Netcool Configuration Manager 6.4.2

API Guide

2023-4219-01

Note

Before using this information and the product it supports, read the information in “Notices” on page 29.

This edition applies to version 6.4.2 of IBM Tivoli Netcool Configuration Manager (5725-F56) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this publication

IBM Tivoli Netcool Configuration Manager provides configuration management capabilities for network devices, as well as extensive configuration policy thresholding capabilities.

The *IBM Tivoli Netcool Configuration Manager API Guide* provides information about how to use the Java API to programmatically access Netcool Configuration Manager.

Intended audience

This publication is intended for programmers who want to use the Netcool Configuration Manager Java API to write applications that automate configuration tasks, perform data retrieval, and to integrate with external applications.

What this publication contains

This publication contains the following sections:

- Chapter 1, “The Application Programming Interface,” on page 1

  Provides information for anyone who wants to use Netcool Configuration Manager to manage and configure network resources.
- Chapter 2, “Installing the ITNCM - Base API,” on page 7

  Explains the files needed on the client side in order to code, test, and use the ITNCM - Base API.
- Chapter 4, “Terminology used when working with the Java API,” on page 11

  Describes the terminology associated with generalized and network resources.
- Chapter 5, “Netcool Configuration Manager - Base API Examples,” on page 17

  Provides examples on how to use the ITNCM - Base API.
- Chapter 6, “Known workarounds,” on page 27

  Describes workarounds associated with the ITNCM - Base API.

Publications

This section lists publications in the Netcool Configuration Manager PDF document set. The prerequisite publications in the IBM Tivoli Network Manager IP Edition and IBM Tivoli Netcool/OMNIbus library are also listed here. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

Netcool Configuration Manager PDF document set

The following documents are available in the Netcool Configuration Manager library:

- *IBM Tivoli Netcool Configuration Manager Installation and Configuration Guide*

  Describes how to install IBM Tivoli Netcool Configuration Manager. It also describes necessary and optional post-installation configuration tasks. This publication is for administrators who need to install and set up IBM Tivoli Netcool Configuration Manager.
- *IBM Tivoli Netcool Configuration Manager User Guide*

  Describes user tasks for IBM Tivoli Netcool Configuration Manager, such as how to access reports, use devices, and execute the different utilities to maintain and support Auto-Discovery. This publication is for users working with IBM Tivoli Netcool Configuration Manager.
- *IBM Tivoli Netcool Configuration Manager Administration Guide*

Describes administration tasks for IBM Tivoli Netcool Configuration Manager, such as how to set up user accounts, create and manage the OS registry, administer database and policy exports and imports, and perform housekeeping and security tasks. This publication is for administrators who are responsible for the maintenance and availability of IBM Tivoli Netcool Configuration Manager.

- *IBM Tivoli Netcool Configuration Manager Reference Guide*

  Contains reference information about IBM Tivoli Netcool Configuration Manager.

- *IBM Tivoli Netcool Configuration Manager API Guide*

  Provides information about how to use the Java API to programmatically access IBM Tivoli Netcool Configuration Manager.

- IBM Tivoli Netcool Configuration Manager NSM REST API Guide

  Describes the Service Management Interface API.

- *IBM Tivoli Netcool Configuration Manager Integration Guide*

  Describes how to integrate Netcool Configuration Manager with Tivoli Netcool/OMNIbus and Network Manager.

- *IBM Tivoli Netcool Configuration Manager Quick Start Guide*

  Gets you started with a typical installation for IBM Tivoli Netcool Configuration Manager.

- *IBM Tivoli Netcool Configuration Manager Release Notes*

  Gives important and late-breaking information about IBM Tivoli Netcool Configuration Manager. This publication is for deployers and administrators, and should be read first.

Prerequisite publications: IBM Tivoli Network Manager IP Edition

To use the information in this publication effectively when dealing with an integrated installation of Netcool Configuration Manager, Network Manager, and Tivoli Netcool/OMNIbus, you must have some prerequisite knowledge, which you can obtain from the Network Manager documentation, especially the following publications:

- *IBM Tivoli Network Manager IP Edition Release Notes*

  Gives important and late-breaking information about IBM Tivoli Network Manager IP Edition. This publication is for deployers and administrators, and should be read first.

- *IBM Tivoli Network Manager Getting Started Guide*

  Describes how to set up IBM Tivoli Network Manager IP Edition after you have installed the product. This guide describes how to start the product, make sure it is running correctly, and discover the network. Getting a good network discovery is central to using Network Manager IP Edition successfully. This guide describes how to configure and monitor a first discovery, verify the results of the discovery, configure a production discovery, and how to keep the network topology up to date. Once you have an up-to-date network topology, this guide describes how to make the network topology available to Network Operators, and how to monitor the network. The essential tasks are covered in this short guide, with references to the more detailed, optional, or advanced tasks and reference material in the rest of the documentation set.

- *IBM Tivoli Network Manager IP Edition Product Overview*

  Gives an overview of IBM Tivoli Network Manager IP Edition. It describes the product architecture, components and functionality. This publication is for anyone interested in IBM Tivoli Network Manager IP Edition.

- *IBM Tivoli Network Manager IP Edition Installation and Configuration Guide*

  Describes how to install IBM Tivoli Network Manager IP Edition. It also describes necessary and optional post-installation configuration tasks. This publication is for administrators who need to install and set up IBM Tivoli Network Manager IP Edition.

- *IBM Tivoli Network Manager IP Edition Administration Guide*

Describes administration tasks for IBM Tivoli Network Manager IP Edition, such as how to administer processes, query databases and start and stop the product. This publication is for administrators who are responsible for the maintenance and availability of IBM Tivoli Network Manager IP Edition.

- *IBM Tivoli Network Manager IP Edition Discovery Guide*

  Describes how to use IBM Tivoli Network Manager IP Edition to discover your network. This publication is for administrators who are responsible for configuring and running network discovery.

- *IBM Tivoli Network Manager IP Edition Event Management Guide*

  Describes how to use IBM Tivoli Network Manager IP Edition to poll network devices, to configure the enrichment of events from network devices, and to manage plug-ins to the Tivoli Netcool/OMNIbus Event Gateway, including configuration of the RCA plug-in for root-cause analysis purposes. This publication is for administrators who are responsible for configuring and running network polling, event enrichment, root-cause analysis, and Event Gateway plug-ins.

- *IBM Tivoli Network Manager IP Edition Network Troubleshooting Guide*

  Describes how to use IBM Tivoli Network Manager IP Edition to troubleshoot network problems identified by the product. This publication is for network operators who are responsible for identifying or resolving network problems.

- *IBM Tivoli Network Manager IP Edition Network Visualization Setup Guide*

  Describes how to configure the IBM Tivoli Network Manager IP Edition network visualization tools to give your network operators a customized working environment. This publication is for product administrators or team leaders who are responsible for facilitating the work of network operators.

- *IBM Tivoli Network Manager IP Edition Management Database Reference*

  Describes the schemas of the component databases in IBM Tivoli Network Manager IP Edition. This publication is for advanced users who need to query the component databases directly.

- *IBM Tivoli Network Manager IP Edition Topology Database Reference*

  Describes the schemas of the database used for storing topology data in IBM Tivoli Network Manager IP Edition. This publication is for advanced users who need to query the topology database directly.

- *IBM Tivoli Network Manager IP Edition Language Reference*

  Describes the system languages used by IBM Tivoli Network Manager IP Edition, such as the Stitcher language, and the Object Query Language. This publication is for advanced users who need to customize the operation of IBM Tivoli Network Manager IP Edition.

- *IBM Tivoli Network Manager IP Edition Perl API Guide*

  Describes the Perl modules that allow developers to write custom applications that interact with the IBM Tivoli Network Manager IP Edition. Examples of custom applications that developers can write include Polling and Discovery Agents. This publication is for advanced Perl developers who need to write such custom applications.

- *IBM Tivoli Monitoring for Tivoli Network Manager IP User's Guide*

  Provides information about installing and using IBM Tivoli Monitoring for IBM Tivoli Network Manager IP Edition. This publication is for system administrators who install and use IBM Tivoli Monitoring for IBM Tivoli Network Manager IP Edition to monitor and manage IBM Tivoli Network Manager IP Edition resources.

Prerequisite publications: IBM Tivoli Netcool/OMNIbus

To use the information in this publication effectively when dealing with an integrated installation of Netcool Configuration Manager, Network Manager, and Tivoli Netcool/OMNIbus, you must have some prerequisite knowledge, which you can obtain from the Tivoli Netcool/OMNIbus documentation, especially the following publications:

- *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*

Includes installation and upgrade procedures for Tivoli Netcool/OMNIbus, and describes how to configure security and component communications. The publication also includes examples of Tivoli Netcool/OMNIbus architectures and describes how to implement them.

- *IBM Tivoli Netcool/OMNIbus User's Guide*

  Provides an overview of the desktop tools and describes the operator tasks related to event management using these tools.

- *IBM Tivoli Netcool/OMNIbus Administration Guide*

  Describes how to perform administrative tasks using the Tivoli Netcool/OMNIbus Administrator GUI, command-line tools, and process control. The publication also contains descriptions and examples of ObjectServer SQL syntax and automations.

- *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*

  Contains introductory and reference information about probes and gateways, including probe rules file syntax and gateway commands.

- *IBM Tivoli Netcool/OMNIbus Web GUI Administration and User's Guide*

  Describes how to perform administrative and event visualization tasks using the Tivoli Netcool/OMNIbus Web GUI.

Accessing terminology online

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

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

Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center website at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp

Note: If you print PDF documents on 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.

Ordering publications

You can order many Tivoli publications online at the following website:

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 the following website:

   http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss
2. Select your country from the list and click **Go**. The **Welcome to the IBM Publications Center** page is displayed for your country.
3. On the left side of the page, click **About this site** 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.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education website:

https://www.ibm.com/training/search?query=tivoli

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online
Go to the IBM Software Support site at http://www.ibm.com/software/support/probsub.html and follow the instructions.

IBM Support Assistant
The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related information and serviceability tools for problem determination. To install the ISA software, go to http://www.ibm.com/software/support/isa

Conventions used in this publication

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

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. For example, on UNIX systems, the $NCHOME environment variable specifies the directory where the Network Manager core components are installed. On Windows systems, the same environment variable is %NCHOME%. 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.

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

Chapter 1. The Application Programming Interface

This section is intended for anyone who uses ITNCM to manage and configure network resources. This section is intended for anyone who needs to use the Netcool Configuration Manager Java API to access Netcool Configuration Manager functionality, or who needs to understand the high level design and functionality of the API.

Required Knowledge

In order to effectively use the information in this guide, you need to have an understanding of the Netcool Configuration Manager functionality and concepts. You should have at least a basic understanding of Java or a similar programming language. An understanding of high level Java concepts such as jar files, Java packages, and system properties is also recommended. An understanding of configuring network devices is not required, but may be helpful when actually designing an application using this API.

API overview

The Application Programming Interface (API) provides several Java interfaces to allow the API user to interact with the Netcool Configuration Manager in much the same way that a user interacts with Netcool Configuration Manager through the GUI.

Restriction: If you want to use the Sun JRE instead of the IBM JRE for the JVM used by the client API application, download the following files and add them to the classpath:

xalan.jar

You can obtain the xalan.jar file from http://www.apache.org, or another external source.

As of the latest versions of Xalan the packaging has changed and some parts have been moved into separate jars. You must obtain these and add them to the classpath as well:

- serializer.jar
- xercesImpl.jar
- xml-apis.jar

Note: The API is not thread safe.

An API user must login to the server prior to accessing any services provided through the API. After the user has established an API session, the user can perform most of the same activities that are available through the Netcool Configuration Manager GUI. For example, auser can search for and retrieve resources from the server, access resource attributes, make changes to network resource configurations by creating and submitting command set UOWs (Unit Of Work), check the status of UOWs, and view UOW logs.

A few activities performed through the Netcool Configuration Manager GUI are not practical using the API. In particular, making ad-hoc configuration changes to network resource configurations as is done from the Configuration Editor, and creating modeled command sets of any significant complexity require extensive knowledge of the underlying device schema and markup language used internally to model device configurations and configuration changes. Since the device schemas and command set markup language are not currently supported for external use and are subject to change without notice, this approach is not recommend even for experienced XML users/programmers.

The API also provides capabilities that are not available from the GUI. For example, it is possible to programmatically "assemble" command sets together at run time to create a single command set that is applied to the network resource as an atomic change.

The API does not provide the following capabilities:

- Direct access to the database
- Direct access to Worker Servers

- Classes, interfaces, or method calls to directly manipulate or mark up the internal XML documents stored in the various types of resources used by the ITNCM - Base.

The XML documents contained in ITNCM - Base resources can be retrieved or set on those resources, but any editing of the XML documents is outside of the scope of the API. Individuals with adequate knowledge of the XML schema and markup language used internally by ITNCM - Base can attempt to change the XML documents externally, but this approach is not supported as part of normal API usage, and is not generally supported.

Architecture

Starting with Netcool Configuration Manager version 6.2, the API communicates with the Netcool Configuration Manager - Base server using an XML over HTTP/S serialization mechanism that has similar behavior to a Java RMI interface, but avoids J2EE vendor-specific library dependencies.

API functionality that does not require direct access to network resources merely accesses or manipulates data in the underlying database. Functionality that requires access to network resources (including requesting current data from a network resource, and changing the configuration on a network resource) results in data being placed in the database that will result in the Worker Servers using that data to interact with the network resources. With one exception, the presentation server does not interact directly with network resources as the result of API method calls. (The only notable exception to this approach is the use "direct commands", in which case the Presentation server executes commands against network resources. See the caveats listed later in this document regarding using this approach).

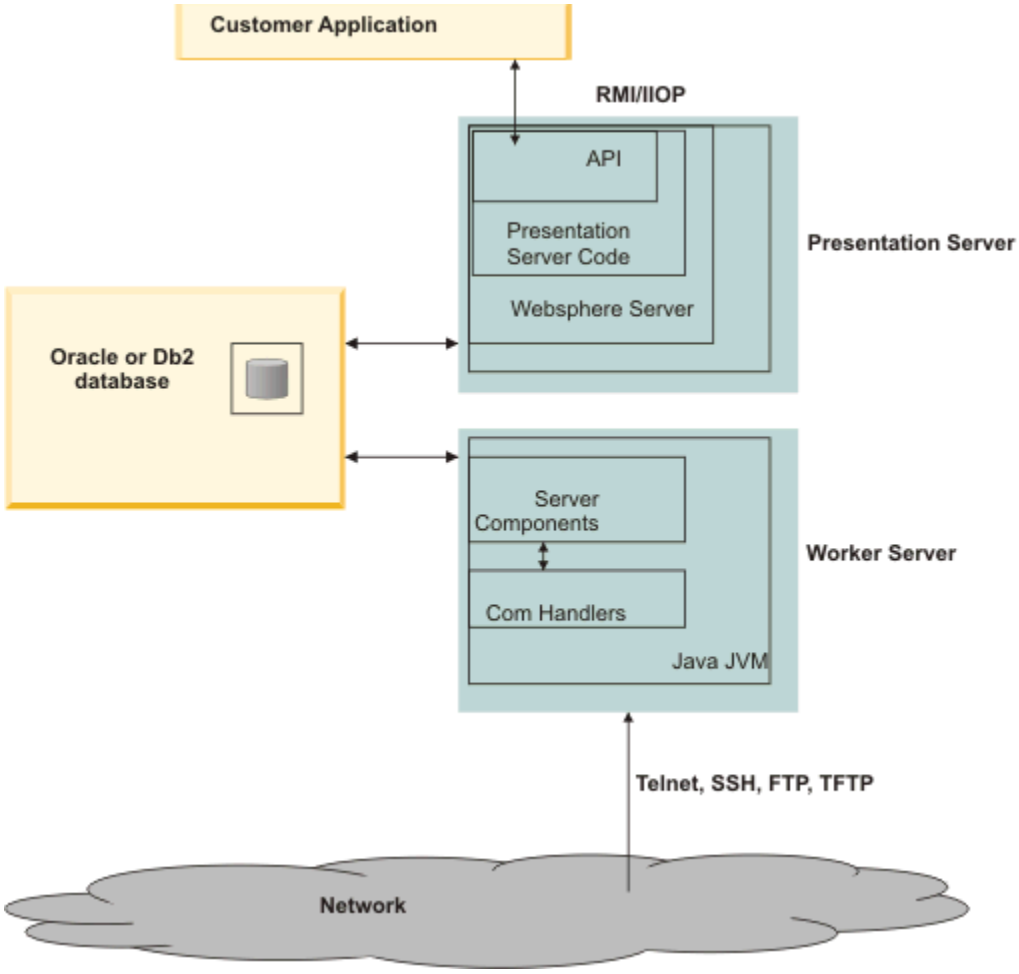

Figure 1. API architecture

Interfaces in the com.intelliden.icos.idc package represent data stored in the underlying database including data regarding network resources, configurations associated with network resources, UOWs, and other Netcool Configuration Manager resources. These interfaces are implemented by classes used in the presentation server and worker server code. The interface representation of the data may or may not be representative of how the is actually stored in the database in any particular Netcool Configuration Manager version.

- Attributes and methods in the implementing classes are exposed only as they are deemed to useful and safe to use from the API.
- The API is generally structured to allow API users to create powerful solutions that may require in-depth manipulation of Netcool Configuration Manager resource data. In some cases many of the underlying "setter" methods are exposed that allow users to attempt to set inaccurate data on resources, such as "last modified time" or "last modified by" attributes. These "setter" methods should be used with caution to avoid introducing inaccurate data into the Netcool Configuration Manager system. In many cases, the Worker Servers may overwrite such user-supplied data as they process work.

ITNCM Internal Workflow, UOWs, and the API

ITCM uses an internal workflow to process work that requires actual communications with network resources (network devices). The API provides methods to submit work into the internal workflow, and to track that work in a similar manner to the ITNCM GUI interface.

If an API method requires access to resource data already stored in the underlying database (for example, the existing stored current configuration for a network resource), then that data is retrieved from or updated in the database in a synchronous manner, and the results of the operation are returned to the user.

If an API method requires an interaction with a network resource, a UOW must be created and submitted to the presentation server from the API. The presentation server does not perform the work directly; the UOW data is entered in the data base and is later acted on by a worker server. If the data is successfully entered in the database, a WorkKey is returned that can be used to check the status of the submitted UOW. If there is a problem with the data that is detected at the time of UOW submission, an exception is returned. Completion time for the UOW depends on the scheduling of the UOW, the backlog of work for the applicable worker servers, and the nature of the UOW. In the case of UOWs submitted for immediate execution, where worker servers are immediately available, the work may be completed in 30 seconds to a minute. Just as with work submitted from the GUI, the nature of the work and the number of resources involved in the UOW can greatly affect the completion time. It is possible to submit UOWs that may take over an hour to complete.

The WorkKey returned when submitting a UOW only indicates that the work has been successfully submitted. If the outcome of the UOW is of interest, the user must check on the status of the UOW using the WorkKey. There are methods in the API that allow either registering to receive events when the UOW changes state, or directly requesting the status of the UOW.

It may be necessary to view the UOW log details to determine specific information related to the success or failure of the UOW once it has completed. There are methods in the API that allow either registering to receive events when the UOW changes state, or to directly query the workflow state and execution status of the UOW.

XML over HTTP/S communications protocol

The protocol used by the Java API for communications between the API client and the Netcool Configuration Manager - Base presentation uses XML over HTTP/S, server starting with Netcool Configuration Manager version 6.2. One of the primary reasons for using this communication mechanism is to avoid dependencies on any particular vendor's J2EE libraries as is usually the case when using RMI.

While the underlying details are not exposed to the user and are subject to change without notice, the high level details of the current version ofNetcool Configuration Manager communications are provided here as background information:

- The client and server implementations both use a proprietary mechanism to serialize and de-serialize Java objects to and from XML format. Direct creation or use of the XML data outside of the API interface is not supported.
- The data in XML format is exchanged between the client and the server over HTTP/S. Using HTTPS requires installing a server certificate on the underlying WebSphere Application Server.
- The server side implementation includes a servlet that the API client connects to. Direct use of the servlet outside of the API interface is not supported.
- The server side acts as a proxy for client requests, de-serializing the XML data, carrying out the API calls on the server, and then serializing the return data and returning it to the client.

All the Java classes that are serialized or de-serialized by the client or server implementation must be in the CLASSPATH for that component. In particular, if you observe serialization errors from your API client application, you should check the CLASSPATH you are using for the API client runtime environment. Additional classes may be required beyond what is required to compile the Java code for your client application, since additional classes may be returned as part of the response data from the server. In general, if you are using the recommended CLASSPATH this is only likely to occur when EJBExceptions are returned as part of the embedded stack trace in the case of server side errors when performing the API call. If you wish to see the EJBException data instead of a serialization error, you will need to include a jar file in your CLASSPATH that contains EJBException. In this case, if you are running the API client from a J2EE environment, you will need to use a jar file that is compatible with your J2EE environment.

Package structure

The API is structured as several Java packages that provide functionality generally corresponding to the GUI functionality, as well as the `com.intelliden.icos.idc` package that contains interfaces that represent the data objects used by the ITNCM - Base servers.

Most of the packages contain a "manager" interface that contains the methods that interact with the ITNCM - Base server, as well as a "data factory" class that is used to instantiate classes that implement the interfaces in the com.intelliden.icos.idc package. "Data factories" are used instead of constructors since interfaces cannot be directly instantiated using a constructor.

com.intelliden.icos.api.accounts
Methods to add, delete, or modify users and groups

com.intelliden.icos.api.admin
Methods to pause and resume the ITNCM workflow process on worker servers

Methods to check the running state of some server components

Methods to retrieve and set certain server information similar to what is available from the GUI

com.intelliden.icos.api.resources
Includes a `NetworkResourceManager` interface as well as a `ResourceManager` interface

Methods to add, delete, search for, and modify resources and network resources

Methods to fetch network resource related data such as VTMOS information and configurations

com.intelliden.icos.api.workflow
Methods to create, populate, submit, and track UOWs

com.intelliden.icos.idc
Contains the various "data" interfaces that are manipulated by the "manager" interfaces in the other packages

com.intelliden.icos.api
Includes the `ApiFactory` class and `ApiSession` interface.

- The `ApiFactory` class provides static methods to create an ApiSession
- ApiSession represents a connection to the ITNCM - Base server

- `ApiSession` has methods to create the various "manager" interfaces and data factories.

com.intelliden.icos
Contains the ITNCM-specific exception classes used by the API.

Exceptions in the ITNCM - Base API are subclasses of com.intelliden.common.IntellidenException. The various subclasses are documented in detail in the ITNCM - Base javadocs, and can be explicitly caught if the exectute type of ITNCM - Base exception is of interest in your program However, in many cases it may be adequate to simply catch IntellidenException, particularly if your program does not differentiate handling of the different subclasses. IntellidenException and its subclasses implement the `getNestedStackTrace` method, which is extremely useful in terms of getting accurate and complete server-side error data when requesting support with troubleshooting problems.

com.intelliden.icos.util
Contains utility or general purpose interfaces and classes that may not logically belong in the packages listed above.

API Usage

This section provides a high level overview for some of the most commonly used API procedures. Some examples include connect to server, get manager instances, fetch resources, use resources, and submit and track work.

Note: This section uses some terms, which are discussed later in the terminology section.

Example

A typical API program that applies a command set to a network resource might use the following process:

1. Create an ApiSession that represents the connection to the ITNCM - Base presentation server. This step also validates the user's credentials.
2. Create instances of the "manager" interfaces that are required to perform the program actions. For example, an instance of ResourceManager will be needed to search for and retrieve ITNCM - Base Resources, and an instance of WorkflowManager will be needed to submit and track UOWs. An instance of ResourceDataFactory and WorkflowDataFactory will also be needed to create instances of idc interfaces such as Commands, and CommandWork object.
3. Call methods on the ResourceManager instance to search for and retrieve the ITNCM - Base resources that are needed to perform the program's actions. In this case, one of the overloaded search() methods can be used to return a Collection of ResourceContentDescriptors containing the command set's descriptor, and that descriptor can be used to fetch a copy of the command set. Similarly, one of the search methods can be used to return a Collection of ResourceContentDescriptors containing the descriptor for the NetworkResource, and the NetworkResourceKey can be obtained from the NetworkResources descriptor.
4. Methods can be called on the CommandSet instance to populate any parameter values based on information available to the program.
5. A new CommandSetCommand is created using the ResourceDataFactory instance, and the CommandSet and NetworkResourceKey are added to CommandSetCommand. The various settings available in the GUI "Apply Command Set" wizard can also be applied to the CommandSetCommand. Note that the equivalent of the GUI Command Set editor is not available via the API for Smart Model Command Sets.
6. A new CommandWork (a UOW) object is created using the WorkflowDataFactory instance, and the CommandSetCommand is added to the CommandWork object. Scheduling data is set on the CommandWork object, and it is submitted to the Presentation Server using one of the "submit" or "executeCommand" methods on the WorkflowManager instance. Depending on the method selected, a Command or WorkKey is returned.
7. If a WorkKey is returned, it can be used to track the status of the CommandWork (a UOW) object and to retrieve the UOW once it is finished. If a Command is returned, it may or may not contain data regarding the completed UOW. The Command contains data indicating if the UOW is finished or not. If the UOW is finished, data regarding the completed work can be retrieved from the Command. If the

UOW is not complete, a WorkKey or UOW ID can be derived from data in the Command, and can be used to track the status of the UOW.

8. Track/poll the Work to determine if it is finished. A WorkKey or UOW ID can be used to check on a UOW's work state in the internal ITNCM workflow to determine when the UOW is complete. In this discussion, "completed" or "finished" are used to refer to UOWs that have exited the ITNCM – Base internal workflow. Technically, UOWs have exited the ITNCM - Base workflow when they have been assigned a WorkState of "Finished", "Expired", or "Cancelled". A WorkState of "Finished means that the UOW exited the workflow without manual or programmatic cancellation or dequeuing. A WorkState of "Cancelled" means that the UOW was manually or programmatically cancelled or dequeued. A WorkState of "Expired" means that the UOW did not begin execution before the end of its scheduled execution window passed. In any case, there are two approaches to determining if UOWs have exited the workflow; polling or listening for WorkEvents.

   Polling
   In this approach, one of the "getWork" methods is called on the WorkflowManager instance using a loop or timer mechanism, and the returned Work object is checked after each call until it reaches a WorkState indicating that it has exited the workflow.

   EventListening
   In this approach, the com.intelliden.icos.api.impl.workflow.ItncmEventListener and com.intelliden.icos.api.impl.workflow.ItncmEventFilter interfaces must be implemented by the programmer. The ItncmEventFilter should be implemented to filter out all WorkStates except the targeted WorkStates, and to filter out all event types except UnitOfWorkEvents, and then the "receiveEvents" method is called on the WorkflowManager instance to register to receive ITNCM Events. Depending on your client design, if may also be practical to use the ItncmEventFilter to filter out all UOWs except the one you are interested in. UnitOfWorkEvents are generated for UOW state changes, and contain the WorkState of the UOW causing the UnitOfWorkEvent. UnitOfWorkEvent registration can be cancelled by calling the methods "stopReceivingEvents" or "stopReceivingAllEvents" on the WorkflowManager instance. There are a various strategies that can be used to check for UOW completion using WorkEvents, but the details are beyond the scope of this brief example.

User accounts

The credentials used with the ITNCM - Base API when creating an API session must be the credentials of an ITNCM - Base application user, not the credentials of a user on the AIX or Linux system the Presentation server runs on, nor the credentials used to connect to a network device. User accounts for the API follow the same rules as for the ITNCM - Base GUI regarding permission and resource access in the ITNCM-Base application.

The ITNCM - Base super user should not be used with the API, except for the purpose of the creation of initial or "seed" users and groups. Other use of the ITNCM - Base super user from the API is generally undefined and is likely to fail.

The user account must have the appropriate privileges in terms of functional activities and permissions, in order to accomplish the requested API operations.

"Remote" account authentication should generally work from the API provided it is set up in ITNCM-Base, but may differ slightly in terms of error details available from the GUI.

"Two factor" account authentication is not generally practical from the API, since the password token is likely to be non-resuable and is also likely to have a limited time span of validity. If your program design requests the password and token from the user at run time and promptly uses this data to login to the ITNCM - Base, authentication may work correctly. However, any problems wih the account, any delays in using the login data, or any retries using the same data will typically result in failed authentication and potentially cause problems with the status of the account in the external two-factor authentication system.

Chapter 2. Installing the ITNCM - Base API

All ITNCM - Base core software, API code, and required third-party software are installed on your server during the installation process.

In order to make use of the API, you will need to set up and configure your client machines. This section describes the contents of the ITNCM - Base client installation. This installs the files needed on the client side in order to code, test, and use the ITNCM - Base API.

To install the ITNCM - Base API, complete the following steps:

1. On a server where ITNCM-Base is installed, uncompress the $ITNCM_HOME/dist/ITNCMClient.zip file to a temporary directory.
2. Start IBM Installation Manager.
3. Add the `ITNCMClient/repository.config` file as a local repository.
4. Install the Netcool Configuration Manager Client API feature.

The ITNMClient can also be installed on other servers, allowing remote access to the API interface.

Chapter 3. Setting the CLASSPATH environment variable to execute ITNCM - Base API programs

A subset of the jar files installed by the client installation are required to successfully run ITNCM - Base API programs. This subset of jar files must be included in the CLASSPATH environment variable when using the ITNCM - Base API.

You can also copy this subset of jar files from the ITNCM - Base server, in `/opt/ IBM/tivoli/ netcool/ncm/lib` in a "standard" installation. Depending on the details of the program, not all the jar files may be required. The details of setting up your CLASSPATH environment variable will vary with your development and runtime environment and are not covered here. Note that it is generally a good idea to specifically set your CLASSPATH environment variable, if possible, rather than simply adding to an existing CLASSPATH environment variable to avoid library conflicts.

You should add to the CLASSPATH environment variable all of the jar files that reside in the `lib` directory that are extracted when executing the client installer.

If you are not using the API client in a J2EE environment, or if you are using it in a WebSphere J2EE environment of the same version used by ITNCM-Base, you can also include com.ibm.ws.admin.client_7.0.0.jar in the CLASSPATH to avoid misleading serialization exceptions in the case that an EJBException is returned from the server. If you are running the client from any other J2EE environment, you will need to find a jar file that include EJBException and that is compatible with your J2EE environment.

Chapter 4. Terminology used when working with the Java API

The Java API provides the objects that define the methods you call to operate on generalized resources and network resources and their associated configurations.

The following topics describe the Netcool Configuration Manager-specific terminology used when working with the Java API.

Generalized resources and network resources

ResourceManager methods are used when working with generalized resources and network resources.

Examples of generalized resources include:

- Authentications
- RADs
- File Transfer resources
- Command Sets

Network resources and their configurations have different characteristics and behavior than generalized resources in some respects, and NetworkResourceManager is used in those cases.

Historically, NetworkResourceManager methods were typically used when working with network resources and configurations, but many of the NetworkResourceManager methods have been deprecated with similar functionality provided by ResourceManager. For compatibility with future ITNCM - Base releases, you should avoid using these deprecated methods, even if the methods in ResourceManager have slightly different (usually more generic) behavior.

NetworkResourceManager continues to provide some methods that are specific to configurations, network resources, and VTMOS related operations.

Work (UOW)

UOWs (subclasses of Work) are not a subclass of Resource, and are not considered to be resources for API purposes.

The high level use and behavior of UOWs is covered in the ITNCM - Base documentation, and is not repeated as part of this API documentation. However, a basic understanding of UOWs is useful for understanding how they are used from the API.

Since Netcool Configuration Manager version 5.0, all UOWs are either instances of CommandWork, or are converted to CommandWork objects by the API at the time they are submitted to the Netcool Configuration Manager - Base server.

Legacy UOWs

Legacy UOWs are direct subclasses of Work that are not CommandWork types, and are still exposed in the API for the purposes of backwards compatibility with OOBC.

Some of these Work subclasses are converted to equivalent CommandWork UOWs at the time they are submitted to Netcool Configuration Manager - Base. These UOW types are ImportNetworkResourceWork, CommandSetWork, and ConfigurationSynchronizationWork.

All other "legacy" Work subclasses typically result in a CommandWork containing a null Command if they are submitted from the API. None of the "legacy" Work types should be used for any new API work; the CommandWork interface should always be used.

CommandWork object

The CommandWork object acts as a scheduling envelope for Commands submitted to the Netcool Configuration Manager - Base workflow engine.

Although "legacy" attributes (from Work) still exist in CommandWork objects, only the attributes related to scheduling and comment data should be set on CommandWork objects using the API. These are Scheduled start and end times, Recurring Schedule data and Work comment (shows up as the UOW description in the GUI). The Work comments attribute can also be set on the Command contained by CommandWork. If this attribute is set on both objects, it is recommended that the same text is used for both instances to avoid confusion regarding which instance is used by the API in different situations. Workstate, execution status, and actual start and end time data will be populated by the Netcool Configuration Manager - Base server(s) as the UOW is processed.

CommandWork objects can contain only one Command. Some attributes regarding worker server names and IDs, and log file names and locations are maintained for compatibility with earlier Netcool Configuration Manager - Base versions, however where this data is no longer applicable its value is set to null.

Commands

Commands contain the details of the actual work to be done.

Examples of Commands are ImportNetworkResourceCommand, CommandSetCommand, NativeCommandSetCommand, ConfigurationSynchronizationCommand, and AutodiscoveryCommand.

Most Commands can contain multiple NetworkResourceKeys, or NetworkResourceCriterion with the NetworkResource name wildcarded so that the Command acts on multiple NetworkResources.

The various settings such as rollback, report-only, and authentication override behavior that can be set from the GUI when submitting work can also be set on Commands from the API.

Workstates

The workstates for UOWs being processed by the Netcool Configuration Manager - Base server are the same in the API and the GUI.

The following list identifies these workstates:

- Pending_Approval
- Ready_For_Execution
- Executing
- Finished
- Expired (Work did not start prior to its execution window passing.)
- Cancelled (Work has been de-queued or was rejected instead of approved.)

Note: In the case of listening for Netcool Configuration Manager events and checking for the Workstate the Event represents, use the com.ibm.tivoli.itncm.events.api.enums.WorkState enumeration rather than the com.intelliden.icos.idc.WorkState interface.

Execution status

Once work has reached the Finished, Expired or Cancelled state, its Execution status can be retrieved.

Prior to that WorkState, the execution status is null. The execution status corresponds to the "Execution Status" GUI, and uses the same values: SUCCESS, FAILURE, PARTIAL_SUCCESS, CANCELLED, REJECTED and N/A.

Work log data

The Work log data can be retrieved via the API using the WorkKey or UOW ID as a parameter to the appropriate call on WorkflowManager.

The log data is the same as is available from the GUI, although the ordering of the data is slightly different. The API does not provide any interfaces to parse or otherwise interpret log data after it is retrieved.

Keys, descriptors, and criteria

Keys, descriptors, and criteria are used frequently within the ITNCM - Base API, so it is useful to understand what these terms mean within the context of the ITNCM - Base API.

The following topics promote an understanding of these terms.

Keys

Keys are used to uniquely identify an object in the Netcool Configuration Manager - Base data store, and can be used to retrieve that object or in some cases to retrieve specific data about the object.

In actual practice, ResourceDescriptors (described in detail later in the document), are used more frequently in the API to search for and fetch Netcool Configuration Manager - Base resources. Keys are still frequently used to fetch ResourceDescriptors. Examples of keys are: NetworkResourceKey, ConfigurationKey, CommandSetKey, RealmKey or ResourceContainerKey.

From a practical viewpoint, keys fall into the following general categories:

- ResourceContentKeys that represent any type of resource except a realm or a UOW.
- ResourceContainerKeys that represent realms.
- WorkKeys that represent UOWs.

Keys of types that represent actual Netcool Configuration Manager - Base data typically have two methods used to return information that can be used to identify the resource, and that have potentially misleading names for reasons related to backward compatibility with earlier versions of the Netcool Configuration Manager - Base API, where the Netcool Configuration Manager - Base used an LDAP directory as a data store. These two methods are:

- `getDistinguishedName` — This typically returns the unique key for the resource in some table in the underlying database.
- `getFriendlyname` — This typically returns the name by which the resource is known to GUI users. Or it can also return the ID for a UOW, the name of a NetworkResource or the name of a generic resource such as an authentication.

Typically, a key can be stored as a string and the appropriate data factory can then be used to create a Key from that string that can be used as a parameter in API methods. The `toString` method for a key typically returns the same string as the `getDistinguishedName` method. As an example, when tracking the state of a UOW, a WorkKey can be created by calling the `new WorkKey(String dn)` constructor on WorkflowDataFactory and passing in the String that was returned from the `toString` or `getDistinguishedName` method on the original WorkKey. The new WorkKey can then be used to fetch the UOW (Work) object by calling `getWork(WorkKey key)`.

Note: It is also possible in this case to fetch the UOW by its UOW ID, obtained by calling the `getFriendlyname` method on the original WorkKey. In current Netcool Configuration Manager versions, the API typically provides methods to fetch UOWs based on their UOW ID instead of their Key, so it may be simpler to persist the UOW ID than the `toString` value of the WorkKey for the UOW.

Descriptors

Descriptors contain metadata or high level data about ITNCM - Base Resources and are frequently used as parameters for method calls that perform search, fetch, move, rename, and delete operations.

Resource information typically includes Resource name, Resource key, Data type, Realm the resource is located in, Created by and last modified by, Created date and last modified date and VTMOS data when applicable. The following list identifies two general categories of ResourceDescriptors:

- ResourceContentDescriptors — Refer to generalized resources and NetworkResources.
- ResourceContainerDescriptors — Refer to Realms. ResourceContainerDescriptors contain lists of ResourceContainerDescriptors and ResourceContentDescriptors; these are the child Descriptors that refer to Resources in the Realm. Typically when a ResourceContainerDescriptor is initially retrieved from the ITNCM - Base server, these lists may be sparsely populated (or may contain no data at all).

Calling any of the overloaded `GetContainerOnly` methods when fetching a ResourceContainerDescriptor will insure that no child Descriptors are returned. Calling `getContainer( ResourceContainerDescriptor descriptor)` will return an instance of the passed in ResourceContainerDescriptor with the child Descriptor list populated.

VtmosCriterion and NetworkResourceCriterion

The `VtmosCriterion` and `NetworkResourceCriterion` interfaces specify "filter" data that can be used to search for a Resource in the underlying database, or that can be used in VTMOS dependent operations such as converting CLI to XML.

A `VtmosCriterion` interface contains data regarding: Vendor, Type, Model, OS Version, Netcool Configuration Manager - Base driver UUID and any values that are null are generally considered wild-card values.

Note that NetworkResourceCriterion is a sub-interface of VtmosCriterion, and can also include the name of the NetworkResource, the realm to start any searches from that involve this NetworkResourceCriterion, and whether to search subrealms of the starting realm.

Examples of key, descriptor, and criterion usage

The following examples illustrate the use of key, descriptor, and criterion usage in the base API.

The credentials used with the ITNCM - Base API when creating an API session must be the credentials of an ITNCM - Base application user, not the credentials of a user on the AIX or Linux system the Presentation server runs on. Neither can they be the credentials used to connect to a network device.

Search for a filtered list of resources

The following example shows the definition of the `ResourceManager.search` method.

```
ResourceManager.search(ResourceContainerDescriptor startingContainer,
                       java.lang.String resourceType,
                       java.lang.String name,
                       VtmosCriterion vtmosCriterion);
```

The `ResourceManager.search` method searches for a filtered list of resources starting from a specified Realm, and returns a Collection of ResourceContentDescriptors or ResourceContainerDescriptors matching the VtmosCriterion and specified Resource Type. The *name* and *vtmosCriterion* parameters may be null to indicate wild card values. If a Resource type of Resource.RESOURCE_CONTAINER is specified, then ResourceContainerDescriptors will be returned; for any other Resource type, ResourceContentDescriptors will be returned.

Retrieve a resource given a ResourceDescriptor

The following example shows the definition of the `ResourceManager.getResource` method.

```
ResourceManager.getResource(ResourceContentDescriptor resourceDescriptor);
```

The `ResourceManager.getResource` method retrieves a Resource using the provided ResourceDescriptor. Since a ResourceContentDescriptor is passed in as a parameter, a generalized resource or a NetworkResource will be returned (not a Realm). The return type is Resource, so the resource will need to be cast to its specific type prior to making type-specific calls against it.

Get a ResourceContentDescriptor

The following example shows the definition of the `ResourceManager.getResourceContentDescriptor` method.

```
ResourceManager.getResourceContentDescriptor(ResourceContentKey key);
```

The `ResourceManager.getResourceContentDescriptor` method gets the ResourceContentDescriptor associated with the passed in ResourceContentKey. A ResourceContentKey can represent a generalized resource or NetworkResource, but not a Realm. Since the current API typically uses descriptors instead of keys, this method can be used to obtain a descriptor if you currently only have a key for the Resource.

Convert ITNCM - Base XML configuration data

The following example shows the definition of the `NetworkResourceManager.convertXMLToResourceCommands` method.

```
NetworkResourceManager.convertXMLToResourceCommands(VtmosCriterion criterion,
                                                    java.lang.String commands,
                                                    boolean full);
```

The `NetworkResourceManager.convertXMLToResourceCommands` method converts ITNCM - Base XML configuration data into native CLI using the passed in VtmosCriterion to determine which XML to CLI translator to use.

Chapter 5. Netcool Configuration Manager - Base API Examples

The Netcool Configuration Manager - Base API provides the Java interfaces that contain the methods you call to perform a variety of useful tasks associated with interacting with the Netcool Configuration Manager - Base server.

Use these examples as a guide to creating your own applications using the Netcool Configuration Manager - Base API.

Logging in

The logging in example shows how to establish an API session using the `APIFactory` class.

The logging in example shows how to call the `getSession` method that is defined by the `APIFactory` class. The `getSession` method takes the user credentials (username and password) and the server name and port.

```
String username;
String password;
String server;
int port;
.
.
.
ApiSession session = null;
try {
//it is assumed that the argument values have been set
session = ApiFactory.getSession(username, password, server, port);
} catch (IcosException ie) {
ie.printStackTrace();
System.err.println(ie.getNestedExceptionStackTrace());
} catch (Exception e) {
e.printStackTrace();
}
if (
session == null) {
System.out.println("Login failed!");
System.exit(1);
}
```

Logging out

The logging out example shows how to clean up client API sessions using the `disconnect` method.

The logging out example shows how to call the `disconnect` method, which performs the tasks necessary to clean up client API sessions. As the example shows, the `disconnect` method takes no parameters.

```
ApiSession session;
.
.
.
try {
session.disconnect();
} catch (Exception ignore) {}
```

Finding a realm descriptor and key by name

The finding a realm descriptor and key by name example shows how to find a realm.

Each realm has a unique realm path, much like a folder or directory in an operating system. For example, the `NCM/realmOne` path indicates that `NCM` is a realm and `realmOne` is a subrealm (subfolder) of `NCM`. This path can be used to find a realm (also known as a container). For various methods, you may

need either a descriptor or a key. Should you find a need to have a RealmKey object, rather than a ResourceContainerKey, use the getKeyAsRealmKey method on the descriptor. Also, you can use the getRootContainer convenience method in ResourceManager to retrieve the root realm descriptor.

```
ResourceContainerDescriptor realmRCD = null;
ResourceContainerKey realmKey = null;
try {
realmRCD = rMgr.getContainer(realmPath);
} catch (IcosException ie) {
System.out.println("Caught IcosException getting realm");
ie.printStackTrace();
System.err.println(ie.getNestedExceptionStackTrace());
} catch (Exception e) {
System.out.println("Caught Exception getting realm");
e.printStackTrace();
}
if (realmRCD == null) {
//handle condition
} else {
realmKey = realmRCD.getKey();
}
```

Finding and retrieving resources by name

The finding and retrieving resources by name example shows how to find a specific resource type, in this case a CommandSet. Use this example to find a CommandSet or any other resource type.

In this example, the search operation is done based on the name of the resource. In general, a search could find more than one instance matching the search criteria, in which case a Collection is returned. For example, CommandSets with the same name may exist in multiple realms. Searching on a common parent of those realms will return all matching resources. (NetworkRescources are the exception in that the name must be globally unique.) Also, the name can include wildcard characters, such as "cmdset*". It is highly recommended to use the static fields on the Resource class to define the resource types.

```
ApiSession session;
ResourceManager rMgr;
ResourceContainerKey containerKey;//realm to search the search
boolean subrealmFlag;//whether to include subrealms
String resourceName;//name of resource to find
.
.
.
//it is assumed the session has been established
rMgr = session.resourceManager();
.
.
.
//it is assumed that the search values have been set
ResourceSearch rs = rMgr.dataFactory().newResourceSearch();
rs.setCriterion(null);
rs.setResourceName(resourceName);
ArrayList<String> rTypeAL = new ArrayList<String>();
rTypeAL.add(Resource.RESOURCE_COMMANDSET); //set resource type(s) as needed
rs.setResourceTypes(rTypeAL);
rs.setStartingContainer(containerKey);
rs.setSubRealms(subrealmFlag);
//now do the search
Collection<ResourceContentDescriptor> csDescs = rMgr.search(rs);
if (csDescs == null || csDescs.isEmpty()) {
//handle case of no matches
} else {
//for each descriptor, retrieve the corresponding CommandSet object
ArrayList<CommandSet> csColl = new ArrayList<CommandSet>();
Iterator csDescIter = csDescs.iterator();
while (csDescIter.hasNext()) {
//retrieve the CommandSet for this descriptor
ResourceContentDescriptor csDesc =
(ResourceContentDescriptor)csDescIter.next();
   csColl.add((CommandSet)rMgr.getResource(csDesc));
  }
}
```

Finding and retrieving resources by VTMOS

The finding and retrieving resources by VTMOS example shows how to find a specific resource type, in this case a CommandSet. Use this example to find a CommandSet or any other resource type.

In this example, the search operation is done based on the VTMOS of the resource, for example, `Cisco/Router/*/*`. You can also initiate a search that combines the VTMOS and name criteria associated with a specific resource.

```
ApiSession session;
ResourceManager rMgr;
ResourceContainerKey containerKey;//realm to search the search
boolean subrealmFlag;//whether to include subrealms
String vendor;//vendor search filter
String type;//type search filter
String model;//model search filter
String os;//os search filter
.
.
.
//it is assumed the session has been established
rMgr = session.resourceManager();
.
.
.
//it is assumed that search values have been set
.
.
.
//set up search criterion based on vendor/type/model/os
//setting a null value is the same as "*"
//for example setVendor(null) means the same as setVendor("*")
VtmosCriterion vtmos = rMgr.dataFactory().newVtmosCriterion();
vtmos.setVendor(vendor);
vtmos.setType(type);
vtmos.setModel(model);
vtmos.setOsVersion(os);

ResourceSearch rs = rMgr.dataFactory().newResourceSearch();
rs.setCriterion(vtmos);
rs.setResourceName(null);
ArrayList<String> rTypeAL = new ArrayList<String>();
rTypeAL.add(Resource.RESOURCE_COMMANDSET); //set resource type(s) as
rs.setResourceTypes(rTypeAL);
rs.setStartingContainer(containerKey);
rs.setSubRealms(subrealmFlag);
//now do the search
Collection<ResourceContentDescriptor> csDescs = rMgr.search(rs);
if (csDescs == null || csDescs.isEmpty()) {
//handle case of no matches
} else {
//for each descriptor, retrieve the corresponding CommandSet object
ArrayList<CommandSet> csColl = new ArrayList<CommandSet>();
Iterator csDescIter = csDescs.iterator();
while (csDescIter.hasNext()) {
//retrieve the CommandSet for this descriptor
ResourceContentDescriptor csDesc = (ResourceContentDescriptor)csDescIter.next();
csColl.add((CommandSet)rMgr.getResource(csDesc));
}
}
```

Finding a specific Unit Of Work

The finding a specific Unit Of Work (UOW) example shows how to find a specific UOW based on the UOWID.

The example shows the declaration of the UOWID as a `String`.

```
ApiSession session;
WorkflowManager wMgr;
String uowid;//the UOWID as a String, not a long
.
.
.
//it is assumed the session has been established
```

```
wMgr = session.workflowManager();
Work work = wMgr.getWork(uowid);
```

Finding Units Of Work by date

The finding Units Of Work (UOWs) example shows how to find UOWs based on a date range.

The date range can be given as explicit dates (as in this example) or relative to the current date/time. The search can be done for all work submitted by the user, or for all work submitted by any user in the current user's groups. Additionally, search filters for the work state can be set, among other criteria.

```
ApiSession session;
ResourceManager rMgr;
WorkflowManager wMgr;
Date startDate;
Date endDate;
.
.
.
//it is assumed the session has been established
rMgr = session.resourceManager();
wMgr = session.workflowManager();
.
.
.
//set up search criterion based on the date range
//it is assumed the date variables have been set
UnitOfWorkSearch uowSearch = rMgr.dataFactory().newUnitOfWorkSearch();
//The WorkState class has fields for each work state and
//a field for all states; it is highly recommended to use them.
//To only search for completed work, for example, use WorkState.FINISHED
uowSearch.setWorkStates(Arrays.asList(WorkState.ALL_STATES));
uowSearch.setAssociation(WorkAssociation.SUBMITTER);
//or if you want to find work from all users in the current user's group
//uowSearch.setAssociation(WorkAssociation.GROUP);
uowSearch.setStartDate(startDate);
uowSearch.setEndDate(endDate);
//now do the search
Collection<Work> workColl = wMgr.getAllWork(uowSearch);
if (workColl == null || workColl.isEmpty()) {
//handle case of no matches
}
```

Retrieving the log data for a Unit of Work

The retrieving the log file for a Unit Of Work (UOW) example shows how to find the log file associated with a specific UOW ID.

The retrieving the log data for a Unit Of Work (UOW) example shows how to fetch the log data associated with a specific UOW. Each UOW that is executed will have a corresponding set of log data stored in Netcool Configuration Manager - Base. The log data will show details of what was done in the course of executing the work. It may contain error messages and other diagnostic information. In this example, the `getWorkLog` method takes as its parameter the return from `getWorkKey`. There is also a `getWorkLog` signature that takes a UOWID of type `String` as its parameter.

```
ApiSession session;
WorkflowManager wMgr;
Work work;
.
.
.
//it is assumed the session has been established
wMgr = session.workflowManager();
//it is assumed that a Work object has been retrieved
try {
byte[] log = wMgr.getWorkLog(work.getWorkKey());
//now do something with it…
} catch (IcosException ie) {
System.err.println("Exception getting Work log");
ie.printStackTrace();
System.err.println(ie.getNestedExceptionStackTrace());
} catch (Exception e) {
```

```
System.err.println("Exception getting Work log");
e.printStackTrace();
}
```

Creating a network resource

The creating a network resource example shows how to create a network resource. Note that this example only creates a “staged” network resource, and that the network resource must later be “imported” prior to managing its configuration.

Use this example as a model for creating a network resource.

```
ApiSession session;
NetworkResourceManager nrMgr;
ResourceContainerKey containerKey;//realm in which to create the NR
String hostname;//hostname (or IP address) of the device
String vendor;//device vendor
String type;//device type
String model;//device model
String os;//device os
.
.
.
//it is assumed the session has been established
nrMgr = session. networkResourceManager ();
.
.
.
//it is assumed that the variables have been set
NetworkResource nr = nrMgr.dataFactory().newNetworkResource();
nr.setHostname(hostname);
nr.setVendor(vendor);
nr.setType(type);
nr.setModel(model);
nr.setActualOsVersion(os);
nr.setRealmKey(containerKey);
nr.validate();
String cfgName = "Blank Configuration";
NetworkResourceKey nrKey = nrMgr.create(containerKey, nr, cfgName);
```

Importing a network resource

The importing a network resource example shows how to import a previously created network resource.

This example builds on the creating a network resource example described in “Creating a network resource” on page 21. Use this example as a model for importing a network resource.

```
ApiSession session;
ResourceManager rMgr;
WorkflowManager wMgr;
Date startDate;//execution window start date/time
Date endDate;//execution window end date/time
NetworkResourceKey nrKey;
Boolean overrideFlag;//whether to override conflicts
.
.
.
//it is assumed the session has been established
rMgr = session.resourceManager();
wMgr = session.workflowManager();
.
.
.
//it is assumed the startDate and endDate defining the execution window
//for the work have been set
//a UOW requires some comment describing the work
String comment = "Re-Import Network Resource(s)";
//instantiate and populate a command
ImportConfigurationCommand cmd = rMgr.dataFactory().newImportConfigurationCommand();
//this command operates on a Collection of NetworkResourceKeys,
//even though you may only have one of them
//it is assumed the key has been established
ArrayList nrKeys = new ArrayList();
nrKeys.add(nrKey);
```

```
cmd.setKeys(nrKeys);
cmd.setComment(comment);
cmd.setMaxNumberOfErrors(-1);
cmd.validate();

//now instantiate and populate the Work object to submit
CommandWork work = wMgr.dataFactory().newCommandWork();
work.setCommand(cmd);
work.setExecutionWindowStartTime(startDate);
work.setExecutionWindowEndTime(endDate);
work.validate();
//finally submit the work
WorkKey workKey = wMgr.submit(work, overrideFlag);
```

Retrieving state/status for a Unit of Work

The retrieving state/status for a Unit Of Work (UOW) example shows how to retrieve the state or status for a UOW.

Previous examples illustrated how to find the Work object for a UOW based on the UOWID or date. When a UOW is submitted, you will have a WorkKey returned. This too can be used to fetch the Work. From the Work, you can determine its state and status. Putting this in a loop provides a simple polling mechanism to wait for a UOW to finish execution.

Note: If you are polling, you need to retrieve the Work object each time you check the state.

```
ApiSession session;
WorkflowManager wMgr;
WorkKey workKey;
.
.
.
//it is assumed the session has been established
wMgr = session.workflowManager();
.
.
.
//it is assumed the work key been set
Work work = wMgr.getWork(workKey);
WorkState state = work.getState();
if (state.equals(WorkState.FINISHED) ||
state.equals(WorkState.CANCELLED) ||
state.equals(WorkState.EXPIRED)) {
//work is finished
String status = work.getExecutionStatus();
} else {
//work is not finished
}
```

Discovering a device

The discovering a device example shows how to discover a device as a network resource in Netcool Configuration Manager - Base.

If you know the VTMOS for a device, creating a Network Resource with that VTMOS and then importing it is the most efficient way to pull the configuration into Netcool Configuration Manager - Base. In cases where you do not know the VTMOS, the Discovery feature can be used.

```
ApiSession session;
ResourceManager rMgr;
WorkflowManager wMgr;
Date startDate;//execution window start date/time
Date endDate;//execution window end date/time
String hostname;
.
.
.
//it is assumed the session has been established
rMgr = session.resourceManager();
wMgr = session.workflowManager();
.
```

```
.
.
//it is assumed the startDate and endDate defining the execution window
//for the work have been set
//a UOW requires some comment describing the work
String comment = "Discover a device";
//instantiate and populate a command
AutoDiscoveryCommand cmd = rMgr.dataFactory().newAutoDiscoveryCommand ();
cmd.setDeviceName(hostname);
//set next flag to false to discover but not import the configuration
cmd.setImportResources(true);
cmd.setRealmKey(realmKey);
cmd.setComment(comment);
cmd.setMaxNumberOfErrors(-1);
cmd.validate();
//now instantiate and populate the Work object to submit
CommandWork work = wMgr.dataFactory().newCommandWork();
work.setCommand(cmd);
work.setExecutionWindowStartTime(startDate);
work.setExecutionWindowEndTime(endDate);
work.validate();
//finally submit the work
WorkKey workKey = wMgr.submit(work, false);
```

Retrieving the current configuration for a device

The retrieving the current configuration for a device example shows how to retrieve the current configuration for a device.

Once a device configuration has been successfully imported, it can be retrieved in either the native format or XML format from the corresponding NetworkResource.

```
ApiSession session;
ResourceManager rMgr;
NetworkResourceKey nrKey;
.
.
.
//it is assumed the session has been established
rMgr = session. resourceManager ();
.
.
.
//it is assumed that nrKey has been set
//(perhaps from an earlier create)
//we will need the NetworkResource object;
//to get it we need the descriptor
ResourceContentDescriptor nrRCD =
(ResourceContentDescriptor)rMgr.getResourceDescriptor(nrKey);
//now we can get the NetworkResource
NetworkResource nr = (NetworkResource)rMgr.getResource(nrRCD);
//next we get the key of the current configuration
ConfigurationKey cfgKey = nr.getCurrentConfigurationKey();
//with the key we can get the descriptor
ResourceContentDescriptor cfgRCD =
(ResourceContentDescriptor)rMgr.getResourceDescriptor(cfgKey);
//and from the descriptor we can get the Configuration object
Configuration config = (Configuration)rMgr.getResource(cfgRCD);
//finally we can get the configuration content
String nativeConfig = config. getGlobalConfigurationVersionBlob();
String xmlConfig = config.getXmlString();
```

Applying a native command set to a device

The applying a native command set to a device example shows how to apply a set of device commands in a Native Command Set that do not modify the configuration of a device. For example, the set of device commands might be the "show" commands on a CISCO router.

This type of native command set is known as an “interrogative” native command set, and is intended specifically to only fetch data from the device. The device's configuration is not synchronized back into Netcool Configuration Manager - Base after executing an interrogative native command set. To make

configuration changes to a device with a native command set, you must use a “configuration change” type of native command set. Configuration change native command sets result in the device's configuration being synchronized back into Netcool Configuration Manager - Base if the command set is successfully applied, so that Netcool Configuration Manager - Base will be in sync with the changed configuration on the device. Note that for interrogative native command sets the response data from the device is stored in Netcool Configuration Manager - Base along with the UOW data as “interrogation results” and can be retrieved using the API. For configuration change native command sets the response data from the device is not captured and stored, except parts of detected error strings in UOW logs.

In a previous example, you learned how to submit a Unit Of Work that is scheduled. This example shows how to execute the command synchronously (blocking until the UOW completes or a timeout value is exceeded).

Command Sets and Native Command Sets may contain parameters that must be resolved (parameter values applied) before submitting them and the example shows how to do that as well.

```
ResourceManager rMgr;
WorkflowManager wMgr;
Properties params; //the parameters (keys) and data (values)
NetworkResourceKey nrKey;
NativeCommmandSet ncs;
.
.
.
//it is assumed the session has been established
rMgr = session.resourceManager();
wMgr = session.workflowManager();
.
.
.
//it is assumed that the NetworkResourceKey has been populated
 // and an interrorgative NativeCommandSet has been fetched
try {
    //it is assumed that params has been populated
    //skip this if you don&apos;t have any parameters
    ncs.applyParameters(params);

    //instantiate and populate a command
NativeCommandSetCommand cmd = rMgr.dataFactory().newNativeCommandSetCommand();
//this command operates on a Collection of NetworkResourceKeys,
    //even though you may only have one of them
    //it is assumed the key has been established
    ArrayList nrKeys = new ArrayList();
    nrKeys.add(nrKey);
    cmd.setKeys(nrKeys);
    cmd.addCommandSet(ncs);
    cmd.setComment("Apply NCS");
    cmd.setContinueOnError(true);
    cmd.validate();
    //now execute the commands, assuming a timeout condition will not occur
    NativeCommandSetCommand resultCmd =
(NativeCommandSetCommand)wMgr.executeCommandBlocking(cmd, 922337203685477L, true);
Collection<InterrogationResult> iResults = resultCmd.getInterrogationResults();
for(InterrogationResult iResult : iResults) {
            String resultStr = iResult.getInterrogationResult();
            //now examine the command output
            System.out.println("NCS interrogation result: " + resultStr);
    }
}catch (IcosException ie) {
                System.out.println(ie.getNestedExceptionStackTrace());
}
```

Applying an immediate or high priority native command set to a device

The following examples of the executable Soap/Java API Methods describes how to call the new interface methods.

```
Java API Class: WorkFlowManager -> WorkFlowManagerProxy
New Method: executeImmediateNativeCommandBlocking
Parameters: NativeCommandSetCommand command, long timeout, boolean overrideConflicts
```

For our purposes we will be looking at a new method call at this level that will retrieve an action manager and execute against new methods in that class. We will implement new methods down to the Netcool Configuration Manager functional level to ensure that the path of execution is separate to normal workflow. This will ensure that existing code and functionality is not affected by the change.

How to call the new interface method on the Java API

```
NetworkResourceCriterion criterion = resourceDF.newNetworkResourceCriterion();

criterion.setRealmKey(container.getKeyAsRealmKey());
criterion.setHostName(criterionHostnameFilter);
criterion.setVendor(criterionVendorFilter);
criterion.setType(criterionTypeFilter);
criterion.setModel(criterionModelFilter);
criterion.setOsVersion(criterionOsFilter);
criterion.setSubrealms(criterionSubRealms);


NativeCommandSetCommand command = resourceDF.newNativeCommandSetCommand();


command.setCriterion(criterion);
command.addCommandSet(ncs1);
command.addCommandSet(ncs2);
command.setReportOnly(reportOnly);


System.out.println("Executing command.......");
command = (NativeCommandSetCommand) workflowManager.executeImmediateCommandBlocking(
command, 1000000, false);


Collection results = command.getInterrogationResults();
```

How to call the new interface method on the SOAP API

```
Soap API Class: IntellidenRSeriesSoapBindingImpl -> IntellidenRSeries
New Method: executeImmediateNativeCommandBlocking
Parameters: NativeCommandSetCommand command, long timeout, boolean overrideConflicts
```

This new Soap API method has the same interface as the Java API one but the Command object that it expects as an input parameter is actually different and has to be a SOAP version of the object.

```
NetworkResourceCriterion criterion = new NetworkResourceCriterion();


criterion.setRealmKey(criterionKey);
criterion.setHostName(criterionHostnameFilter);
criterion.setVendor(criterionVendorFilter);
criterion.setType(criterionTypeFilter);
criterion.setModel(criterionModelFilter);
criterion.setOsVersion(criterionOsFilter);
criterion.setSubrealms(criterionSubRealms);


NativeCommandSetCommand commandRequest = new NativeCommandSetCommand();
NativeCommandSetCommand commandResponse = null;
```

```
commandRequest.setCriterion(criterion);
commandRequest.setCommandSets(ncs);
commandRequest.setReportOnly(reportOnly);
commandRequest.setProcessingOrder(CommandSetProcessingOrder.COMMAND_AT_A_TIME);


System.out.println("Executing the command......");
commandResponse = (NativeCommandSetCommand)
intelliden.executeImmediateCommandBlocking(
commandRequest, 1000000, false);


InterrogationResult[] results = commandResponse.getInterrogationResults();
```

Parsing results and response handling

Results parsing and response handling is the same as for the existing native command set execution, the new methods are Blocking so they wait for the completion of the UOW and return the response/results as the return object of the call. If the UOW fails the UOW Status within the results will convey this.

Chapter 6. Known workarounds

The ITNCM Base API has some known caveats and workarounds.

The following topics describe these caveats and workarounds.

Using an ApiSession in a Thread Other than the Creating Thread

This workaround describes how to get the "manager" interfaces (for example, ResourceManager) created in a first thread to work in a second thread.

In previous versions of the ITNCM - Base if an ApiSession was created in one thread and then passed to another thread, the "manager" interfaces (such as ResourceManager) that were actually created in the first thread would not work in the second thread. However, if this is found to be a problem the workaround is to pass the credentials to the second thread, and create an ApiSession there instead of passing the ApiSession.

Potential for XML Library Conflicts

This workaround describes how to avoid conflicts between versions of XML libraries.

The ITNCM - Base Java API uses objects that contain XML Documents as well as XML formatted Strings in some cases. Processing of the XML Document occurs in the client JVM for some methods, such as applying parameters to modeled CommandSets. In this situation, the version of XML libraries used by the ITNCM - Base Java API is required to occur first in the CLASSPATH used by the client JVM. This may cause conflicts with XML library versions required by other application APIs that are running in the JVM.

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 grant 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 character set (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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 may 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.

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
958/NH04
IBM Centre, St Leonards
601 Pacific Hwy
St Leonards, NSW, 2069

Australia

IBM Corporation
896471/H128B
76 Upper Ground
London SE1 9PZ
United Kingdom

IBM Corporation
JBF1/SOM1
294 Route 100
Somers, NY, 10589-0100
United States of America

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 measurements 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, ibm.com®, Netcool®, Passport Advantage®, Tivoli®, the Tivoli logo, and WebSphere 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, Portable Document Format (PDF), 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 Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a registered 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.

"Powered by Cryptzone MindTerm" is copyright 1997 – 2017 Cryptzone North America, Inc. All rights reserved.

Other company, product, or service names may be trademarks or service marks of others.

Index

T

V

W

Part Number:

Printed in the Republic of Ireland

(1P) P/N:

2023-4219-01