HP Network Automation

Software Version 9.10

API Reference Guide

Document Release Date: March 2011
Software Release Date: March 2011

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice. 021511

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© Copyright 2011 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.
Java™ is a US trademark of Sun Microsystems, Inc.

Acknowledgements

ANTLR, Apache, Bouncy Castle, GNU, Jaxen, Jython, Netaphor, MetaStuff, Radius, Sleepcat, TanukiSoftware

Documentation Updates

This guide's title page contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.
- To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

  http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Table of Contents

Getting Started

HP Network Automation (NA) is a powerful software solution for network configuration control with sophisticated Web and command-line interfaces for interactive use with NA. The Java, Perl, and SOAP APIs add another dimension to NA by integrating NA with other software. You can link NA to a variety of third-party and custom-built applications, such as ticketing, asset tracking, workflow, change request, and network management software solutions.

This document is intended for network engineering professionals who:

- Write scripts to automate device configuration.
- Are comfortable with basic Java and Perl programming, and have an understanding of database schema and access methods.
- Have knowledge of the NA's Command Line Interface (CLI).
- Integrate various third-party systems with NA version 9.1, such as network management, workflow, and trouble ticketing solutions.

The document includes three chapters and one appendix:

- Chapter 1: Java API
- Chapter 2: Perl API
- Chapter 3: SOAP API
- Appendix A: Installing the Perl API

To view help for the CLI commands, once you are in the CLI, enter `help` to see a list of all the CLI commands. Enter `help <command name>` to see detailed help on a specific CLI command.

Note: For NA 9.1, there is an enhanced Perl API that uses standard Web services (SOAP) for simultaneously accessing NA and functions with multiple NA versions. In addition, the enhanced Perl API enables your scripts to simultaneously make connections to multiple NA servers.

Chapter 1: Java API

Java is a modern, object-oriented language that can run on a variety of platforms. It lends itself to high performance, scalable, highly-available solutions. Java applications are also flexible and easily maintainable. Java is your best choice when using professional development resources, require high performance, and your solution is expected to be in used long term.

Requirements

To use the Java API, you must be running NA 7.20 or higher. The server must be running and accessible to the client where your application runs via port 1099 (Java API). You must also have a valid license on the NA server to use the Java API. A copy of the NA client package is required to write the programs and run the examples.

Note: The server can be bound to a port other than 1099, but in this case, the session API must be explicitly provided with the port number.

Operating Systems

The Java API has been tested with the following operating systems:

- Windows 2000 Professional with Service Pack 2
- Windows 2000 Server with Service Pack 2
- Windows 2003 Server
- Red Hat Linux Enterprise AS (update 2 and 3)
- Solaris 10

Java JDK

Download the Java JDK from Sun Microsystems at *http://java.sun.com/downloads/*

The Java API is tested with Java version 1.4.2 and Java 1.6.

NA JAR Files

You will need two JAR files from the NA server or client installation:

- truecontrol-client.jar
- bcprov-jdk16-141.jar

See the instructions below for details on the location of these files.

Programming Model

The Java API is designed to expose a straightforward programming model with a relatively small set of objects to learn.

Centralized or Distributed Applications

The Java API can be accessed from your NA server machine. This is the simplest programming model.

The Java API also enables you to run applications remotely from the NA server. This means you can run NA on machine A and API-based applications on machine B. In distributed software terminology, this is called remoting. By remoting your application, you will create a client/server application where NA is the server and your application is the client. This might be desirable for load balancing, ease of setting up a development environment, security, or a variety of other reasons.

If you use remoting, you will need to configure your network to allow traffic on port 1099 (Java RMI) to reach the NA server.

Request/Response

The Java API generally follows a request/response model. Your application makes a request via the `exec` method and waits for a response from the server. Details are explained in the “Programming with the Java API” section.

Threading model

The Java API is synchronous on the client side and asynchronous on the server side. This means that when your application makes a request, the calling thread in your application is blocked until a response is available from the server.

This response may mean that your command has been executed and the results returned (such as `list user` command, which immediately returns a list of users), or it may mean that an NA task has been created and queued for future execution (such as the `get snapshot` command, which will schedule a NA snapshot task).

If you want to issue multiple overlapped commands, you will need to use standard Java multi-threading techniques in your application.

Relationship to JDBC

You may notice a strong similarity between the Java API and certain JDBC calls. In particular, NA returns results in a `ResultSet` object derived from JDBC. This makes it easier for developers familiar with JDBC to get up and running with the Java API. Some of the `ResultSet` methods are not applicable to NA, and will return exceptions if used. These are detailed in the “Programming with the Java API” section.

Windows Installation

Installing from CD

This section describes how to install the Java API components from the installation CD.

From the NA installation CD, use either the Server or Client installation options. With either of these options, you will get a copy of:

- The Java Runtime Environment
- The NA client JAR
- Any library JAR files necessary to run the NA SDK

If you are running the SDK application on a machine that already has a NA server installed, no further installation is required. Use the Client install if you will be connecting to a remote NA server for your SDK application.

Note: The file paths referenced in this document assume you installed NA to the default file location, `C:\NA`. If you installed NA to a different location, then replace `C:\NA` with the root directory that you provided at installation.

Setup Java SDK

Install the Java SDK from Sun Microsystems or use the Java JRE that is installed with NA. You need two JAR files to use the Java API:

- <installation directory>/client/truecontrol-client.jar
- <installation directory>/jre/lib/ext/bcprov-jdk14-119.jar

If you are using the Java JRE installed with NA, the bcprov-jdk14-119 jar file is in the Java library path.

If you are using the Java SDK from Sun Microsystems, you must explicitly put the bcprov-jdk14-119jar file in your Java CLASSPATH as described below.

Java API JAR

The Java API JAR is located at `C:\NA\client\truecontrol-client.jar`. However, it may be copied to another location.

Configuration Files

NA and the Java API use several configuration files with an RCX extension. If you use the Java API on the same machine where you installed NA, the RCX files will already be where they need to be (`C:\NA\jre`). If you want to use the API on another machine, you need to manually copy the RCX files to the JRE/JDK directory on the other machine.

The RCX configuration files used by the Java API are:

- messages.rcx
- logging.rcx
- commandlineclient.rcx

The Java API samples are located in `C:\NA\client\sdk\examples\java`.

Java Documentation

The Java API documentation consists of the Javadocs for the API. Javadocs are located in `<install directory>\client\sdk\docs\api.`

Setting Up a Command-line Environment

If you are invoking javac and java from the command line, you can set up a command line environment to use the Java API. Append `truecontrol-client.jar` to your classpath.

Note: Do not put the `truecontrol-client.jar` in jre/lib/ext. NA' Java processes will not start.

Example: (Note that there should be no new line. The example appears to wrap due to documentation margin restrictions.)

```
set CLASSPATH=%CLASSPATH%;<installation directory>\client\truecontrol-client.jar;
<installation directory>\jre\lib\ext\bcprov-jdk16-141.jar
```

To verify that your environment is correct, edit `Example0.java` to set a valid username and password, then compile and run it. Here is what you should see:

```
C:\NA\client\sdk\examples\java>set CLASSPATH=.;c\NA\client\truecontrol-client.jar

C:\NA\client\sdk\examples\java>javac -d . Example0.java

C:\NA\client\sdk\examples\java>java com.rendition.api.examples.Example0
Starting Example0
Session connectivity verified

C:\NA\client\sdk\examples\java>
```

Unix Installation

Installing from CD

This section describes how to install the Java API components from the installer CD.

From the NA installation CD, use either the Server or Client installation options. With either of these options, you will get a copy of:

- The Java Runtime Environment
- The NA client JAR
- Any library JARs files necessary to run the NA SDK

If you are running the SDK application on a machine that already has an NA server installed, no further installation is required. Use the Client install if you will be connecting to a remote NA server for your SDK application.

Note: The file paths referenced in this document assume you installed NA to the default file location, `<install directory>/jre`. If you installed NA to a different location, then replace `<install directory>/jre` with the root directory that you provided at install time.

Java API JAR

The Java API JAR is located at `<install directory>/jre/client/truecontol-client.jar`. Note that it might be copied to another location.

Configuration Files

NA and the Java API use several configuration files with an RCX extension. If you use the Java API on the same machine that you installed NA to, the RCX files will already be where they need to be, in the directory `<installed directory>/jre`. If you want to use the API on another machine, you need to manually copy the RCX files to the JRE/JDK directory on the other machine.

The RCX configuration files used by the Java API are:

- messages.rcx
- logging.rcx
- commandlineclient.rcx

The Java API samples are located in:
`<installed directory>/client/sdk/examples/java`

Unix Installation Troubleshooting

If you see the following error:

```
com.rendition.api.RenditionAPIException: Could not connect to
server:10.101.22.21, LoginException:
javax.naming.CommunicationException [Root
exception is java.rmi.ConnectException: Connection refused to host:
127.0.0.1;
```

Where it says `Connection refused to host: 127.0.0.1`, even though you are connecting to NA on a non-loopback address (10.101.22.21 in this error shown above), Java cannot determine the correct IP address for the host that is running the NA Core.

To correct the issue, either:

- Adjust the hostname resolution so that the hostname of the NA Core host resolves to its correct IP address (typically this would be done by modifying the */etc/hosts* file)

  or

- Add: `wrapper.java.additional.5=-Djava.rmi.server.hostname=<correct-IP-address>` to `$NA/server/ext/wrapper/conf/jboss_wrapper.conf`. and restart NA.

Setting up an Integrated Development Environment

Setting up for an Integrated Development Environment (IDE) is similar to the command-line environment. You need to provide the location of `truecontrol-client.jar` to your IDE. In many editors, this is an option for the project. Details follow for selected IDEs.

Eclipse:

1. Go to File → New → Project.
2. Select Java Project and click Next.
3. Set the Project Name to Java API and click Next.

4. Click the Libraries tab and click Add External JARs.
5. Add truecontrol-client.jar.
6. Click Finish.

Jbuilder 5:

1. Go to the menu Project:Project Properties.
2. In the dialog box, select the Paths tab then the Required Libraries sub-tab.
3. Click Add then the New button.
4. Enter the name “NA API.”

Navigate to the correct directory and select truecontrol-client.jar

Setting Up a Command-line Environment

If you are invoking javac and java from the command line, you can easily set up a command line environment to prepare to use the Java API. Append `truecontrol-client.jar` to your classpath.

Do not put the `truecontrol-client.jar` in jre/lib/ext. NA’ Java processes will not start.

To verify that your environment is correct, compile and run `Example0.java`. If `Example0.java` is not in your local directory, you can copy the file from `$INSTALL_DIRECTORY/client/sdk/examples/java/Example0.java` or you can “cd” into that directory.

Keep in mind that if you “cd” into the directory, you must have directory permissions to compile the class. Here is what you should see:

```
bash# INSTALL_DIRECTORY=/<install directory>
bash# export INSTALL_DIRECTORY
bash# export CLASSPATH=$CLASSPATH:.:$INSTALL_DIRECTORY/client/truecontrol-
client.jar:$INSTALL_DIRECTORY/jre/lib/ext/bcprov-jdk16-141.jar
bash# <install directory>/jre/bin/javac -d . Example0.java
bash# <install directory>/jre/bin/java com.rendition.api.examples.Example0

Starting Example0
Session connectivity verified

<install directory>/client/sdk/examples/java>
```

Programming with the Java API

If your Java environment is pointing to a different Java than the default NA one, include the following jar to your CLASSPATH:

`$NA/jre/lib/ext/bcprov-jdk14-119.jar`, where `$NA` is your NA local directory.

You will need this file to run the Java examples that are shipped with NA.

Working with the Session object

All interaction with the Java API starts with a Session object.

Session contexts

`Session.open()` creates a session context for execution of commands. This method actually contacts the NA server via Java RMI on port 1099, and authenticates the user using the supplied arguments. The server parameter is optional; if omitted, localhost will be contacted.

Make sure that you close the session context when done with it via the `Session.close()` method. Like file handles, there is a finite supply of sessions.

Session objects are thread-safe, so you may use the Session object across threads to do overlapping operations.

UserIDs and Permissions

When opening the session, you must provide a user name and password for a valid NA user. NA makes no distinction between the user identities used to log into the WebUI, CLI, or Telnet/SSH Proxy and those used to access the API.

Each NA API call will be validated against the user identity provided to ensure the user has sufficient privileges to run the requested operation, just as the user's privileges would be validated by the WebUI, CLI, or Telnet/SSH Proxy.

It is recommended that you set up dedicated NA users for API access, with appropriate privilege levels for the kinds of applications you are writing. For example, an application that only retrieves data from NA might require a Limited Access user, whereas an application to remove out-of-date information from the system would require Admin privileges.

When calling `Session.open()`, note that the user name and password are case sensitive. If you provide bad authentication information, you will receive a NAAPIException.

Executing Requests

You can send commands to the NA server through the Session object.

Relationship between the API and the CLI or Telnet/SSH Proxy

`Session.exec()` is used to send a request to the NA API. The commands accepted by `Session.exec()` are, with the exceptions noted below, syntactically identical to those accepted by the CLI or the Proxy interface interactive mode. You may find it convenient to test commands intended for your programs by telnetting to your server and manually entering the commands.

CLI-only Commands

All commands accepted by the CLI or Telnet/SSH Proxy are valid for `Session.exec()`, except for the `help`, `connect`, `os ping`, and `os traceroute` commands. The API does not support these.

Handling Results

This section covers the returned objects and exceptions thrown by `Session.exec()`.

Status

The return value from `Session.exec()` is a Result object. `Result.getSucceeded()` will return `true` if the command completed successfully; or `false` if the command failed. You can get extended information codes via `Result.getReturnStatus()`.The status codes vary based on the type of request; they are documented in the “Commands” section.

> Certain API commands schedule a task to be run on the server and then return immediately without waiting for the task to complete. Call `Result.getTex()` to get a string containing the Task ID number of the task that is created.
>
> Most of the commands accept an optional –sync argument to indicate that the API call should block until the task has completed and then return the task result. If –sync is used, `Result.getText()` returns the result output from running the task instead of the Task ID number. See Appendix A for information on which commands accept the –sync option.

Simple Results

If the command returns a simple String result, use Return.getString to examine the result. The commands with String results are shown in the “Commands” section.

Complex results: ResultSet type

Many commands return a complex result with many fields, or several rows of such field-based data. The commands with complex results are shown below in the Commands section.

The Java API uses JDBC’s ResultSet interface to provide access to complex results. You can learn more about this interface in numerous books and online resources for JDBC. The samples `Example1.java, Example2.java` and `Example3.java` show how to work with ResultSet data.

To interact with ResultSet data, you must know the valid columns and types for each command. This information is provided below in the Commands section, under the table heading Return Value(s). You can also use the metadata interface to work with ResultSets in a generic way, so that you do not have to hard code the data types being returned from a given command.

The `Result` object has a `toSring()` method that is useful for debugging to get more information about the results of your API calls.

Exceptions

The following exceptions are sent by Session.exec(). Details can be found in the javadocs.

- NAAPIException: Generic API exceptions.
- ResultSetException: Thrown when incorrect method is used to retrieve a field from a ResultSet, e.g. calling getInt on a String field.
- NotSupportedException: Thrown when an unsupported ResultSet method is called. See the javadocs for which methods are supported.

Metadata

Metadata (meaning data-about-data) describes the data fields returned in a ResultSet. You can use metadata to determine how many fields were returned in the result set, the name for each field, and the data type for each field. `ResultSet.getMetaData()` is the method that returns metadata for a result set.

`Example3.java` shows a useful application for metadata, processing any user-supplied command. You can see how metadata is required to print results from a command whose identity is not known at compile time.

Note: Developers familiar with C-based languages such as Java and C++ should note that the column indexes for all metadata methods are 1-based not 0-based.

Integration Hooks

Run External Application Tasks

NA's Run External Application task enables you to invoke applications and scripts from within NA. This includes the ability to run your own Java API applications. In other words, you can extend NA's functionality by using this Java API to write your own application that integrates with outside applications and datasources.

Using the Web UI, you can configure NA to invoke your own application when certain system events occur. Note that if you need to call out to third-party software from your custom application, you have several options:

- Use that application's Java API, if one is provided.
- Use that application's non-Java API via RMI.
- Use a communication channel such as message queuing, CORBA, sockets, and so on.
- Interact via the file system or databases.
- Call that application directly via `Runtime.getRuntime().exec()`

Callbacks

There are two important callback methods from NA to your Java code that you can use to customize the NA engine:

- The Approver interface
- The Cleaner interface

Note that these callbacks cannot be remoted. The code must be present on the NA server. You can provide a server-side stub which uses your own RMI calls to pass the call along to the client.

Also note that the following directions require you to modify NA's configuration files. Make sure to keep a backup copy, as a corrupted configuration may make the server unstable.

Approver Callback Interface

The approver interface is provided to allow an external ticketing system to approve or deny a particular user's access to a device.

NA will call the user-provided approver in the following circumstances:

- Before the Telnet/SSH Proxy opens a device session – approveInterceptorSessionLogin() is invoked
- Before a device configuration is modified – hasModifyConfigPermission() is invoked
- Before a device group configuration is modified – hasGroupModifyConfigPermission() is invoked
- Before any CLI command is processed – hasPermission() is invoked

See the javadoc comments for details on when these methods are invoked, and what parameters are passed. Note, some methods are overloaded.

Approver Use Cases

Here are two possible cases where this might be useful. The cases integrate NA with a third-party ticketing system (3PT).

Case 1: External task approval

- *Network Engineer* – Schedules a config deployment for ticket T and work request W.
- *NA* – Requests approval for change to device D with ticket T and work request W.
- *Ticketing System* – Returns true or false with a reason R.
- *ONA* – Lets the task run, or marks it as failed setting the Result to 'not approved by 3PT because R'.

Third-party ticketing system (3PT) should synchronously return true or false using internal data (such as time of day and ticket status) so no timeout is needed.

Case 2: External session approval

- *Network Engineer* – Requests session on Device D for work request W.
- *NA* – Requests approval for connection to device D for work request W.
- *Ticketing System* – Returns true or false with a reason R.
- *NA* – Starts the session or displays the error 'Session not approved by 3PT because R'.

Approver Coding

NA will use the configuration file appserver.rcx to determine what class to use for the session approver. A default do-nothing (always approve) approver, `com.rendition.api.DefaultApprover`, is provided by NA.

To install your own approver, follow these steps:

- Code your own approver that implements the `com.rendition.api.Approver` interface
- Modify "approver/className" option in appserver.rcx file, specifying your own class.
- Build a JAR file that contains all your new classes and copy it into <installation directory>/server/ext/jboss/server/default/lib directory.

Cleaner Callback Interface

The cleaner interface returns custom actions upon user exiting a NA device session. NA will call the user-provided cleaner when the Telnet/SSH Proxy closes a device session.

Cleaner use case

Case 1: External change annotation

- *Network Engineer* – Configures Device D for work request W. Closes session.
- *NA* – Calls cleaner for connection to device D for work request W.
- *Custom code* – Calls out to ticketing system.
- *Ticketing System* – Returns reason R for change.
- *Custom code* – Calls Java API to copy reason R into custom data on device.

Cleaner Coding

NA will use the configuration file appserver.rcx to determine what class to use for the session cleaner. A default do-nothing cleaner, `com.rendition.api.DefaultCleaner`, is provided by NA.

To install your own cleaner, follow these steps:

- Code your own cleaner that implements the `com.rendition.api.Cleaner` interface
- modify "cleaner/className" option in appserver.rcx file, specifying your own class
- build a JAR file that contains all your new classes and copy it into <installation directory>/server/ext/jboss/server/default/lib directory

User Permissions

The following table describes user permissions that are required to execute the CLI commands described in Appendix A. These roles are the default roles created by NA. An administrator can create new permission groups and roles, and assign them to users.

User Permissions Matrix

			System Administration			
User	**Description**	**Reconfigure Devices** Log into enable mode View unmasked passwords Run configuration scripts Deploy configuration Change passwords	**Highly Sensitive** Manage users Delete historical information Edit/delete any users's tasks Define custom diagnostics	**Other** Administrative settings Authentication rules View all telnet/SSH sessions	**Group Tasks** Custom scripts & diagnostics Snapshots & polling Driver discovery Syslog configuration Password deployment Import FQDN lookup	**Modify NA Information** Devices Groups Configuration comments
Admin	Admins are highly trusted users responsible for administering the NA application, managing users, setting policy, and running network-wide operations requiring a high degree of skill and care. They have permission to take any action in the NA system on any device.	All Devices	**X**	**X**	**X**	**X**
Power User	Power users are highly trusted expert engineers allowed to perform most actions in the system. They can reconfigure and act on groups of devices in the system. They may be restricted to which devices they have permission to reconfigure.	Specified Devices		**X**	**X**	**X**
Full Access	Full Access users are qualified network engineers trusted with passwords to configure some or all devices in the network. They have permission to modify most information in the NA database, and can reconfigure devices one-at-a-time but not in batch. They may be restricted as to which devices they have permission to reconfigure.	Specified Devices				**X**

User Permissions Matrix

			System Administration			
User	**Description**	**Reconfigure Devices** Log into enable mode View unmasked passwords Run configuration scripts Deploy configuration Change passwords	**Highly Sensitive** Manage users Delete historical information Edit/delete any users's tasks Define custom diagnostics	**Other** Administrative settings Authentication rules View all telnet/SSH sessions	**Group Tasks** Custom scripts & diagnostics Snapshots & polling Driver discovery Syslog configuration Password deployment Import FQDN lookup	**Modify NA Information** Devices Groups Configuration comments
Limited Access	Limited Access users are operator users that do not have passwords to configure network devices. They have permission to view but not modify most information in NA. Sensitive information such as device passwords will be masked out. They cannot run batch operations or operations that reconfigure network devices.	**No** Devices				

Chapter 2: Perl API

The Perl API enables NA to communicate with external systems and vice-versa. The Perl API can be used to add and retrieve data to and from NA.

Common tasks, such as adding devices into NA and alerting third-party systems when a device configuration changes, can be programmatically accessed using the Perl API. Users who want to use other languages can automate their common functions using CLI or Telnet protocols.

Installing the Enhanced Perl API

The following modules are provided on the NA 9.1 Distribution CD:

Opsware::NAS::Util

Opsware:NAS::Client

Opsware::NAS::Connect

Note: Due to limitations of ActiveState ActivePerl on Windows, if you use this environment you will not be able to use SSH connections with the NA Perl API. As a workaround, install the NA client on a supported Linux or Solaris system and run the NA Perl API from that system.

Installation Requirements

Perl version 5.8 or later is required for installing on a Linux/Solaris platform. ActivePerl 5.8.x is required when installing on a Windows platform.

Note: If Perl is installed on your the NA Server host, the NA installer will automatically install the NA Enhanced Perl API modules when NA is installed. If you install Perl after installing NA, install the NA Enhanced Perl API with the Auto Installer.

If you want to install the Perl API after installing NA, see Appendix A for instructions on installing the NA Perl API.

Perl Documentation

After the Perl API is installed, you can view the following Perl POD pages:

- perldoc Opsware::NAS::Client
- perldoc Opsware::NAS::Connect
- perldoc Opsware::NAS::Client::4_5_x
- perldoc Opsware::NAS::Client::6_0_x

Your Perl distribution can also build HTML files for the documentation.

Examples

There are Perl API examples in the demo directory. These examples show how to use the Perl API. Keep in mind that it is possible to run the examples without installing the Perl modules by remaining in the demo directory and supplying the relative (or full) path to each example, as in:

- `<installation directory>/client/perl_api/demo/list_users.pl -user <NA user> -pass <NA password> -host <NA server host>:80`
- `<installation directory>/client/perl_api/connect.pl --user=<NA User> --pass=<NA PASSWORD>--host='localhost:8023' --devicc=<device-IP>`

If NA is not running with the default HTTP port (80), replace 80 in the examples with the port number NA is using.

Chapter 3: SOAP API

Using the WSDL file

As part of the installation, there are files named *api.wsdl.** in the *client/sdk/* directory of the core and client installations. These files represent the toolsets on which testing was performed. These toolsets can be stub compilers or other toolsets that can import WSDL files. The following information describes specific toolsets and known issues.

As with any WSDL file, the endpoint is specified in the <service> element. The value of the endpoint will be set to 'localhost' by default. This should be changed if it poses a problem for the target environment.

gSOAP - c/c++ stub compiler

For this toolset, use the *api.wsdl.gsoap* file.

When using the wsdl2h tool, there is a long list of warnings related to the use of a "type" versus an "element" in the types section. These warnings are benign. Normally, when using this tool, you use the generated .nsmap file. This structure contains URIs that do not work well with a server side implementation. As a workaround, use something similar to the following in the program:

```
SOAP_NMAC struct Namespace namespaces[] =
{
        {"SOAP-ENV", "http://schemas.xmlsoap.org/soap/envelope/", NULL,
NULL},
        {"SOAP-ENC", "http://www.w3.org/2003/05/soap-encoding",
"http://www.w3.org/*/soap-encoding", NULL},
        {"xsi", "http://www.w3.org/2001/XMLSchema-instance",
"http://www.w3.org/*/XMLSchema-instance", NULL},
        {"xsd", "http://www.w3.org/2001/XMLSchema",
"http://www.w3.org/*/XMLSchema", NULL},
        {"wsdl", "http://tempuri.org/wsdl.xsd", NULL, NULL},
        {"nas", "http://opsware.com/nas/", NULL, NULL},
        {NULL, NULL, NULL, NULL}
};
```

axis1.4 and wsdl2java - Apache axis stub compiler for java

For this toolset, use the *api.wsdl.axis* file.

Java has a hard limit of 256 parameters to a method. This tool generates constructors for each of the objects it finds in the WSDL file. One of these objects, Row, contains more than 256 members. As a result, one of the generated constructors has too many arguments. This can be solved in your make or ant environment by doing a deletion or replacement, for example:

```
sed -i -e '/public Row($$/, /}/d' $OUT_CLASS_DIR/Row.java
```

axis2 and wsdl2java - Apache axis2 stub compiler for java

For this toolset, use the *api.wsdl.axis2* file.

The constructor problem described in the previous section does not apply to this tool for client-side generated code.

The generated stub will be located at:

src/com/opsware/nas/_72/NetworkManagementApiStub.java

You must modify the generated stub to work properly against the service to avoid the runtime exception regarding an incorrect subelement *Result*. A command like the following can be used to remedy this problem prior to compiling the stubs to class files:

```
sed -i -e 's/QName(""/QName("http:\/\/opsware.com\/nas\/72"/g'
src/com/opsware/nas/_72/*.java
```

wsdl2py - Python stub compiler from the ZSI package

For this toolset, use the *api.wsdl.wsdl2py* file.

Once the stubs are created, there is a change which must be made to the resulting files. One of the exported functions is called "import". Since this is a reserved word in Python, using import creates a conflict. The Python stub compiler creates a method for each of the exposed methods of the service. For this module to compile, you must rename this method. As part of the build process, doing a simple replacement in the stub file after it is generated solves the problem:

```
sed -i -e 's/def import/def _import/g' `grep -l "def import" * | grep -
v Makefile `
```

The reason for the embeded grep is because different versions of ZSI will generate different stub names.

SOAPpy - Python module

For this toolset, use the *api.wsdl.soappy* file.

This module allows the dynamic loading of a file, rather than creation of stubs like other tools. One way to make usage easier is making sure the endpoint in the <service> tag points to the desired NA server before using the file to create the WSDL.Proxy object. By default, this file comes with an endpoint of localhost. It may be easier to modify the file that this tool uses rather than explicitly changing the endpoint when using this toolset.

.NET wsdl.exe compiler

For this toolset, use the *api.wsdl.dotnet* file.

There are compabibility issues with Visual Studio .NET 2003. At this time, only wsdl.exe from Visual Studio .NET 2005 is supported. If used with VS 2003 is required, generating stubs in VS 2005 and using 2003 should solve the issue.

wsdl.exe does not create custom exceptions based on the <fault> information in the WSDL. To handle exceptions thrown by the server, something like the following will be required in the program:

```
try
{
           result = api.login( inParms );
}
catch( System.Web.Services.Protocols.SoapException e )
{
    Console.WriteLine( "Caught exception: " + e.Message );
    //  e.Detail contains the parsable node
}
```

Appendix A: Installing the Perl API

There are two methods for installing the Perl API:

- Auto Installer method
- Manual Install method

Auto Installer Method

The Auto Installer installs all of the Opsware::NA modules, as well as their dependencies.

- Open a shell. If you are on a Windows platform, open a command prompt. If you are on a Linux or Solaris platform, you can either open a command shell or SSH into the NA server.

  Note: You will need privileges to both create and modify files for NA and Perl. As a result, you might need Administrator privileges on a Windows Platform and root privileges on Linux or Solaris platforms.

- Change to the directory where NA is installed. This directory will have been set when you installed NA.
- To run the install script, enter: `perl client/perl_api/har/install.pl`

  Note: If Perl is not in your path and/or you have multiple Perl versions installed, use the full path to the Perl executable that you will be using. This should also match the value for the Perl interpreter set in the NA server configuration.

This procedure installs all of the Opsware::NAS modules, as well as their dependencies. However, only "pure perl" dependencies are provided. For example, SOAP::Lite is provided, which includes a minimalist lightweight XML parser. For the best performance, it is recommended that you have the XML::Parser module installed.

If you are using ActivePerl (with a Perl version of 5.8 or better), the XML::Parser module is included with the distribution. Otherwise, you will need to use PPM, CPAN, or manually download and install the module.

Manual Install Method

Confirm that certain versions of Perl and/or Perl modules (that are not part of some core Perl distributions) are installed before you begin. See the META.yml file within each package/tarball for its requirements.

If your Perl distribution does not contain all of the required Perl modules, they are available at http://www.cpan.org and/or via PPM. (If you are using ActivePerl, try PPM first.)

To install any of the required modules, use one of the following commands:

- ppm install SOAP-Lite
- cpan install SOAP::Lite

Note that PPM (ppm.exe) is part of the ActivePerl distribution. If you are using ActivePerl, it is recommended that you use the PPM method. You can also run PPM without arguments and then issue the install command. You may need to do this for some Perl modules that have multiple versions to choose from, followed by install # where # is the item in the list returned by the install command. Keep in mind that PPM prefers to use the '-' as a namespace separator in place of the Perl '::' separator.

Note: NMAKE.EXE is installed when installing NA on a Windows platform. It is located the /client directory. CPAN is simply a wrapper for the perl -MCPAN -e shell command. The CPAN command (or cpan.exe) is part of the core Perl install on all Perl versions since 5.8.0 (including ActivePerl).

Keep in mind that the installation could fail if your Perl installation does not meet certain requirements. See the “Installation Requirements” section. In addition, the Opsware::NA Perl modules are distributed as compressed tarballs, similar modules on CPAN. They are located in <NA_ROOT>/client/perl_api/Opsware/.

To untar and uncompress all of the modules at one time, use the ptar command. ptar is distributed as part of the popular Perl module Archive::Tar, which is included in the standard ActivePerl distributions. To view the contents of the directory and to extract the contents into your current directory, enter: ptar -xzvf PATH/TO/whatever.tar.gz.

For each of the following modules, uncompress and untar the module(s) and change to the directory that was created:

- Opsware::NAS::Util
- Opsware::NAS::Client
- Opsware::NAS::Connect

To install the Perl API on a Windows platform with ActivePerl (or any platform running a version of Perl that has the Module::Build module installed):

- perl Build.PL
- perl Build build
- perl Build test
- perl Build install

You may also use the traditional CPAN method. Enter:

- perl Makefile.PL
- make
- make test
- make install

Note: If you are using the CPAN method on a Windows platform, you will need to enter nmake rather than make.