port

- Wait for DSA connections and create connection-specific sockets
- Perform handshaking with the DSA
- Listen for operation requests from the DSA
- Request the operation parameters from the DSA
- Perform the operations requested by the DSA
- Return operation results to the DSA

Creating a socket

At startup, the custom socket server must create a new socket and bind to the port that it will use for communication with the socket DSA. You specify this port in the DSA properties file when you configure the DSA, as described in "Configuring the socket DSA" on page 143.

Waiting for DSA connections

Procedure

After you created a new socket, the socket server must listen at the port for a connection from the socket DSA. When a connection arrives, the server must create a new socket to use for communication specific to that connection.

Performing handshaking with the DSA

Procedure

After the DSA establishes a connection, it sends the greeting string `hi` to the socket server. The socket server must reply with its own identical `hi` message in order for handshaking to be complete.

Listening for operation requests from the socket DSA

The socket DSA is capable of sending the following operation requests to the socket server:

- `GetByFilter`
- `GetByKey`
- `GetByLinks`
- `AddDataItem`

After the DSA and the server exchange handshaking messages, the DSA sends an operation request to the server. The operation request is a message that consists of the name of the operation (for example, `AddDataItem`). The socket server must accept this request and determine which tasks to perform based on the contents of the message.

Requesting operation parameters from the socket DSA

After the socket server has received the operation request from the socket DSA, it must request the operation parameters from the DSA one at a time in a series of messages. The DSA replies by sending the parameter values as specified in the call to `GetByFilter`, `GetByKey`, `GetByLinks` or `AddDataItem` in the Netcool/Impact policy.

The following table shows the contents of the messages that the socket server must send to the socket DSA to request the parameters for a `GetByFilter` operation.

Table 48. GetByFilter Operation Request Messages

Request Message	Description
`sendtype`	Requests the name of the data type associated with the operation. This is the `DataType` parameter specified in the call to the `GetByFilter` function in a Netcool/Impact policy. The DSA returns a string that contains the data type name.
`sendfilter`	Requests the filter string associated with the operation. This is the `Filter` parameter specified in the call to the `GetByFilter` function in a Netcool/Impact policy. The DSA returns a string that contains the filter.

The following table shows the contents of the messages that the socket server must send to the socket DSA to request the parameters for a `GetByKey` operation.

Table 49. GetByKey Operation Request Messages

Request Message	Description
`sendtype`	Requests the name of the data type associated with the operation. This is the `DataType` parameter specified in the call to the `GetByKey` function in a Netcool/Impact policy. The DSA returns a string that contains the data type name.
`sendkey`	Requests the filter string associated with the operation. This is the `Key` parameter specified in the call to the `GetByKey` function in a Netcool/Impact policy. The DSA returns a string that contains the filter.

The following table shows the contents of the messages that the socket server must send to the socket DSA to request the parameters for a `GetByLinks` operation.

Table 50. GetByLinks Operation Request Messages

Request Message	Description
`sendfromtype`	Requests the name of the source data type associated with the operation. This is the data type of the first element in the `DataItems` parameter specified in the call to the `GetByLinks` function in a Netcool/Impact policy. The DSA returns a string that contains the data type name.
`sendfromkey`	Requests the filter string associated with the operation. This is the `Key` parameter specified in the call to the `GetByKey` function in a Netcool/Impact policy. The DSA returns a string that contains the filter.
`sendtotype`	Requests the name of the target data type associated with the operation. This is the data type of the first element in the `DataTypes` parameter specified in the call to the `GetByLinks` function in a Netcool/Impact policy. The DSA returns a string that contains the data type name.
`sendfilter`	Requests the filter string associated with the operation. This is the LinkFilter parameter specified in the call to the `GetByLinks` function in the Netcool/Impact policy. The DSA returns a string that contains the filter.

The following table shows the contents of the messages that the socket server must send to the socket DSA to request the parameters for a `AddDataItem` operation. Note that `AddDataItem` returns a set of name/value pairs to the Netcool/Impact server that represent the contents of the new data item added.

Table 51. AddDataItem Operation Request Messages

Request Message	Description
sendtype	Requests the name of the data type associated with the operation. This is the `DataType` parameter specified in the call to the `AddDataItem` function in a Netcool/Impact policy. The DSA returns a string that contains the data type name.
`sendattributes`	Requests the attributes of the data item associated with the operation. These are a series of name/value pairs that represent the member variables in the `ContentToCopy` parameter specified in the call to `AddDataItem`. The DSA returns a series of names and values, each of which is a separate string. The DSA indicates that there are no more attributes in the data item by sending an empty string.

Performing operations requested by the DSA

Procedure

After the socket server requests the parameters from the socket DSA, it can perform operations to retrieve data from or add data to the underlying software, device or system. For example, you can use the information sent by the socket DSA to query an external database or to send a message on a message system.

Returning operation results to the DSA

Procedure

After the socket server has performed the requested operation, it can return the results to the DSA. The results must be returned as a series of messages that describe the contents of the data items resulting from the operation. The first message in this series is a string that contains the number of data items that will be returned. Following this are sets of messages that contain name/value pairs that represent data item fields. The socket server indicates the end of each data item by sending a newline character.
For more details, refer to the sample socket implementation and to the inline comments in the socket server code.

Socket DSA and socket server connection state

The connection state between the socket DSA and a socket server is affected when either the DSA or the server goes down during the communication process.

If the socket DSA goes down, the server will stay up. Any communication between the components is terminated.

If the socket server goes down, the DSA sends messages to the server log that indicate that it cannot connect to the server. The DSA will then try to reconnect one time before terminating the communication process with the socket server.

When the socket server is brought back up, the DSA will automatically reconnect the next time it handles a request for an operation from Netcool/Impact. If it cannot reconnect, it will send a message to the server log indicated that it was not able to communicate with the socket server.

The socket DSA and sample socket server do not time out connections after a certain length of time. You can extend the sample socket server to handle timeouts using information in the `Net::Server` documentation.

Socket server threading

If you have configured Netcool/Impact to use a multi-threaded event processor, the best practice is to run the socket server as a multi-threaded application.

The default behavior of `Server.pl` is to allow multiple threads based on UNIX forking. This function is provided by the `Net::Server` module, which provides a flexible set of threading options that you can use to adapt to your specific implementation.

Recent versions of Perl also provide additional options for managing application threading.

Chapter 14. Working with the Cramer DSA

[**Important:** This feature is deprecated.] The Cramer Dimension DSA is a customized XML DSA that you can use to integrate network inventory data with Netcool/OMNIbus.

Restriction: The integration was tested against Cramer Dimension version 4. However, Cramer Systems has confirmed that the interface for Cramer Dimension version 5 is backward compatible with the tested version.

Cramer Dimension DSA overview

The Cramer Dimension DSA provides a communication layer between Netcool/Impact and Cramer Dimension.

Netcool/Impact uses the DSA to send queries via HTTP to the Cramer Dimension server. The server responds with an XML string that contains the information requested. The DSA then populates a set of data items in Netcool/Impact with the information in the XML string.

To set up the Cramer Dimension DSA, you must install it, and then configure it by editing its HTTP types file. After you install the DSA, it runs as a process in the Impact Server, so you do not have to start or stop it independently.

The Cramer Dimension data model consists of a data source and two sets of data types that are created automatically when you install the DSA.

You retrieve network inventory information stored in Cramer Dimension from within a policy. A set of sample policies is provided with the DSA that demonstrate how to use the Cramer Dimension DSA, before you use it in your production environment. You will use the Cramer Dimension DSA in your custom policies by calling the GetByFilter, and GetByLinks functions.

Files used with the Cramer Dimension DSA

A list of files provided with, or used with the Cramer Dimension DSA.

File	Description
DSA Properties File	The `$IMPACT_HOME/dsa/XmlDsa/XmlDsa.properties` file.
XML Setup Scripts	These files located in the `$IMPACT_HOME/dsa/XmlDsa/bin` directory.
XML Configuration Script	The `$IMPACT_HOME/dsa/XmlDsa/XmlHttpTypes` file.
How-To Documents	The files located in the `$IMPACT_HOME/dsa/XmlDsa/docs` directory.
PERL script	The `$IMPACT_HOME/add-ons/cramer/DynamicFilter` script can be used as an example for the ObjectServer.
Sample Implementation	You can find sample Cramer DSA policies, and data types in `$IMPACT_HOME/add-ons/cramer/importData`. You must run `nci_import` to import them into your Netcool/Impact installation.

Setting up the Cramer Dimension DSA

Use this procedure to set up the Cramer Dimension DSA.

Before you begin

- Familiarize yourself with the integration by reading the Cramer DSA documentation, in the `$IMPACT_HOME/add-ons/cramer/docs` directory.
- Configure Cramer System (SAA Adapter) to use basic authentication. For more information, see "Configuring Cramer System to use basic authentication" on page 157.
- Obtain the realm details, and the connection information from the Cramer System administrator, and test if the realm on the Cramer side is set up. You should be able to sign on to the realm, for example, if you type this address in your browser address bar:

  ```
  http://hostname:port
  /Cramer5/httpgateway/RequestDispatcherAction.do?instance=hostname
  ```

  where *hostname*, and *port* is the Cramer System address, and port.

Procedure

1. Import the Cramer project into the Impact Server, using the `nci_import` script.

   ```
   $IMPACT_HOME/bin/nci_import NCI $IMPACT_HOME/add-ons/cramer/importData
   ```

   Running this command creates a Cramer project that contains sample policies and data types for Cramer DSA.

   Attention: You must import the project into a running Impact Server. Make sure no files are locked on the server when you import the Cramer project, or the import will fail.
2. Navigate to the `$IMPACT_HOME/add-ons/cramer` directory, and copy the following files to the `$IMPACT_HOME/dsa/XmlDSA` directory:
   - `cramerDim.dtd`
   - `cramerOr.dtd`
3. Update the `$IMPACT_HOME/dsa/XmlDSA/XmlHttpTypes` file with the following entries for Cramer DSA:

   ```
   XmlDsa.httpTypes.1.typeName=CramerDim
   XmlDsa.httpTypes.1.dtdFile=dsa/XmlDsa/cramerDim.dtd
   XmlDsa.httpTypes.1.prefix=CramerDim_
   XmlDsa.httpTypes.1.url=
   http://198.51.100.218:7777/pls/dat2/HTTPGateway.Listener
   XmlDsa.httpTypes.1.user=Cramer
   XmlDsa.httpTypes.1.password=
   {aes}337C5EF0D2D85CF4420F856E325DA084
   XmlDsa.httpTypes.1.realm=dat2
   XmlDsa.httpTypes.2.typeName=CramerOR
   XmlDsa.httpTypes.2.dtdFile=dsa/XmlDsa/cramerOR.dtd
   XmlDsa.httpTypes.2.prefix=CramerOR_
   XmlDsa.httpTypes.2.url=
   http://198.51.100.218:7777/pls/dat2/HTTPGateway.Listener
   XmlDsa.httpTypes.2.user=
   Cramer
   XmlDsa.httpTypes.2.password=
   {aes}337C5EF0D2D85CF4420F856E325DA084
   XmlDsa.httpTypes.2.realm=dat2
   ```

Attention: Change the sample values in the example to real values, in particular the URL, user, password, and possibly the realm properties. The password must be encrypted using nci_crypt, and you must paste the encrypted string as the value for the `XmlDsa.httpTypes.#.password` property. For more information about using the nci_crypt tool, see the *"nci_crypt"* section, in the *Administration Guide*. Modify the properties indices according to how many sets of properties you already have in your `XmlHttpTypes` file.

4. Restart the Impact Server for the changes to take effect.

Configuring Cramer System to use basic authentication

You must configure Cramer System to use basic authentication so that it can work with Netcool/Impact.

Procedure

1. Remove the SSO Authentication/Application Filter.

   To do that, edit the `<deployment_root>/applications/HTTPGatwewayWAR/httpgateway/WEB-INF/web.xml` file within the HTTP Gateway Web-application.
2. Configure the Web-container to enable HTTP basic authentication for the HTTP Gateway Web application.

   Edit the `orion-application.xml` file, by following the **Advanced Properties** link under **HTTPGatewayWAR**, in the administration console.
3. Configure appropriate basic authentication user names and passwords within the Web container.

   In the administration console, follow the **Security** link under **HTTPGatewayWAR**, and add the `ntlcra/ntlcra` user to the `jazn.com` realm.
4. Map each basic authentication user to an appropriate Cramer distinguished name (DN) within the HTTP Gateway Web-application.

   Edit the `<deployment-root>/applications/HTTPGatwewayWAR/httpgateway/WEBINF/resources/xml/usermappings.xml` file.

Cramer Dimension data model

The Cramer Dimension data model consists of a data source and two sets of data types that are created automatically when you install the DSA.

This data model is capable of holding all the data returned by queries to Cramer Dimension.

Note: When you install the Cramer Dimension DSA, the data source and data types are inserted into the Impact Project called Cramer on the target Impact Server.

Cramer Dimension data source

The Cramer Dimension data source, XmlDsaMediatorDataSource, represents the Cramer Dimension server as a source of data for use in Netcool/Impact policies.

The DSA installer automatically sets the configuration properties for XmlDsaMediatorDataSource. Do not change the data source name or the Mediator class name after installation.

Cramer Dimension data types

The Cramer Dimension data model contains two sets of data types, and each set corresponds to a DTD file that describes a set of data that Netcool/Impact can request from the Cramer Dimension server.

These DTD files are provided by Cramer Dimension and need to be copied from `$IMPACT_HOME/add-ons/cramer` directory to `$IMPACT_HOME/dsa/XmlDsa` directory when you install the DSA. As with the XmlDsaMediatorDataSource, the data types are inserted into the Impact porject called Cramer on the target Netcool/Impact server. Again, the DTD files need to be located in `$IMPACT_HOME/dsa/XmlDsa`.

The Cramer Dimension data types have a mapping relationship with the corresponding DTD file. In this relationship, there is one data type for each XML element defined in the DTD. Each XML attribute in the DTD file is specified as a field in the corresponding data type.

The first set of data types are CramerDim data types. These correspond to elements in the cramerDim.dtd file. This DTD specifies the contents of XML strings in the Dimension format. When Netcool/Impact requests information in this format from the Cramer Dimension server, the content of the XML string that is returned reflects the properties of this DTD file. CramerDim data types begin with the CramerDim_ prefix.

The second set of data types are CramerOR data types. These correspond to elements in the cramerOR.dtd file. This DTD specifies the contents of XML strings in the Object Reference format. CramerOR data types begin with the CramerOR_ prefix.

The Dimension and Object Reference formats define similar sets of data. To determine which format you should use, you should be familiar with the DTD files and your Cramer Dimension configuration.

Cramer Dimension policies

You can find sample Cramer Dimension policies that demonstrate how to use the Cramer Dimension DSA in the Cramer project.

You can use these policies with minor modification to retrieve information from Cramer Dimension and use it to accomplish a variety of tasks with Netcool/Impact. You can also review these policies to better understand how to use the Cramer Dimension DSA in your environment. You can also find an example policy, `CramerDirector.ipl`, in the `$IMPACT_HOME/add-ons/cramer/docs` directory.

In your own Cramer Dimension policies, you use GetByFilter to retrieve the XML data from the Cramer Dimension server. When you call the GetByFilter function, you specify a top-level Cramer data type and a filter string as runtime parameters. When Netcool/Impact encounters this operation in a policy, the DSA sends a request by HTTP to the Cramer Dimension server.

The server responds with an XML string that contains the requested data. The DSA then populates the Cramer Dimension data model with the information in the XML string.

You can traverse the returned XML data within the policy, using the GetByLinks function or the .links notation and use the data to enrich Netcool/OMNIbus events or to perform other tasks.

You can also access XML element and attribute values from within your Cramer Dimension policy.

Retrieving XML Data from Cramer Dimension

The first step in writing a policy is to call GetByFilter and pass the name of a top-level Cramer Dimension data type and a filter string.

When Netcool/Impact encounters this statement in the policy, the DSA makes an HTTP request to the Cramer Dimension server for the specified information. The DSA then uses the information in the XML string returned by the server to populate data items in the corresponding set of data types.

The filter string that you use with the call to GetByFilter uses a special format that provides the HTTP operation type, the relative URL path of the HTTP gateway listener on the Cramer Dimension server, and an XML string that specifies what data to return to Netcool/Impact. The filter string has the following format:

```
"Operation = POST AND FilePath = URLPath AND XMLRequest = XmlString"
```

Note: The filter argument values must be enclosed in double quotes "" and separated by the key word AND.

The URL path of the HTTP gateway listener is specified as the path relative to the root of the Cramer HTTP adaptor. The following is an example of a valid path:

```
pls/dat2/HTTPGateway.Listener
```

This path corresponds to a location on the server such as `http://192.168.1.1:7777/pls/dat2/HTTPGateway.Listener`.

The XmlString parameter is a segment of XML data that specifies what data to return to Netcool/Impact. You can create this string manually, or you can use the standard policy CramerRequestMaker to automatically assemble the XML string using the appropriate format. For information on the format of the XML query string, see the comments in the CramerRequestMaker policy. This policy is installed automatically when you install Netcool/Impact and is located by default in the Global Repository.

The following example shows how to retrieve an XML data set from Cramer Dimension using GetByFilter. In this example, you use the CramerRequestMaker policy to generate the XML query string. This example uses the CramerDim format.

```
// Generate the XML query string using CramerRequestMaker.
// CramerRequestMaker stores the resulting query string
// in the XmlString variable.
QueryType = "TRAFFIC";
InputFormat = "DIM";
OutputFormat = "DIM";
DimObjectReference = newobject();
DimObjectReference.Node = 1755;
Activate(Null, "CramerRequestMaker");
// Call GetByFilter and pass the top-level CramerDim data type
// and the filter string as runtime parameters.
Type = "CramerDim";
Filter = "Operation = POST AND FilePath = pls/dat2/HTTPGateway.Listener AND \
XmlString = " + XmlString + "";
CountOnly = False;
Results = GetByFilter(Type, Filter, CountOnly);
```

Using GetByLinks to traverse the XML data

After you retrieved the desired set of XML data from the Cramer Dimension server, the next step is to traverse the data within the policy so that you can access individual XML elements and their attributes.

When the DSA populates data items in the data model with XML information that it retrieved from the Cramer Dimension server, it creates static links between individual data items. These links represent the containment relationship between the XML elements. Data items that represent an XML element are statically linked to other data items that represent their child nodes.

One way to traverse the XML data is by using successive calls to GetByLinks. To traverse the XML data in this way, you must first retrieve the top-level data item by calling GetByLinks and passing the name of the top-level data type as a runtime parameter. Then you can call GetByLinks any number of subsequent times to retrieve the data stored in child XML nodes.

The following example shows how to traverse the XML data using GetByLinks. In this example, the data returned from the Cramer Dimension server is in CramerDim format and is as follows:

```
<ENVELOPE DESTINATION="Netcool" SENDER="Fault Manager Adaptor">
<BODY>
<GETSERVICERESPONSE>
<OUTPUTFORMAT>Cramer FMA Object Ref</OUTPUTFORMAT>
<SERVICELIST>
<SERVICE DIMENSIONID="42990" NAME="FMA Test Service" SERVICETYPE="Standard"
 STATUS="Maintenance" PROTECTIONSTATUS="Affected">
<SUBSCRIBER DIMENSIONID="1176" NAME="TestSub_2"></SUBSCRIBER>
</SERVICE>
<SERVICE DIMENSIONID="42991" NAME="Cable TV" SERVICETYPE="Standard"
 STATUS="On" PROTECTIONSTATUS="Affected">
<SUBSCRIBER DIMENSIONID="1177" NAME="TestSub_5"></SUBSCRIBER>
<SUBSCRIBER DIMENSIONID="1178" NAME="TestSub_6"></SUBSCRIBER>
</SERVICE>
</SERVICELIST>
</GETSERVICERESPONSE>
</BODY>
</ENVELOPE>
```

To traverse this data, you can first call GetByLinks and pass the name of the root-level data type as a runtime parameter. GetByLinks returns an array that contains a single data item. This data item represents the root-level ENVELOPE element in the XML data. In subsequent calls, the elements as defined in the xml file&apos;s hierarchy are returned, preserving the hierarchical relationship as it exists in the xml file. In each subsequent call, GetByLinks returns a variable in which each data item is associated with a child node of the source data item.

Note: GetByLinks always returns an array.

The following code segment continues from the policy example in Retrieving XML Data from Cramer Dimension, in which a call to GetByFilter returned an array named Results.

```
// Call GetByLinks and pass the name of the root-level data type.
// In this instance, the data type is CramerDim_ENVELOPE.
DataTypes = {"CramerDim_ENVELOPE"};
```

```
LinkFilter = "";
Envelope = GetByLinks(DataTypes, LinkFilter, 1, Results);
// Call GetByLinks and pass the name of a data type that represents
// a child node of ENVELOPE.
DataTypes = {"CramerDim_BODY");
LinkFilter = "";
Body = GetByLinks(DataTypes, LinkFilter, 1, Envelope);

// Call GetByLinks and pass the name of a data type that represents
// a child node of BODY.
DataTypes = {"CramerDim_GETSERVICERESPONSE");
LinkFilter = "";
GetServiceResponse = GetByLinks(DataTypes, LinkFilter, 1, Body);
```

Using the Embedded Linking Syntax to traverse the XML data

You can also use the embedded linking syntax to traverse the XML data returned from the Cramer Dimension server.

The embedded linking syntax is easier to use than many successive calls to GetByLinks. The following example shows how to use the linking syntax to retrieve the first child element data item linked to the root-level data item, where the data type of the child item is CramerDim_BODY. In this example, you have previously retrieved an array named Envelope using a call to GetByLinks.

```
Body = Envelope[0].links.CramerDim_BODY.first;
```

Accessing XML element and attribute values

You can access XML element and attribute values from within a policy by referencing the PCDATA variable or attribute member variables in the corresponding data item.

The value of an XML element is stored in a member variable named PCDATA and the value of any associated XML attributes is stored in member variables that have the same name as the attribute.

The following example shows how to access the value of the OUTPUTFORMAT element in the XML fragment used in previous examples. In this example you have already obtained the data item that corresponds to the element using GetByLinks and you have stored the item in a variable named OutputFormat.

```
Log(OutputFormat[0].PCDATA);
```

This statement prints the following to the policy log:

```
Cramer FMA Object Ref
```

The following example shows how to access the value of the DESTINATION attribute in the ENVELOPE element in the XML fragment used in previous examples. In this example you have already obtained the data item that corresponds to the element using GetByLinks and you have stored the item in a variable named Envelope.

```
Log(Envelope[0].DESTINATION);
```

This statement prints the following to the policy log:

```
Netcool
```

Sample Implementation

The DSA for Cramer Dimension installation comes with an optional sample implementation that demonstrates basic alarm suppression and customer impact analysis.

To use this solution, you must first make the following changes to the ObjectServer, and Netcool/Impact:

- Add fields to the ObjectServer.
- Add data types to Netcool/Impact, in addition to the data types that were created during the installation.
- Configure the Event Reader to start different policies when events occur in the ObjectServer.

Updating the ObjectServer

To use the sample implementation you must create fields in the ObjectServer.

Add the following fields:

Table 52. Fields to be added to the ObjectServer (for integrated solution)

Field	Field type	Field size	Description
Shelf	varchar	64	Shelf number
Slot	varchar	64	Slot number
Card	varchar	64	Card number
PhysicalPort	varchar	64	Physical Port number
LogicalPort	varchar	64	Logical Port number
CramerImpact	int		Specifies that the event is to be processed by the Cramer/Impact application. A value of 1 to 9 specifies that the Event Reader has to process the event. A value of 2 also specifies that Netcool/Impact must conduct a customer impact analysis of the event. A value of 10 specifies that the event will not be processed.
Correlatable	int		Identifies for Netcool/Impact whether the event is a Probable Fault Event (PFE) (value of 1) or a Probable Symptom Event (PSE) (value of 2).
CorrelatedTo	int		The unique ID of the PFE for this PSE.
Service	varchar	64	Identifies the affected service.
Customer	varchar	64	Customer name.

After you added these fields to the ObjectServer, you should create an event list view that shows these fields. For information about adding fields to the ObjectServer, see the *Netcool/OMNIbus Administration Guide*. For information about creating an event list view, see the *Netcool/OMNIbus User Guide*.

Configuring Netcool/Impact

To set up the sample implementation, you must make the following changes to Netcool/Impact:

- Create a new data type called CramerEvents.

- Configure the event reader to handle incoming events.

CramerEvents data type

The CramerEvents data type is an ObjectServer data type. You must create this data type and provide connection information for the ObjectServer that you modified in the previous step.

Event Reader

The event reader service must be set up to point to the ObjectServer providing the events. The filter for the event reader is be similar to the following example:

```
( (CramerImpact > 0) AND (CramerImpact < 10) )
```

You must also configure the event reader so that its event mappings run the appropriate policies when certain events occur in the ObjectServer. For information about setting up event mappings, see the *User Interface Guide*.

DSA for Cramer Dimension Standard Policies

The DSA for Cramer Dimension is installed with a collection of standard policies.

Extensive inline comments within each policy file describe the policy's runtime parameters, outputs, and behavior. The following standard policies are provided:

CramerDimTestPolicy
You can use the CramerDimTestPolicy to help you understand how to get data from the Cramer Dimension database using the Dimension ID format. Refer to the description tag of this policy and the comments included in the policy for more information.

CramerORTestPolicy
The CramerORTestPolicy is provided to help you understand how to get data from the Cramer Dimension database using the Object Reference format. Refer to the description tag of this policy and the comments included in the policy for more information.

ServiceImpactAnalyzer
The ServiceImpactAnalyzer policy analyzes the impact on services that are related to the involved event. It queries the Cramer Dimension database for all the services that are affected by the event and then sends new Service Impact events to the ObjectServer for each service that is affected by this event.

PFEProcessor
The PFEProcessor (Probable Fault Event Processor) policy is run for events that are marked as probable root causes of events. It queries the Cramer Dimension database for all the circuits and related ports that are affected by the event (as specified in the node+Shelf+Card+Slot of the event).

It then updates those particular port/circuit events in the ObjectServer, setting the CorrelatedTo field with the unique ID of the event that suppressed this event.

PFSProcessor
The PFSProcessor (Probable Fault Symptoms Processor) policy is run for all events that are marked as probable symptom events. It queries the Cramer Dimension database for all the circuits and related ports that are related to the event.

It will then update its CorrelatedTo field to the unique ID of the event that matches the related events that came back from the Cramer Dimension database. In other words it is suppressed by that event.

CramerGetORTraffic
This policy returns a list of circuits that are affected by an event. It is in the object reference format.

CramerGetORServices
The CramerGerOrServices policy returns a list of services that are affected by an event. It is in the object reference format.

CramerRequestMaker
The CramerRequestMaker policy is a utility routine that produces the xml string that is needed by Cramer Dimension. You should review the CramerRequestMaker policy to ensure that it is building xml strings with the parameters required by your Cramer Dimension installation.

CramerGetORTraffic parameters

The CramerGetORTraffic policy has the following runtime parameters:

DimObjectReference
This variable provides the context for the object for which the affected circuits are being determined. This is provided in the form of the Cramer Dimension ID. Either this parameter or the ORObjectReference parameter should be null. A typical value for this variable would look like:

```
DimObjectReference.NODE = 1755
```

ORObjectReference
This variable provides the context for the object for which the affected circuits are being determined. This is provided in the form of object references. A typical value for a card that was affected is given as:

```
ORObjectReference.NODE = "Core 1";
ORObjectReference.Shelf = 1;
ORObjectReference.Slot = 2;
ORObjectReference.Card = 10;
```

Output parameters return a list of circuits that were affected by the event. The result is an array of circuit objects, that look like the following example:

```
Circuit.TrafficType = "Terminating";
Circuit.CircuitType = "SDH MS Bearer";
Circuit.Status = "Maintenance";
Circuit.DimensionId = 1234;
Circuit.Name = "MSB Core1 Hub 1";
Circuit.Aend_Node_Name = "Core 1";
Circuit.Aend_Node_Status = "Maintenance";
Circuit.Aend_Shelf_Number = 1;
Circuit.Aend_Slot_Number = 1;
Circuit.Aend_Slot_Position = 1;
```

The name of the array is CircuitArray. The number of circuits in this array is given by CircuitCount.

CramerGetORServices parameters

The CramerGetORServices policy has the following runtime parameters:

DimObjectReference
This variable provides the context for the object for which the affected

circuits are being determined. This is provided in the form of the Dimension ID. Either this parameter or the ORObjectReference parameter should be null. A typical value for this variable would looks like:

```
DimObjectReference.NODE = 1755
```

ORObjectReference
This variable provides the context for the object for which the affected circuits are being determined. This is provided in the form of object references. A typical value for a card that was affected is given as:

```
ORObjectReference.NODE = "Core 1";
ORObjectReference.Shelf = 1;
ORObjectReference.Slot = 2;
ORObjectReference.Card = 10
```

Output parameters return a list of circuits that were affected by the event. The result is an array of circuit objects that look like the following example:

```
Service.Name = "ServiceToMicromuse";
Service.Type = "GOLD";
Service.Status = "Maintenance";
Service.Protection = "Possibly Affected";
Service.Subscriber = "Micromuse";
```

The name of the array is ServiceArray. The count of services is given by ServiceCount.

CramerRequestMaker parameters

The CramerRequestMaker policy has the following runtime parameters:

QueryType
This parameter applies whether it is a GetTraffic or GetService type. Possible values are TRAFFIC and SERVICE. The default is TRAFFIC.

InputFormat
This parameter specifies the format of the input request. Possible values are DIM and OR. The default is DIM. It specifies whether is in a Dimension ID format or an object reference format.

OutputFormat
This parameter specifies the response format. Valid values are DIM or OR. The default is DIM.

CircuitTypeFilterList
This parameter is an array object that contains a list of circuit filters. This array is valid only when the QueryType is set to TRAFFIC.

ServiceTypeFilterList
This parameter is an array object that contains a list of service filters. This array is valid only when the QueryType is set to SERVICE.

DimObjectReference
This parameter holds the context that is needed to represent an object in the Dimension ID format. It is made up of Node, Card, and Port. For example, if the object reference is a Card with a Dimension ID of 2026, the object will have a value similar to the following example:

```
DimObjectReference.Node = 0;
DimObjectReference.Card = 2026;
DimObjectReference.Port = 0;
```

Only one of these members can have a non-zero value.

This parameter is only relevant when InputFormat is set to DIM (Dimension ID-based).

ORObjectReference
This parameter holds the context that is needed to represent an object in the object reference format. It is made up of Node, Shelf, Slot, Card, PhysicalPort, and LogicalPort. Together they give a complete containment representation of a particular object's reference. A typical PhysicalPort reference is made up of the Node, Shelf, Slot, and Card and is represented by a value similar to the following example:

```
ORObjectReference.Node = 24; /* Node number 24 */
ORObjectReference.Shelf = 2; /* Shelf 2 in this node*/
ORObjectReference.Card = 7; /* Card 7 in this shelf */
ORObjectReference.Slot = 3; /* Third slot */
ORObjectReference.PhysicalPort = 31;
ORObjectReference.LogicalPort = 3;
```

This parameter is relevant only when InputFormat is set to OR (Object Reference-based).

The CramerRequestMaker policy has one output parameter:

XMLString
The XML string is created as a result of parsing the runtime parameters. It complies with the DTD for the respective runtime formats.

If the parameter has a null value, it implies that some of the runtime parameters contain invalid values.

Appendix A. 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

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

© Copyright IBM Corp. 2006, 2014

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

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

Printed in USA

SC27-4852-01