Technical Report

Security hardening guide for NetApp Manageability SDK

Swati Godha, NetApp

May 2024 | TR-4945

Abstract

This technical report provides guidance and configuration settings for NetApp® Manageability SDK (NMSDK) to help organizations meet prescribed security objectives for information system confidentiality, integrity, and availability.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Introduction

The evolution of the current threat landscape presents an organization with unique challenges for protecting its most valuable assets: data and information. The advanced and dynamic threats and vulnerabilities we face are ever increasing in sophistication. Coupled with an increase in the effectiveness of obfuscation and reconnaissance techniques on the part of potential intruders, system managers must address the security of data and information in a proactive manner. This guide seeks to help operators and administrators in that task with the confidentiality, integrity, and availability integral to the NetApp solution.

Verifying integrity of NMSDK packages

This section describes the two methods of verifying the integrity of NMSDK binaries: verifying the checksums and verifying the signature.

Verifying the checksums

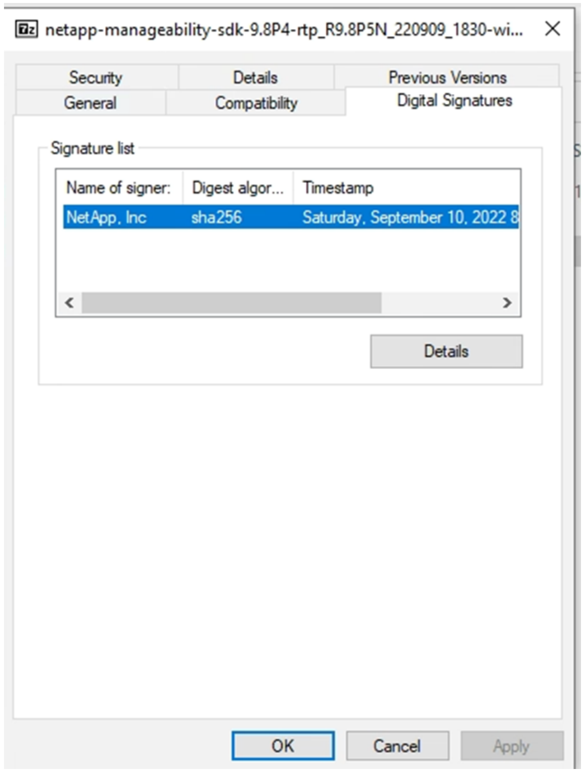
Checksums are provided on the NMSDK download page. Users must verify the checksums of the downloaded packages against the checksum provided on the NMSDK download page.

Verifying the signature

Verifying the signature on Windows

After downloading the NMSDK from NetApp Support Site, users can verify the signature of exe. To verify, right-click the downloaded exe and open Properties. In the Properties dialog, from the Digital Signatures tab, the name of signer should display as NetApp, Inc (Figure 1).

Figure 1) Verifying the signature on Windows.



Name of signer:	Digest algor...	Timestamp
NetApp, Inc	sha256	Saturday, September 10, 2022 8

Verifying the signature on Linux

Along with the product zip for Red Hat Enterprise Linux (RHEL), NetApp shares public key in signed folder with customers on the product download page. User can use the public key to verify the signature for the following product zip:

```
#> openssl dgst -sha256 -verify <public key> -signature <signature file> <Binary>
Example:
#> openssl dgst -sha256 -verify csc-prod-NMSDK-LINUX.pub -signature netapp-manageability-sdk-
9.8P8-linux.zip.sig netapp-manageability-sdk-9.8P8-linux.zip
Verified OK => response
```

Establishing a connection with ONTAP

Username and password

While using NMSDK to test any NetApp ONTAP® API, the ONTAP username and password are required to connect with ONTAP. The ONTAP password policy includes a combination of digits, lowercase characters, uppercase characters, special characters, and password-length requirements.

Certificate-based authentication API access

When using the NMSDK API access to ONTAP, you must use certificate-based authentication instead of the user ID and password authentication.

To generate a self-signed certificate and install on ONTAP, complete the following steps:

1. Using OpenSSL, generate a certificate by running the following command:

```
openssl req -x509 -nodes -days 1095 -newkey rsa:2048 -keyout test.key -out test.pem -subj
"/C=US/ST=NC/L=RTP/O=NetApp/CN=cert_user"
```

This command generates a public certificate named `test.pem` and a private key named `key.out`. The common name, CN, corresponds to the ONTAP user ID.

2. Install the contents of the public certificate in privacy enhanced mail (pem) format in ONTAP by running the following command and pasting the certificate's contents when prompted:

```
security certificate install -type client-ca -vserver ontap9-tme-8040
```

3. Enable ONTAP to allow client access through SSL and define the user ID for API access.

```
security ssl modify -vserver ontap9-tme-8040 -client-enabled true
security login create -user-or-group-name cert_user -application ontapi -authmethod cert -role
admin -vserver ontap9-tme-8040
```

4. In the following example, the user ID `cert_user` is now enabled to use certificate-authenticated API access. A simple Manageability SDK Python script using `cert_user` to display the ONTAP version appears as follows:

```
#!/usr/bin/python

import sys
sys.path.append("/home/admin/netapp-manageability-sdk-9.5/netapp-manageability-
sdk9.5/lib/python/NetApp")
from NaServer import *

cluster = "ontap9-tme-8040"
transport = "HTTPS"
port = 443
style = "CERTIFICATE"
cert = "test.pem"
key = "test.key"
```

```
s = NaServer(cluster, 1, 30)
s.set_transport_type(transport)
s.set_port(port) s.set_style(style)
s.set_server_cert_verification(0)
s.set_client_cert_and_key(cert, key)

api = NaElement("system-get-version")
output = s.invoke_elem(api)
if (output.results_status() == "failed"):
  r = output.results_reason()
  print("Failed: " + str(r))
  sys.exit(2)

ontap_version = output.child_get_string("version")
print ("V: " + ontap_version)

The output of the script displays the ONTAP version.
./version.py
V: NetApp Release 9.5RC1: Sat Nov 10 05:13:42 UTC 2018
```

Establishing a connection with Active IQ Unified Manager

While using NMSDK for testing any Active IQ Unified Manager APIs, Active IQ Unified Manager's user name and password is required to connect with Active IQ Unified Manager. Active IQ Unified Manager’s password policy includes a combination of digits, lowercase characters, uppercase characters, special characters, and password-length requirements.

Logging details

Audit log

The audit log provides ONTAP API invocation history; it is located at `/etc/log/auditlog`.

For Active IQ Unified Manager, you can find the invocation history at <`dfm-server-installdirectory>/log/`.

EMS log

The EMS log provides ONTAP API processing information; it is located at `/etc/log/ems`.

Syslog

Syslog provides ONTAP API error messages; it is located at `/etc/messages`.

DFM server log

The DFM server log provides error logging information about Active IQ Unified Manager APIs for the Active IQ Unified Manager Core Package failure; it is located at <dfm-server-installdirectory >/log/.

NMSDK logging

Beginning with NMSDK 9.8P6, it captures the minimalistic logs from all C and Java APIs on the client machine. The following is an example of an `nmsdk.log`:

```
2023-04-25 17:08:24.032 [application:DEBUG]: [6572:0x27e8]: na_server_invoke:IN    API: system-
get-version
2023-04-25 17:08:24.082 [application:DEBUG]: [6572:0x27e8]: na_server_invoke_elem invoked
2023-04-25 17:08:24.127 [application:DEBUG]: [6572:0x27e8]: na_server_invoke_elem_http:IN     on
host=<host_IP>,look_up_host=1,major=<major_version>,minor=<minor_version>,style=0,
2023-04-25 17:08:25.096 [application:DEBUG]: [6572:0x27e8]: na_server_invoke_elem_http:OUT
```

```
2023-04-25 17:08:25.146 [application:DEBUG]: [6572:0x27e8]: na_server_invoke:OUT    API: system-
get-version
```

Linux platforms

Customization of log paths using an environment variable:

On Linux platforms, the default nmsdk log folder path is `/var/log/nmsdk/`. This folder path is customized by setting an environment variable named NMSDK_LOG_LOCATION to a user defined location. See the below example of a command in a Linux shell session. Ensure that this location has all the required read and write permissions.

```
setenv NMSDK_LOG_LOCATION /var/customlocation/nmsdk/
```

or

```
export NMSDK_LOG_LOCATION=/var/customlocation/nmsdk/
```

Since the above technique is to set an environment variable, it works only in a Linux shell session into which you typed this command. If you need to keep an environment variable across all the sessions, then add the following example lines in `.bash_profile` and `.bashrc` files by `root` user.

In file `.bash_profile`

```
NMSDK_LOG_LOCATION=/var/customlocation/nmsdk/
export NMSDK_LOG_LOCATION
```

In file `.bashrc`

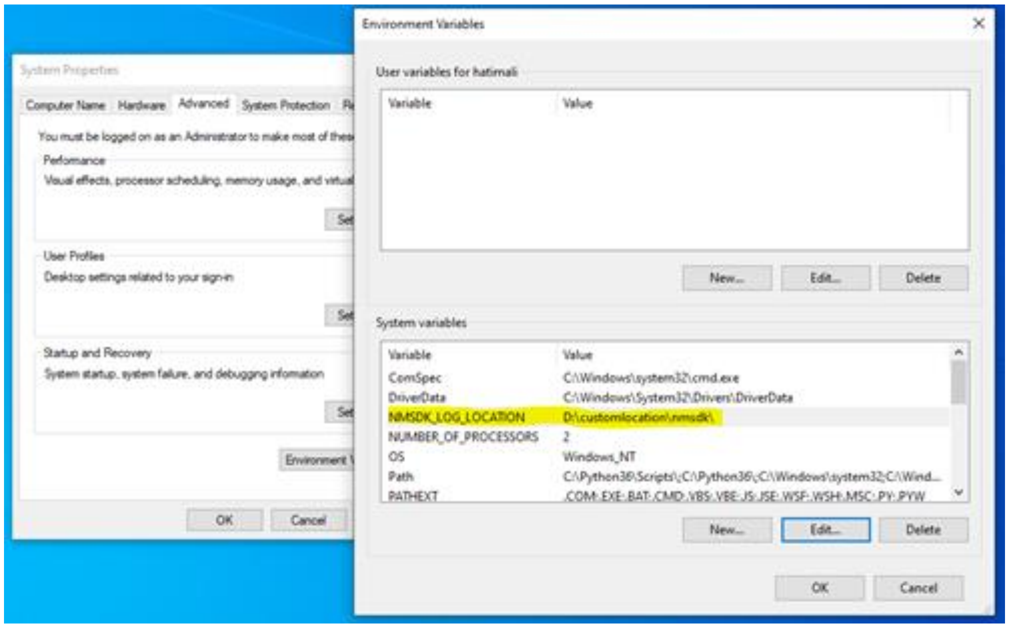
```
export NMSDK_LOG_LOCATION=/var/customlocation/nmsdk/
```

Windows platforms

Customization of log paths using environment variables:

On Windows platforms, the default `nmsdk` log folder path is `C:\nmsdk\`. This folder path is customized by setting the system environment variable named `NMSDK_LOG_LOCATION` to a user defined location. Ensure that this location has all the required read and write permissions.

Figure 2) Environment variable - NMSDK_LOG_LOCATION.



You must be logged on as an Administrator to make most of thes

Visual effects, processor scheduling, memory usage, and virtual

Desktop settings related to your sign-in

System startup, system failure, and debugging information

User variables for hatimali

Variable	Value

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NMSDK_LOG_LOCATION	D:\customlocation\nmsdk\
NUMBER_OF_PROCESSORS	2
OS	Windows_NT
Path	C:\Python36\Scripts\;C:\Python36\;C:\Windows\system32;C:\Wind...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW

NMSDK log rotation and integrity scripts

NMSDK log rotation and integrity is a technique whereby the log generated from NMSDK APIs in a specific folder location auto rotates; this technique compresses the log for efficient use of disk storage and checks for unauthorized changes.

There are certain utilities that are used for implementation of log rotation and integrity scripts, which are described in detail in later sections.

Implementation

The scripts written for log rotation and integrity are implemented on both Linux and Windows platforms. The auth.log file is updated with information about the log file’s integrity once the log rotation is executed. If any log file is tampered with, the information is logged. The following is an example of an auth.log:

```
The log file nmsdk.log.5.gz for respective checksum file nmsdk.log.5.gz.cksum is deleted or
tampered
Checksum mismatch for nmsdk.log.4.gz
```

Linux platforms

The following are the configurations and utilities that are used to test the scripts on the Linux platform:

Operating system:

- Red Hat Enterprise Linux

Packages:

- Logrotate version 3.8.6
- Python version 3.6.8
- sha256sum (GNU coreutils) version 8.22

Note: All the above installed packages’ location paths should be defined in PATH environment variables.

The log rotation and integrity script is comprised of the following three files. All three files are placed in the same folder location.

1. logrotatescript.config
2. generateLogFileChecksum.py
3. verifyLogIntegrity.py

The "logrotatescript.config" file is a configuration file which is responsible for invoking two python files named "generateLogFileChecksum.py" and "verifyLogIntegrity.py" during log rotation. The following is the blueprint of the "logrotatescript.config" file. This configuration file should be written in Linux text style format:

```
<NMSDK_LOG_FILE_PATH_PLACE_HOLDER>
{
        size <NMSDK_LOG_FILE_SIZE_PLACE_HOLDER>
        rotate <NMSDK_LOG_FILE_ROTATION_COUNT_PLACE_HOLDER>
        copytruncate
        compress
        delaycompress
        missingok

        firstaction
                echo firstaction IN > /dev/null
                python3 <NMSDK_PROPERTY_FOLDER_PATH_PLACE_HOLDER>verifyLogIntegrity.py $1 >
/dev/null
                echo firstaction OUT > /dev/null
        endscript
        lastaction
                echo lastaction IN > /dev/null
                python3 <NMSDK_PROPERTY_FOLDER_PATH_PLACE_HOLDER>generateLogFileChecksum.py $1 >
/dev/null
                echo lastaction OUT > /dev/null
        endscript
}
```

Windows platforms

The following are the configurations and utilities that are used to test the scripts on Windows platforms:

Operating system:

- Windows 10 Enterprise Version 21H2
- Microsoft .NET Framework 4.7.2 or later

Packages:

- Logrotate version 0.0.0.18 (Install it from link: https://sourceforge.net/projects/logrotatewin/
- Python version 3.6.8

Note: All the above installed packages' location paths should be defined in PATH environment variables.

The log rotation and integrity script is comprised of the following three files. All three files are placed in the same folder location.

1. logrotatescript.config
2. generateLogFileChecksum.py
3. verifyLogIntegrity.py

The “logrotatescript.config” file is a configuration file which is responsible for invoking two python files named “generateLogFileChecksum.py” and “verifyLogIntegrity.py” during log rotation. The following is the blueprint of “logrotatescript.config” file. This configuration file should be written in Linux text style format:

```
<NMSDK_LOG_FILE_PATH_PLACE_HOLDER>
{
        size <NMSDK_LOG_FILE_SIZE_PLACE_HOLDER>
        rotate <NMSDK_LOG_FILE_ROTATION_COUNT_PLACE_HOLDER>
        copytruncate
        compress
        delaycompress
        missingok

        prerotate
                echo prerotate IN > nul
                python.exe <NMSDK_PROPERTY_FOLDER_PATH_PLACE_HOLDER>verifyLogIntegrity.py %1 > nul
                echo prerotate OUT > nul
        endscript
        postrotate
                echo postrotate IN > nul
                python.exe <NMSDK_PROPERTY_FOLDER_PATH_PLACE_HOLDER>generateLogFileChecksum.py %1
> nul
                echo postrotate OUT > nul
        endscript
}
```

There are four placeholders used in this file which will have default values and if you have to customize it, you will need to replace them with their respective values.

- **<NMSDK_LOG_FILE_PATH_PLACE_HOLDER>:** This placeholder needs to be replaced with the fully qualified file path of the nmsdk.log file for the log rotation and its integrity also needs to be checked.
- **<NMSDK_LOG_FILE_SIZE_PLACE_HOLDER>:** This placeholder needs to be replaced with the threshold size of the log file at which the log rotation takes place when the size of the log file hits this threshold size. Log files are rotated only if they grow bigger than size bytes. If size is followed by k, the size is assumed to be in kilobytes. If the M is used, the size is in megabytes, and if G is used, the size is in gigabytes. So, size 100, size 100k, size 100M and size 100G are all valid.
- **<NMSDK_LOG_FILE_ROTATION_COUNT_PLACE_HOLDER>:** This placeholder needs to be replaced with the number of rotations taking place for that file. Log files are rotated count times before being removed or overwritten. If the count is 0, old versions are removed rather than rotated.
- **< NMSDK_PROPERTY_FOLDER_PATH_PLACE_HOLDER>:** This placeholder needs to be replaced with the fully qualified folder path where the nmsdk property/configuration/script files reside.

References

Please refer to the following link for your references:

- https://linux.die.net/man/8/logrotate
- https://sourceforge.net/p/logrotatewin/wiki/LogRotate/
- https://sourceforge.net/projects/logrotatewin/
- https://github.com/plecos/logrotatewin

Using NMSDK utilities

NMSDK provides two utilities that help users test the APIs: apitest and ZExplore Development Interface (ZEDI).

apitest

Apitest is a command-line utility to test APIs. This utility is suitable for API users who are at the beginner's level. It is available in C, Java, Perl, C#, VB.NET, PowerShell in Windows, Python, and Ruby. In UNIX-like environments, apitest is available in C, Java, Perl, Python, and Ruby.

Command - `ruby apitest.rb 10.236.156.69 admin <password> system-get-version`

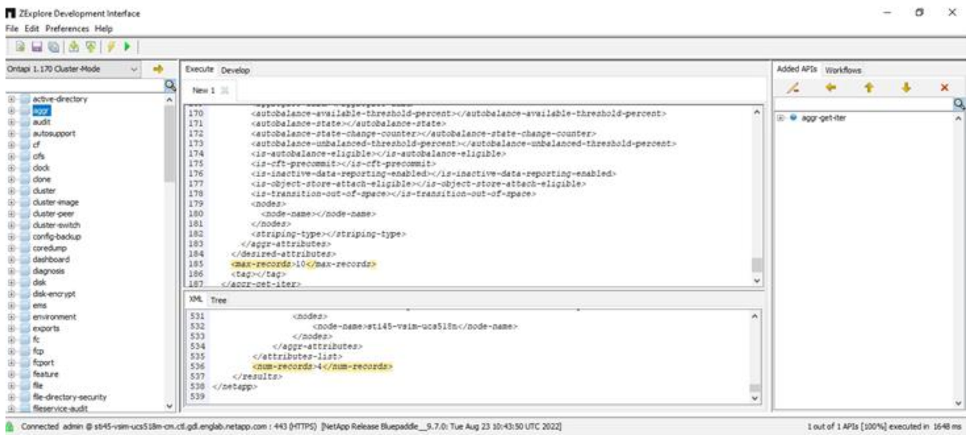
Output:

```
<results status="passed">
        <build-timestamp>1661323269</build-timestamp>
        <is-clustered>true</is-clustered>
        <version>NetApp Release Yellowdog__9.12.1: Wed Aug 24 06:41:09 UTC 2022</version>
        <version-tuple>
                <system-version-tuple>
                               <generation>9</generation>
                                <major>12</major>
                                <minor>1</minor>
                </system-version-tuple>
        </version-tuple>
</results>
```

ZExplore development interface

ZExplore is a UI that can be used to test the ONTAP APIs and NetApp Active IQ® Unified Manager APIs for the Active IQ Unified Manager Core Package. ZEDI supports Java Runtime Environment (JRE) 1.6.0 or later.

Figure 3) ZEDI



```
        <autobalance-available-threshold-percent></autobalance-available-threshold-percent>
        <autobalance-state></autobalance-state>
        <autobalance-state-change-counter></autobalance-state-change-counter>
        <autobalance-unbalanced-threshold-percent></autobalance-unbalanced-threshold-percent>
        <is-autobalance-eligible></is-autobalance-eligible>
        <is-cft-precommit></is-cft-precommit>
        <is-inactive-data-reporting-enabled></is-inactive-data-reporting-enabled>
        <is-object-store-attach-eligible></is-object-store-attach-eligible>
        <is-transition-out-of-space></is-transition-out-of-space>
        <nodes>
          <node-name></node-name>
        </nodes>
        <striping-type></striping-type>
      </aggr-attributes>
    </desired-attributes>
    <max-records>10</max-records>
    <tag></tag>
  </aggr-get-iter>
```

```
                <nodes>
                    <node-name>sti45-vsim-ucs518n</node-name>
                </nodes>
            </aggr-attributes>
        </attributes-list>
        <num-records>4</num-records>
    </results>
</netapp>
```

Connected admin @ sti45-vsim-ucs518m-cn.ctl.gdl.englab.netapp.com : 443 (HTTPS) [NetApp Release Bluepaddle__9.7.0: Tue Aug 23 10:43:50 UTC 2022]   1 out of 1 APIs [100%] executed in 1648 ms

Security-related class and function details

Table 1) Security-related class details with description.

Class name	Description
NaServer	Encapsulates an administrative connection to a NetApp storage system running Data ONTAP 6.4 or later. NaServer can also be used to establish connection with Active IQ Unified Manager (formerly OnCommand Unified Manager). You can construct NaElement objects that represent queries or commands and use `invoke_elem()` to send them to the storage systems or Active IQ Unified Manager.
NaElement	Encapsulates one level of an XML element. Elements can be arbitrarily nested. They have names, corresponding to XML tags, attributes (only used for results), values (always strings), and possibly children, corresponding to nested tagged items. For instructions on using NaElements to invoke ONTAPI API calls, see the NaServer description above.
ARCFour	A utility class implementing ARCFour (the alleged RC4 algorithm) for encryption of a data stream. RC4 was a trade secret of RSA Data Security, Inc. until 1994 when it was released anonymously to a mailing list. Because there is no proof that the leaked version was in fact RC4 and because RC4 is a trademark, it is called "ARCFour," short for "Allegedly RC4".
Base16	Encodes the character arrays into a Base16 string and then decodes them again.
AuthInfo	Contains information about keystore and truststore details that are required for certificate-based authentication.
Base64	Encodes and decodes to and from Base64 notation.
HTTPRequest	Represents an HTTP request.
shttpc.c	This HTTP client functions for TCP/SSL.

Table 2) Function details for Python language.

API name	Description	Class name
__init__(self, server, major_version, minor_version)	Creates a new connection to server. Before use, you either need to set the style to `hosts.equiv` or set the user name (always `root` at present) and password with `set_admin_user()`.	NaServer
set_style(self, style)	• Pass in LOGIN to cause the server to use HTTP simple authentication with a user name and password. • Pass in HOSTS to use the `hosts.equiv` file on the filer to determine access rights (the user name must be `root` in that case). • Pass in CERTIFICATE to use certificate-based authentication with the Unified Manager server. **Note:** If style = CERTIFICATE, you can use certificates to authenticate clients who attempt to connect to a server without the need of a user name and password. This style internally sets the transport type to HTTPS. Verification of the server's certificate is required in order to properly authenticate the identity of the server. Server certificate verification is enabled by default using this style and server certificate verification always enables host name verification. You can disable server certificate (with host	NaServer

	name) verification by using `set_server_cert_verification()`.	
get_style(self)	Gets the authentication style.	NaServer
set_admin_user(self, user, password)	Sets the admin user name and password. At present, `user` must always be `root`.	NaServer
set_server_type(self, server_type)	Pass in one of these keywords: FILER, DFM, or OCUM to indicate whether the server is a storage system (filer) or Active IQ Unified Manager. If you also use `set_port()`, call `set_port()` after calling this routine. The default value is FILER.	NaServer
get_server_type(self)	Gets the type of server to which this server connection applies.	NaServer
set_transport_type(self, scheme)	Overrides the default transport type. The valid transport types are currently HTTP and HTTPS.	NaServer
get_transport_type(self)	Retrieves the transport used for this connection.	NaServer
set_port(self, port)	Overrides the default port for this server. If you also call `set_server_type()`, you must call it before calling `set_port()`.	NaServer
get_port(self)	Retrieves the port used for the remote server.	NaServer
use_https(self)	Determines whether HTTPS is enabled.	NaServer
set_client_cert_and_key(self, cert_file, key_file)	Sets the client certificate and key files that are required for client authentication by the server using certificates. If key file is not defined, then the certificate file will be used as the key file.	NaServer
set_ca_certs(self, ca_file)	Specifies the certificates of the certificate authorities (CAs) that are trusted by this application and that will be used to verify the server certificate.	NaServer
set_server_cert_verification(self, enable)	Enables or disables the server certificate verification by the client. Server certificate verification is enabled by default when the style is set to CERTIFICATE. The host name (CN) verification is enabled during the server certificate verification. Host name verification can be disabled by using the `set_hostname_verification()` API.	NaServer
is_server_cert_verification_enabled(self)	Determines whether the server certificate verification is enabled. Returns True if it is enabled; otherwise, it returns False.	NaServer
set_hostname_verification(self, enable)	Enables or disables host name verification during server certificate verification. The host name (CN) verification is enabled by default during the server certificate verification.	NaServer
is_hostname_verification_enabled(self)	Determines whether host name verification is enabled or not. Returns True if it is enabled; otherwise, it returns False.	NaServer
child_add_string_encrypted(self, name, value, key=None)	Same as `child_add_string`, but encrypts value with `key` before adding the element to the current object. This is only used at the present for certain key exchange operations. Both the client and server must know the value of `key` and agree to use this routine and its companion, `child_get_string_encrypted()`. The default key is used if the given key is None.	NaElement

child_get_string_encrypted (self, name, key=None)	Gets the value of child named `name` and decrypt it with `key` before returning it. The default key is used if the given key is `None`.	NaElement
toEncodedString(self)	Encodes the string embedded with special characters such as &,<,and >. This is mainly useful when passing string values embedded with special characters such as &,<,and > to API. For example: `server.invoke("qtree-create","qtree","abc<qt0","volume","vol0")`	NaElement

Table 3) Function details for Ruby language.

API name	Description	Class name
initialize(server, major_version, minor_version	Creates a new connection to server. Before use, you either need to set the style to `hosts.equiv` or set the user name (always `root` at present) and password with `set_admin_user()`.	NaServer
set_style(style)	• Passes in LOGIN to cause the server to use HTTP simple authentication with a user name and password. • Passes in HOSTS to use the `hosts.equiv` file on the filer to determine access rights (the user name must be `root` in that case). • Passes in CERTIFICATE to use certificate-based authentication with the Unified Manager server. **Note:** If `$style = CERTIFICATE`, you can use certificates to authenticate clients who attempt to connect to a server without the need of a user name and password. This style internally sets the transport type to HTTPS. Verification of the server's certificate is required in order to properly authenticate the identity of the server. **Note:** The server certificate verification is enabled by default using this style and server certificate verification always enables host name verification. You can disable server certificate (with host name) verification by using `set_server_cert_verification()`.	NaServer
get_style()	Gets the authentication style.	NaServer
set_admin_user(user, password)	Sets the admin user name and password. At present, `user` must always be `root`.	NaServer
set_server_type(server_type)	Passes in one of these keywords: FILER, DFM, or OCUM to indicate whether the server is a storage system (filer) or an Active IQ Unified Manager server. If you also use `set_port()`, call `set_port()` after calling this routine. The default value is FILER.	NaServer
get_server_type()	Gets the type of server this server connection applies to.	NaServer
set_transport_type(scheme)	Overrides the default transport type. The valid transport type are currently HTTP and HTTPS.	NaServer
get_transport_type()	Retrieves the transport used for this connection.	NaServer

set_port(port)	Overrides the default port for this server. If you also call `set_server_type()`, you must call it before calling `set_port()`.	NaServer
get_port()	Retrieves the port used for the remote server.	NaServer
use_https()	Determines whether HTTPS is enabled.	NaServer
set_server_cert_verification(enable)	Enables or disables server certificate verification by the client. Server certificate verification is enabled by default when the style is set to CERTIFICATE. Host name (CN) verification is always enabled during server certificate verification.	NaServer
is_server_cert_verification_enabled()	Determines whether server certificate verification is enabled or not. Returns `True` if it is enabled; otherwise, it returns `False`.	NaServer
set_client_cert_and_key (cert_file, key_file = nil, key_passwd = nil)	Sets the client certificate and key files that are required for client authentication by the server using certificates. If the key file is not defined, then the certificate file is used as the key file.	NaServer
set_ca_certs (ca_file)	Specifies the certificates of the CAs that are trusted by this application and that will be used to verify the server certificate.	NaServer
set_hostname_verification (enable)	Enables or disables host name verification by the client during server certificate verification.	NaServer
is_hostname_verification_enabled ()	Determines whether host name verification is enabled or not. Returns `True` if it is enabled; otherwise, it returns `False`.	NaServer
child_add_string_encrypted(name, value, key = nil)	Same as `child_add_string`, but encrypts value with key before adding the element to the current object. This is only used at present for certain key exchange operations. Both client and server must know the value of key and agree to use this routine and its companion, `child_get_string_encrypted()`. The default key is used if the given key is `None`.	NaElement
child_get_string_encrypted (name, key = nil)	Gets the value of child named `name`, and decrypts it with `key` before returning it. The default key is used if the given key is None.	NaElement
toEncodedString(self)	Encodes the string embedded with special characters such as &,<,>. This is mainly useful when passing string values embedded with special characters such as &,<,and > to API. For example : `server.invoke("qtree-create","qtree","abc<qt0","volume","vol0")`	NaElement

Table 4) Function details for Perl language.

API name	Description	Class name
new($filer, $majorversion, $minorversion)	Creates a new connection to filer `$filer`. Before use, you either need to set the style to `hosts.equiv` or set the user name (always `root` at present) and password with `set_admin_user()`.	NaServer

yle($style)	• Passes in LOGIN to cause the server to use HTTP simple authentication with a username and password. • Passes in HOSTS to use the `hosts.equiv` file on the filer to determine access rights (the user name must be `root` in that case). • Passes in CERTIFICATE to use certificate-based authentication with the Unified Manager server. **Note:** If `$style = CERTIFICATE`, you can use certificates to authenticate clients who attempt to connect to a server without the need of user name and password. This style internally sets the transport type to HTTPS. Verification of the server's certificate is required in order to properly authenticate the identity of the server. Server certificate (with host name) verification is enabled by default using this style. You can disable the server certificate (with host name) verification by using `set_server_cert_verification()` and you can disable only host name verification by using `set_hostname_verification()`.	NaServer
get_style()	Gets the authentication style.	NaServer
set_admin_user($user, $password)	Sets the admin user name and password. At present, `$user` must always be `root`.	NaServer
set_server_type($type)	Passes in one of these keywords: FILER, DFM, or OCUM to indicate whether the server is an ONTAP storage system or Active IQ Unified Manager server. If you also use `set_port()`, call `set_port()` after calling this routine. The default value is FILER.	NaServer
get_server_type()	Gets the type of server to which this server connection applies.	NaServer
set_transport_type($scheme)	Overrides the default transport type. The valid transport types are currently HTTP and HTTPS.	NaServer
get_transport_type()	Retrieves the transport used for this connection.	NaServer
set_port($port)	Overrides the default port for this server. If you also call `set_server_type()`, you must call it before calling `set_port()`.	NaServer
get_port()	Retrieves the port used for the remote server.	NaServer
use_https()	Determines whether HTTPS is enabled.	NaServer
set_server_cert_verification()	Enables or disables server certificate verification by the client. Server certificate verification is enabled by default when the style is set to CERTIFICATE. Host name verification is enabled by default during server certificate verification.	NaServer
is_server_cert_verification_enabled()	Determines whether server certificate verification is enabled or not. Returns `1` if it is enabled; otherwise, it returns `0`.	NaServer
set_client_cert_and_key()	Sets the client certificate and key files that are required for client authentication by the server using certificates. If key file is not defined, then the certificate file will be used as the key file.	NaServer

set_ca_certs()	Specifies the certificates of the CAs that are trusted by this application and that will be used to verify the remote server certificate.	NaServer
set_hostname_verification( )	Enables or disables host name verification by the client during server certificate verification.	NaServer
is_hostname_verification_e nabled()	Determines whether host name verification is enabled or not. Returns 1 if it is enabled; otherwise, it returns 0.	NaServer
verify_server_certificate()	Subroutine which verifies the common name in the server certificate against the given host name. This subroutine returns `undef` on success.	NaServer
set_sslv3($enable)	Enables or disables the SSLv3 protocol for use over HTTPS transport. By default, the SSLv3 protocol is disabled.	NaServer
is_sslv3()	Determines whether the SSLv3 protocol is enabled for use over HTTPS transport. Returns 1 if SSLV3 is enabled; otherwise, it returns 0.	NaServer
child_add_string_encrypte d($name, $value, $key)	Same as `child_add_string`, but encrypts `$value` with `$key` before adding the element to the current object. **Note:** This is only used at present for certain key exchange operations. Both the client and server must know the value of `$key` and agree to use this routine and its companion, `child_get_string_encrypted()`. The default key is used if the given key is `undef`.	NaElement
child_get_string_encrypted ($name, $key)	Gets the value of child `$name` and decrypts it with `$key` before returning it. The default key is used if the given key is `undef`.	NaElement
toEncodedString()	Encodes the string embedded with special characters such as &,<,and >. This is mainly useful when passing string values embedded with special characters such as &,<,and > to API. For example: `$server->invoke("qtree-create","qtree","abc<qt0",volume,"vol0");`	NaElement

Table 5) Function details for Java language.

API name	Description	Class name
setApiVersion(int majorVersion, int minorVersion)	Sets the API version for requests.	NaServer
synchronized void setKeepAliveEnabled(bool ean enabled)	Turns HTTP keep-alives on or off. The default is Off. If keep-alives are enabled, you must call `close()` when you are done using this object.	NaServer
synchronized void setPort(int tcpPort)	Sets the TCP port used for API invocations on the server. The following are the default ports on the storage system used for ONTAP API communication: • HTTP transport: 80 • HTTPS transport: 443 The following are the default ports on the Active IQ Unified Manager 5.2 or earlier servers for API communication:	NaServer

	• HTTP transport: 8080 • HTTPS transport: 8488 The following is the only supported port on the Active IQ Unified Manager 6.0 or later servers for API communication: • HTTPS transport: 443 **Note:** Use the above port numbers as input to this API if the server has default port settings.	
getPort()	Gets the TCP port used for API invocations on the server.	NaServer
getServerType()	Gets the server type, which is one of the `SERVER_TYPE_` values.	NaServer
synchronized void setServerType(int serverType)	Sets the URI and TCP port appropriate for the given server type.	NaServer
getTransportType()	Gets the transport type, which is one of the `TRANSPORT_TYPE_` values.	NaServer
synchronized void setTransportType(int transportType)	Sets the transport type. After setting the transport type, call the `setPort()` to set the port number used for that transport type.	NaServer
setAdminUser(String login, String password)	Sets the login and password used for authenticating when an API is invoked.	NaServer
setStyle(int style)	Sets the authentication style for subsequent API authentications.	NaServer
getStyle()	Gets the current authentication style.	NaServer
setKeyStore(String keyStore, String keyStorePasswd)	Sets the location of the keystore file where the client certificate and the key reside.	NaServer
setKeyStoreType(int keyStoreType)	Sets the keystore type for client certificates.	NaServer
enableServerCertVerification()	Enables the server certificate verification by the client. This method internally enables host name verification. The server certificate verification is enabled by default when the authentication style is set to `STYLE_CERTIFICATE`. Use `setTrustStore()` method to specify the truststore containing the certificates of the trusted CAs required for authenticating the server. You can disable the host name verification by using `disableHostnameVerification()`. **Note:** You need to set the transport type to HTTPS (`TRANSPORT_TYPE_HTTPS`) before calling this method.	NaServer
disableServerCertVerification()	Disables the server certificate verification by the client. This method internally disables host name verification. **Note:** You need to set the transport type to HTTPS (`TRANSPORT_TYPE_HTTPS`) before calling this method.	NaServer
isServerCertVerificationEnabled()	Determines whether the server certificate verification is enabled by the client.	NaServer

enableHostnameVerification()	Enables host name verification by the client during server certificate verification. Host name verification is enabled by default when the authentication style is set to `STYLE_CERTIFICATE`. Host name verification ensures that the host name to which the client connects matches the host name (CN name) in the certificate that the server sends back as part of the SSL connection. For host name verification, the application must specify the host name (CN name) instead of the IP address it is connecting to the server in NaServer constructor. Server certificate verification must be enabled before using this method.	NaServer
disableHostnameVerification()	Disables host name verification by the client during server certificate verification.	NaServer
isHostnameVerificationEnabled()	Determines whether host name verification is enabled by the client during server certificate verification.	NaServer
setTrustStore(String trustStore)	Sets the default location of the truststore containing the certificates of the trusted CAs required for authenticating the server.	NaServer
enableTLS()	Enables the TLS protocol for use on connection over HTTPS transport. **Note:** By default, the TLS protocol is enabled.	NaServer
disableTLS()	Disables the TLS protocol for use over HTTPS transport. **Note:** By default, the TLS protocol is enabled.	NaServer
isTLSEnabled()	Determines whether the TLS protocol is enabled for use over HTTPS transport.	NaServer
enableSSLv3()	Enables the SSLv3 protocol for use over HTTPS transport. **Note:** By default, the SSLv3 protocol is disabled.	NaServer
disableSSLv3()	Disables the SSLv3 protocol for use over HTTPS transport. **Note:** By default, SSLv3 protocol is disabled.	NaServer
isSSLv3Enabled()	Determines whether the SSLv3 protocol is enabled for use over HTTPS transport.	NaServer
setEncryptedContent(String content)	Encrypts the given value and sets the encrypted value as the content of this element.	NaElement
addNewEncryptedChild(String name, String content)	Encrypts the data contained in content. Adds a new child element with a given name `ant_sty;encrypted` content.	NaElement
getChildEncryptContent(String name)	Finds a child with the specified name, decrypts its string content, and returns the decrypted content. Returns null if it's unable to find child element with the specified name.	NaElement
getEncryptContent()	Decrypts and returns the value of content of this element.	NaElement
init()	Initializes the crypto engine. **Note:** Call this method before each unique encryption or decryption operation.	ARCFour

crypt(char[] input, char[] output)	Encrypts or decrypts. This can be called multiple times to encrypt a stream of data. **Note:** Be sure to call `init()` before each unique encryption or decryption operation.	ARCFour
encryptAndEncode(String input)	A convenience method that both encrypts and Base16 encodes a string using the default key.	ARCFour
decodeAndDecrypt(String input)	A convenience method that undoes the encoding and encrypts from `encryptAndEncode()` by using the default key.	ARCFour
encode(char[] bytes)	Encodes some bytes using Base16.	Base16
decode(String text)	Decodes a string of Base16 characters into its bytes.	Base16
setKeyStore(String keyStoreFile, String keyStorePasswd, String keyPasswd)	Sets the keystore details where the client certificate and the key resides.	AuthInfo
setKeyStoreType(String keyStoreType)	Sets the type of the client keystore file.	AuthInfo
setTrustStore(String trustStoreFile)	Sets the truststore file required for server authentication.	AuthInfo
getKeyStoreFile()	Gets the client keystore file.	AuthInfo
getKeyStorePasswd()	Gets the client keystore password.	AuthInfo
getKeyPasswd()	Gets the client private key password.	AuthInfo
getKeyStoreType()	Gets the type of the client keystore file.	AuthInfo
getTrustStoreFile()	Gets the location of the server truststore file.	AuthInfo
getTrustStoreType()	Gets the type of the server truststore file.	AuthInfo
setHost(String host)	Sets the host name that is required for server authentication.	AuthInfo
getHost()	Gets the host name that is required for server authentication.	AuthInfo
enableCBA()	Enables the certificate-based authentication mechanism.	AuthInfo
disableCBA()	Disables the certificate-based authentication mechanism.	AuthInfo
isCBAEnabled()	This flag indicates whether to use the certificate-based authentication mechanism.	AuthInfo
enableServerCertVerification()	Enables server certificate verification by the client.	AuthInfo
disableServerCertVerification()	Disables server certificate verification by the client.	AuthInfo
isServerCertVerificationEnabled()	A flag that indicates whether or not to use server certificate verification.	AuthInfo
enableHostnameVerification()	Enables host name verification by the client during server certificate verification.	AuthInfo
disableHostnameVerification()	Disables host name verification by the client during server certificate verification.	AuthInfo
isHostnameVerificationEnabled()	Determines whether host name verification is enabled by the client during server certificate verification.	AuthInfo
compare(AuthInfo a1, AuthInfo a2)	Compares the given authentication information objects according to their keystore and truststore properties.	AuthInfo

	Note: Returns `True` if both the properties match.	
encodeObject(java.io.Serializable serializableObject)	Serializes an object and returns the Base64-encoded version of that serialized object. **Note:** If the object cannot be serialized or there is another error, the method will return.	Base64
encodeBytes(byte[] source, int off, int len, boolean breakLines)	Encodes a byte array into Base64 notation.	Base64
encodeString(String s, boolean breakLines)	Encodes a string in Base64 notation with line breaks after every 75 Base64 characters. **Note:** You should only need to encode a string if there are non-ASCII characters in it such as many non-English languages.	Base64
readFile(String file, boolean encode)	Reads a file and either encodes or decodes it.	Base64
writeFile(byte[] data, String file, boolean encode)	Writes a byte array to a file either encoding or decoding it as specified.	Base64
encodeFromFile(String rawfile)	Simple helper method that Base64 encodes a file and returns the encoded string or if there was an error.	Base64
decodeFromFile(String encfile)	Simple helper method that Base64 decodes a file and returns the decoded data or if there was an error.	Base64
encodeToFile(byte[] rawdata, String file)	Simple helper method that Base64 encodes data to a file.	Base64
decodeToFile(byte[] encdata, String file)	Simple helper method that Base64 decodes data to a file.	Base64
decode(String s)	Decodes data from Base64 notation.	Base64
decodeToString(String s)	Decodes data from Base64 notation and returns it as a string. **Note:** This is equivalent to calling <code>new String(decode(s))</code>.	Base64
decodeToObject(String encodedObject)	Attempts to decode Base64 data and deserialize a Java object within. Returns null if there was an error.	Base64
decode(byte[] source, int off, int len)	Decodes Base64 content in byte array format and returns the decoded byte array.	Base64
getPath()	Gest the URI path for this HTTP request.	HTTPRequest
setBasicAuthorization(String username, String password)	Adds a header for HTTP basic authentication.	HTTPRequest
HTTPRequest read(InputStream in)	Creates a new HTTP request by reading data from a socket.	HTTPRequest

Table 6) Function details for .NET language.

API name	Description	Class name
SetApplicationName(String applicationName)	Sets the name of the client application that is using NMSDK core APIs.	NaServer

GetApplicationName()	Gets the name of the client application that is using NMSDK core APIs.	NaServer
SetAdminUser(string login, string password)	Sets the user name and password for the given server context, where the authentication style is `LOGIN_PASSWORD`. **Note:** The user that you specify should have administrative privileges on the server.	NaServer
Port	Gets or sets the TCP port used for API invocations on the server. The following are the default ports on the storage system used for ONTAP API communication: • HTTP transport: 80 • HTTPS transport: 443 The following are the default ports on Active IQ Unified Manager server 5.2 or earlier: • HTTP transport: 8080 • HTTPS transport: 8488 The following is the default port on Active IQ Unified Manager server 6.0 or later: • HTTPS transport: 443 **Note:** Use the above port numbers as input to this API if the server has default port settings.	NaServer
NaServer.SERVER_TYPE ServerType	Gets or sets the server type, which is one of the `SERVER_TYPE` values. Following are the supported values: • FILER to connect to a NetApp storage system. • DFM to connect to a Active IQ Unified Manager server 5.2 or earlier. • OCUM to connect to a Active IQ Unified Manager server 6.0 or later. • AGENT to connect to a NetApp host agent. **Note:** The default server type is FILER.	NaServer
NaServer.AUTH_STYLE Style	Gets or sets the authentication style, which is one of the `AUTH_STYLE` values. The following are the supported values: • `LOGIN_PASSWORD` • `RPC` • `HOSTSEQUIV` • If the style = LOGIN_PASSWORD, you have to furnish an administrator user name and password by using the `SetAdminUser()` API.	NaServer

	• If the style = RPC, your code should run on a Windows machine, so that a native Windows authentication based on a remote procedure call is used. • If the style = HOSTSEQUIV, the server context authenticates against the `/etc/hosts.equiv` file on the NetApp storage system. You do not have to set the administrator user name and password in this case. • If the style = CERTIFICATE, you can use certificates to authenticate clients who attempt to connect to a server without the need of a user name and password. This style internally sets the transport type to HTTPS. You can specify the certificate for the server to authenticate the client by using the `SetClientCertificate()` method. Verification of the server's certificate and host name is required in order to properly authenticate the identity of the server. Server certificate (with host name) verification is enabled by default using this style. You can disable host name verification by using `HostnameVerification` property. The server certificate verification can be disabled by using `ServerCertificateVerification` property. **Note:** HOSTSEQUIV and RPC styles are applicable for ONTAP APIs only and CERTIFICATE style is currently applicable for Unified Manager server API communication only.	
NaServer.TRANSPORT_TYPE TransportType	Gets or sets the transport type, which is one of the `TRANSPORT_TYPE` values. The default transport type is HTTP for `SERVER_TYPE.FILER` and `SERVER_TYPE.DFM`. For `SERVER_TYPE.OCUM`, HTTPS is the only supported transport type and is set by default.	NaServer
ServerCertificateVerification	This property enables or disables the server certificate verification by the client. You need to set the transport type to HTTPS before setting the value. **Note:** Host name verification is enabled by default during server certificate verification. The server certificate verification is enabled by default when the authentication style is set to CERTIFICATE. **Note:** The trusted root certificates to be used for server certificate verification must be added into the Trusted Root Certification Authorities folder in the Windows certificate store.	NaServer
HostnameVerification	This property enables or disables the host name verification during server certificate verification. Server certificate verification must be enabled before setting the value.	NaServer

	Host name verification is enabled by default when the authentication style is set to CERTIFICATE. Host name verification ensures that the host name to which the client connects matches the hostname (CN name) in the certificate that the server sends back as part of the SSL connection. For host name verification, the application must specify the host name (CN name) instead of the IP address it is connecting to the server in NaServer constructor.	
SetClientCertificate(String certificate)	Sets the location of the client certificate for the server to authenticate during the SSL connection.	NaServer
AddNewEncryptedChild(String name, String content)	Encrypts the data contained in content. Adds a new child element with a given name and encrypted content.	NaElement
GetChildEncryptContent(String name)	Finds a child with the specified name, decrypts its string content, and returns the decrypted content.	NaElement
GetEncryptContent()	Decrypts and returns the value of this element's content.	NaElement
Encode(char[] bytes)	Encodes some bytes using Base16.	Base16
Decode(String text)	Decodes a string of Base16 characters into its bytes.	Base16
Init()	Initializes the crypto engine. **Note:** Call this method before each unique encryption or decryption operation.	ARCFour
Crypt(char[] input, char[] output)	Encrypts or decrypts. This can be called multiple times to encrypt a stream of data. **Note:** Be sure to call `init()` before each unique encryption or decryption operation.	ARCFour
EncryptAndEncode(String input)	Convenience method that both encrypts and Base16 encodes a string using the default key.	ARCFour
DecodeAndDecrypt(String input)	Convenience method that undoes the encoding and encrypt from `encryptAndEncode()` by using the default key.	ARCFour

Table 7) Function details for C language.

API name	Description	Class name
shttpc_new(shttpc_type_t sh_type, int timeout)	Creates a socket (IPv4 socket only) and assigns it to `sh_socket` member. Caller must call `shttpc_delete` to close the socket and to release memory.	shttpc.c
shttpc_new_ipv6(shttpc_type_t sh_type, int timeout, struct addrinfo* AddrInfo)	Creates an IPv4/IPv6 socket based on AddrInfo. Caller must call `shttpc_delete` to close the socket and to release memory.	shttpc.c
shttpc_delete(shttpc_t sock)	Closes the socket and deletes the `shttpc` object.	shttpc.c
shttpc_setsock_timeout(shttpc_t sock)	Sets socket timeout and also sets socket options.	shttpc.c
shttpc_can_retry_on_error(int rc)	Checks whether we can retry on a non-fatal error.	shttpc.c
shttpc_connect_status(int rc)	Extracts the connection error status.	shttpc.c

shttpc_get_connect_error (shttpc_t sock, uint64_t endtime, struct timeval *tv)	Extracts the connection error details.	shttpc.c
shttpc_certificate_passwd_cb(char *buf, int size, int rwflag, void *passwd)	Default password callback called when loading a PEM certificate with encryption.	shttpc.c
shttpc_verify_server_certificate(SSL *ssl, cert_auth_info *cert_info)	Function that verifies the common name in the peer certificate against the given host name. Returns `0` if CN matches against the given host name; otherwise, it returns `-1`.	shttpc.c
shttpc_get_verify_cert_error_string(long err_no)	Returns a human readable error string for the given certificate verification error.	shttpc.c
shttpc_connect_ssl_with_cert(shttpc_t sock, uint64_t endtime, struct timeval *tv, cert_auth_info *cert_info)	Establishes an SSL connection. Uses OpenSSL libraries. This function is a wrapper over `shttpc_connect_ipv6()`, which takes an additional certificate details as input.	shttpc.c
shttpc_connect(shttpc_t sock, const struct sockaddr * addr, int addrlen)	Connects to a host using plain TCP/SSL. Uses OpenSSL libraries to connect using SSL.	shttpc.c
shttpc_connect_ipv6_with_cert(shttpc_t sock, struct addrinfo * addr, size_t addrlen, cert_auth_info *cert_info)	This function is a wrapper over `shttpc_connect_ipv6()`, which takes an additional certificate details as input.	shttpc.c
shttpc_connect_ipv6(shttpc_t sock, struct addrinfo * addr, size_t addrlen)	Connects to a host using plain TCP/SSL. Uses OpenSSL libraries to connect using SSL.	shttpc.c
shttpc_read(shttpc_t sock, void * buf, size_t len)	Reads the socket to the buffer `buf`. The maximum buffer size is len to connect using SSL.	shttpc.c
shttpc_write(shttpc_t sock, const void * buf, size_t len)	Writes the data to the socket. It keeps trying until whole data is written.	shttpc.c
http_parse_url(const char * url, http_url_t * purl)	Splits apart a URL into its components.	http.c
http_free_url(http_url_t * purl)	Unallocates space allocated in `http_parse_url()`.	http.c
bool_t http_write(shttpc_t sock, const void * line, ssize_t len)	Writes a line to a socket. Returns `True` on success.	http.c
bool_t http_print(shttpc_t sock, const char * fmt, ...)	Formats and writes a line to a socket. Returns `True` on success.	http.c
bool_t http_getline(shttpc_t sock, char *line, int len)	Read a line from a socket into a buffer. Strips off the `\r\n` at the end of each line. Returns `True` on success.	http.c
http_strip_headers(shttpc_t sock, stab_t * headers)	Read the headers, save them in a stab. The return value is negative if an error (such as ENOMEM) occurs. **Note:** On success, returns the HTTP status. If status is HTTP_OK, then the headers value is filled in, and you must free them (for example, with STAB_DELETE). If the status	http.c

	indicates an error, then you don't have to worry about freeing the headers.	
http_bind_socket(const shttpc_t socket, uint16_t reservedPort, const struct addrinfo *addrInfo)	Binds the socket to a reserved port and family (IPv4 or IPv6). This function tries to bind the socket to any one of the available reserved ports starting from the specified port to port 1. Returns `0` on success or `EPERM` error on failure.	http.c
http_open_socket_reserved_ex_wt_wcert(const char *host, uint16_t port, shttpc_t *socketP, bool_t reserved, shttpc_type_t conn_type, int timeout, cert_auth_info *cert_info, bool_t use_sslv3)	This function is a wrapper over `http_open_socket_reserved()`, which takes an additional certificate details as input.	http.c
bool_t http_get_auth(const AuthInfo * auth, char **encoded_str)	Gets the authentication message.	http.c
http_post_request(shttpc_t sock, const char * url, const AuthInfo * auth_info, const void * post_data, size_t post_data_len, stab_t * headersp)	`http_post_request` is the HTTP POST wrapper for `http_method_request()`. **Note:** For parameter details, see `http_method_request()`. The return value is negative if an error occurs and `0` if everything is OK.	http.c
http_get_request(shttpc_t sock, const char * url, const AuthInfo * auth_info, stab_t * headersp)	`http_get_request` is the HTTP GET wrapper for http_method_request(). **Note:** For parameter details, see `http_method_request()`. The return value is negative if an error occurs and `0` if everything is OK.	http.c
bool_t http_read_chunk(shttpc_t sock, char ** pbuf, size_t * pread)	Reads one chunk of data from the server. `pbuf` contains the data that is read; pread contains the number of bytes that is read from the socket. Returns `True` on success and `False` on failure.	http.c
na_startup(char * errbuff, int errbuffsize)	A one-time startup to prepare for API.	na.c
na_startup_without_ntapadmin(char * errbuff, int errbuffsize)	A one-time startup to prepare for API and network things. This function internally calls `na_startup` API, which disables loading of the `ntapadmin` library. This is used for RPC communications on Windows systems; on UNIX systems, it simply calls `na_startup` API.	na.c
na_server_set_timeout(na_server_t *s, int timeout)	Sets a timeout on the server when making a connection or when reading and writing data from a socket.	na.c
na_server_set_server_type (na_server_t *s, na_server_type_t type)	Passes in one of these keywords: FILER, DFM, or OCUM to indicate whether the server is a storage system (filer) or an Active IQ Unified Manager server. If you also use `set_port()`, call `set_port()` after calling this routine. The default value is FILER.	na.c
na_server_set_style(na_server_t *s, na_style_t style)	• Pass in LOGIN to cause the server to use HTTP simple authentication with a user name and password.	na.c

	• Pass in HOSTS to use the `hosts.equiv` file on the filer to determine access rights (the user name must be `root` in that case). • Pass in CERTIFICATE to use certificate-based authentication with the Unified Manager server. **Note:** If style = CERTIFICATE, you can use certificates to authenticate clients who attempt to connect to a server without the need of a user name and password. This style internally sets the transport type to HTTPS. Verification of the server's certificate is required in order to properly authenticate the identity of the server. Server certificate verification is enabled by default using this style and server certificate verification always enables host name verification. You can disable server certificate (with host name) verification by using `set_server_cert_verification()`.	
na_server_set_transport(na_server_t *s, na_server_transport_t transport, const union zfd_setopt *transportarg, int port)	Override the default transport type. The valid transport types are currently HTTP and HTTPS.	na.c
na_server_transport_t na_server_get_transport_type(na_server_t * s)	Retrieves the transport type used for this connection.	na.c
na_server_set_port(na_server_t *s, int port)	Overrides the default port for this server. If you also call `set_server_type()`, you must call it before calling `set_port()`.	na.c
na_server_get_port(na_server_t * s)	Retrieve the port used for the remote server.	na.c
na_server_close(na_server_t * s)	Closes the connection from server.	na.c
na_encrypt_basic(const char *key, const char* input, char * output, size_t nbytes)	Takes an input value (a series of bytes) and produces an encrypted version of it.	na.c
na_elem_t* na_child_add_string_encrypted(na_elem_t * n, const char * name, const char * contents, const char *key)	Same as `child_add_string`, but encrypts `value` with key before adding the element to the current object. This is only used at present for certain key exchange operations. Both the client and server must know the value of key and agree to use this routine and its companion, `child_get_string_encrypted()`. The default key will be used if the given key is `None`.	na.c
char * na_child_get_string_encrypted(na_elem_t * n, const char * name, const char *key)	Gets the value of the child named `name` and decrypts it with key before returning it. The default key is used if the given key is `None`.	na.c
na_elem_t * na_zapi_get_result(na_elem_stack_t * elemStack)	Gets the result of the invoked zapi.	na.c

na_server_init_cert_info(na_server_t *srv)	Initializes the memory for certificate authentication structure for the given server context.	na.c
na_server_set_client_cert_and_key(na_server_t *srv, const char *cert_file, const char *key_file, const char *key_passwd)	Sets the client certificate and key files that are required for client authentication by the server using certificates. If key file is not defined, then the certificate file is used as the key file.	na.c
na_server_set_server_cert_verification(na_server_t *srv, int enable)	Enables or disables server certificate verification by the client. Server certificate verification is enabled by default when style is set to CERTIFICATE. Host name (CN) verification is enabled during server certificate verification. Host name verification can be disabled by using the `set_hostname_verification()` API.	na.c
na_server_is_server_cert_verification_enabled(const na_server_t *srv)	Determines whether server certificate verification is enabled or not. Returns `True` if it is enabled; otherwise, it returns `False`.	na.c
na_server_set_ca_certs(na_server_t *srv, const char *CAfile)	Specifies the certificates of the CAs that are trusted by this application and that will be used to verify the server certificate.	na.c
na_server_set_hostname_verification(na_server_t *srv, int enable)	Enables or disables host name verification by the client during server certificate verification.	na.c
na_server_is_hostname_verification_enabled(const na_server_t *srv)	Determines whether or not host name verification is enabled. Returns `True` if it is enabled; otherwise, it returns `False`.	na.c
na_server_set_sslv3(na_server_t *srv, int enable)	Enables or disables the SSLv3 protocol for use over HTTPS transport. By default, the SSLv3 protocol is disabled.	na.c
na_server_is_sslv3_enabled(na_server_t *srv)	Determines whether the SSLv3 protocol is enabled for use over HTTPS transport. Returns `1` if SSLV3 is enabled; otherwise, it returns `0`.	na.c
do_encrypt_test(char * input, int nbytes)	Checks to make sure a password that is encrypted twice comes out with the same value with which it started.	na_test.c
main_encryptalot(int argc, char* argv[])	Encryption test with random password values–you can control the number of passwords and their length.	na_test.c
nc_api_error_t nc_api_set_transport(nc_api_transport_t transport, const zfd_setopt_t *opt)	Sets the transport type. Possible values are: • `NC_API_TRANSPORT_HTTP` • `NC_API_TRANSPORT_HTTPS`	nc_api.c
nc_api_error_t nc_api_read_file(const char *host, int port, const AuthInfo *auth, const char *file, char **out)	Copies the specified remote file to a buffer. Caller must free the memory of the specified file and buffer when finished.	nc_api.c

Where to find additional information

To learn more about the information that is described in this document, review the following documents and/or websites:

- NetApp Product Documentation
  https://www.netapp.com/support-and-training/documentation/

Version history

Version	Date	Document version history
Version 1.0	November 2022	Initial release.
Version 2.0	May 2023	Update NMSDK logging details
Version 3.0	May 2024	Update NMSDK Logging integration scripts Update NMSDK Linux binaries code signing verification.