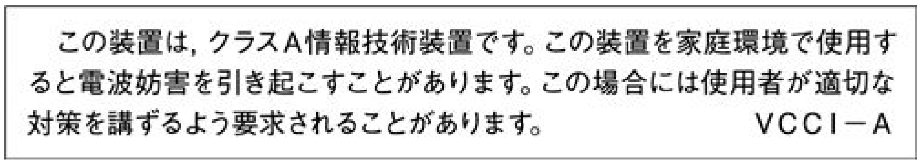
CommandCenter Secure Gateway WS-API Programming Guide

September 2024
API Version 1.7.0

FCC Information

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a commercial installation. This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. Operation of this equipment in a residential environment may cause harmful interference.

VCCI Information (Japan)



この装置は，クラスＡ情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。　VCCI－A

Raritan is not responsible for damage to this product resulting from accident, disaster, misuse, abuse, non-Raritan modification of the product, or other events outside of Raritan's reasonable control or not arising under normal operating conditions.

If a power cable is included with this product, it must be used exclusively for this product.

Contents

Introduction

Web Services API uses standardized Web Services technologies to allow a client machine to perform node, power, user, and logging management services.

This client is independent of the CC-SG, but aims to provide the same capabilities that the CC-SG's HTML-based Access Client provides, through use of the API and a TCP/IP network.

This client works independent of the Client Certificate Authentication. Enabling the Client Certificate does not affect the WS-API feature.

This SDK allows Independent Software Vendors (ISVs) and Raritan customers to build an application using a development environment compatible with SOAP, such as .NET and Java.

In This Chapter

Change Log

The following changes have been made in API version 1.6.0 in association with CC-SG v11.0:

- setUserPassword allows updating the password of managed devices.
- getInterfaceURL has some document fixes where unused inputs were removed.
- getNodeByAttribute retrieves a node by an attribute name having the specified value.
- getReportDocument retrieves accounting reports on resources like device ports.

The following changes have been made in API version 1.5.0 in association with CC-SG v10.0:

- getAllUsers has some minor bug fixes and no longer returns remote users that have not accessed the CC-SG.
- importCSV can now import categories, nodes, and power strips.
- Updated C# example for Visual Studio 2022.
- New shell client example.

The following changes have been made in API version 1.4.0 in association with CC-SG v9.0:

- runReport no longer creates scheduled reports.
- Authentication and Authorization Services:
  - Data Types:
    - ConnectionInfo (on page 14)
    - InterfaceConnectionInfo (on page 15)
  - Services:
    - closeInterfaceConnections (on page 17)
    - getInterfaceConnections (on page 17)

The following changes have been made in API version 1.3.0 in association with CC-SG v8.0:

- Common Data Types:
  - Common Data Types (on page 10)
- System Management Services:
  - Data Types: SystemInfo (on page 12)
  - Services:
    - importCSV (on page 13)
    - getImportStatus (on page 13)
- Authentication and Authorization Services:
  - Data Types: SessionInfo (on page 14)
  - Services:
    - getSessions (on page 15)
    - forceSignOff (on page 16)
- Device Management Services:
  - Data Types: DeviceStatus (on page 18)
  - Services: getDeviceStatus (on page 19)

The following change in CC-SG v7.0 affects your WS-API application:

- The certificate is updated in CC-SG 7.0. If your client uses a truststore, then it will need to be updated.

The following changes have been made in API version 1.2.0

- New data:
  - DeviceData has new fields. See DeviceData (on page 18).
    - String model - The device's model.
    - Integer portCount - The port count reflects the number of feature ports such as KVM, serial, power, and outlet.

The following changes have been made in API version 1.1.0

- New services:

    - getDevice() - See getDevice (on page 19).
    - deleteDevice() - See deleteDevice (on page 19).
    - getNodeStatus() - See getNodeStatus (on page 27).
- New behaviour:
    - Wild card behavior is more uniform now. See Standard Wild Card Search Field (on page 11).

Connecting to CC-SG

► *To connect to CC-SG:*

1. Establish client configuration on CC-SG Admin Client. See Add Web Services API Client Configuration on CC-SG (on page 7)
2. Download client and server/CC-SG certificates to the client machine for use by the client. To connect to the CC-SG using SOAP over port 9443, the client must first manually install the x.509 digital client certificate that is generated using the CC-SG Admin Client. CC-SG contains a server certificate that will be retrieved using SSL/TLS APIs implemented in the client application to establish a trust relationship. See Add Web Services API Client Configuration on CC-SG (on page 7) for instructions on creating the client certificate. See Certificates (on page 9).

Details:

- Open the PKCS #12 file using the password and store the client certificate public and private keys in the keystore that is accessible to the WS-API client application.
- Connect on TCP Port 9443 to the CC-SG's IP address and exchange certificates using the SSL/TLS protocol.

Note: CC-SG verifies that the client IP address matches the address set within the client configuration on the CC-SG.

- Accept the self signed server certificate from CC-SG. This may require special handling on Java. See Certificates information for Java users (on page 42)

3. Download WSDL files from the CC-SG. You can use a web browser or a simple client like wget to access the WSDL URLs. See WSDL URLs (on page 9)
4. Choose a WS client library for your target language.
5. Use the tools provided with your chosen WS client library to generate stubs in your target language. Each stub should be a complete web service operation such that all that you must do is call the web service as a method/function with the appropriate parameters.
6. Begin writing the client. Call the signOn() service found in AuthenticationAndAuthorizationService. If successful, the signOn() returns a session ID that you must provide for some services. Access the other services as needed. Call signOff() to end your session when your application finishes.

Note: The signOn() service is only required for services that take the session ID as a parameter. signOff() is only needed if signOn() is called.

Add Web Services API Client Configuration on CC-SG

You must accept the End User Agreement before adding a Web Services API client to CC-SG. See the CC-SG Web Services API Guide for details on using the API.

► *To add a Web Services API:*

1. Select Access > Add Web Services API. This option is available only for users with the CC Setup and Control Privilege.
2. Read the End User Agreement.
   - You can copy and paste the text to save it, or choose Secure Gateway > Print.
   - After you complete configuration, this agreement will also be available in the Access menu.
3. Click Accept. The New Web Services API Configuration window opens.
4. Type in the data requested about your web services client.
   - Web Services Client Name: Maximum 64 characters.
   - License Key: Your license key from Raritan. Each CC-SG unit must have a unique license key.
   - IP Address/Hostname: Maximum 64 characters.
   - HTTPS Web Services Port: Read-only field. CC-SG uses port 9443 when trust establishment is generated.
   - Licensed Vendor Name: Maximum 64 characters.
5. Generate a self-signed certificate.
   a. Encryption Mode: If Require AES Encryption between Client and Server is selected in the Administration > Security > Encryption screen, AES-128 is the default. If AES is not required, DES 3 is the default.
   b. Private Key Length: 2048 is the default. Choose 1024 through 4096.
   c. Validity Period (days): Maximum 4 numeric characters.
   d. Country Code: CSR tag is Country Name.
   e. State or Province: Maximum 128 characters. Type in the whole state or province name. Do not abbreviate.
   f. City/Locality: CSR tag is Locality Name. Maximum 128 characters.
   g. Registered Company Name: CSR tag is Organization Name. Maximum 64 characters.
   h. Division/Department Name: CSR tag is Organization Unit Name. Maximum 64 characters.
   i. Fully Qualified Domain Name: CSR tag is Common Name.
   j. Administrator Email Address: Type in the email address of the administrator who is responsible for the certificate request. Maximum 256 characters.
   k. Challenge Password: Maximum 64 characters.
   l. Subject Alternative names (SAN): Multiple entries can be added separated by commas.

   Note: The Challenge Password is used internally by CC-SG to generate the certificate. You do not need to remember it.

   m. Password: Enter a keystore password. Use this password to open the .P12 file that you will save in step 7. If you copy the generated certificate and import into your own keystore instead, you do not need to remember this keystore password.
6. Click Generate Certificate. The text appears in the Certificate box.
7. Click Save to File to save the certificate to a .P12 file. Or, copy the generated certificate and import it into your own keystore.
8. Click Add to save your changes.

WSDL URLs

The web services of the CC-SG are categorized based on their associated CC-SG functionality. You can retrieve the WSDL that you need to develop your own applications from your CC-SG using a web browser. You can find the URL in this document at the beginning of each service category in the API Definitions (on page 10) section.

- System Management Services (on page 11)
- Authentication and Authorization Services (on page 14)
- Device Management Services (on page 17)
- Node Management Services (on page 20)
- User Management Services (on page 29)
- Logging Management Services (on page 32)
- Category Management Services (on page 36)

Certificates

The CC-SG's Web Services require mutual certificates such that both the CC-SG and the WS client present a certificate. Upon CC-SG WS client configuration, the CC-SG will know to recognize the client by the generated client certificate. The client also needs to recognize the CC-SG certificate and present the generated client certificate to the CC-SG.

The CC-SG WS-API server certificate (port 9443) is shared with the CC-SG's HTTPS certificate (port 443) and is configurable using the Admin Client.

The client must accept this certificate, however, a typical WS client would not be designed to present a certificate acceptance dialog to the user. One can simply use a trust store to contain the CC-SG's server certificate thereby telling the client certificate library to trust the CC-SG.

Obtain the CC-SG's certificate then create a new trust store or add it to an existing one. The WS client must be able to access the trust store to be able to communicate with the CC-SG.

If you don't want to manually add the certificate to a trust store, you can make provisions in the client source to always trust the CC-SG or to save the CC-SG's certificate into the trust store automatically.

Remotely Authorized Users

Users authorized via remote servers require some special handling in CC-SGs Web Services. Only AD users can be remotely authorized.

1. The signOn() service takes the plain user name just as the remote user would enter it on the CC-SG's login page.
2. Whenever a WS client accesses a service that takes a user name as a parameter (including signOff()) and the targeted user is a remotely authorized user, the user name needs to have the remote server's module name appended as follows: USER@MODULE

USER is the plain user name and MODULE is the name that the administrator gave the remote module configuration in CC-SG.

API Definitions

In This Chapter

Conventions

The following conventions are used within this document.

- String sessionID - The authentication token that was originally assigned to the user via the SignOn method. Whenever you see a parameter named sessionID, it refers to this definition.

  Note: The session ID will be invalidated if no session activity is detected for 5 minutes.

- String interfaceID – CC-SG generated unique identifier of an interface.
- String username - The name of a CC-SG user account.

Common Data Types

1. Exceptions

Each CC-SG WS-API service returns an exception upon error. If the error is specific to the service, the exception will be defined as follows:

- Elements
  - String code – Simple definition of the error.
  - String message – Specific error message.

2. xsd:dateTime

Standard XML type used in Web services and based on ISO 8601. Your WS system will map it to a type appropriate for your target programming language.

3. KeyValue

Key-value pair

- Elements
  - String key
  - String value

Standard Wild Card Search Field

A parameter labeled as a Standard Wild Card Search Field can be a literal string or it can include wild cards (simplified regular expression matching) to expand results. If this field is the empty string (“”), it will match the empty string except where noted. If this field is null, the operation will ignore this field (equivalent to *).

Wild cards are characters that can be used in search strings to yield the following results:

- % and *: Match nothing or a string of any characters.
- _ and ?: Match any single character
- [-]: Match a single character from some range of characters. For example, [0-9] to match any single digit in that range.

System Management Services

This set of services is for general CC-SG settings and information.

http(s)://CC_IP_ADDRESS/CommandCenterWebServices/SystemManagementServicePort?wsdl

Data Types

SystemManagementException

All errors are returned via exception as defined under Common Data Types (on page 10).

ClusterStatus

Cluster state of both nodes.

- Elements
  - String name
  - String primaryAddress - Address of the primary node for identification.
  - String primaryState - The primary node can have the following states:
  - Standalone - There is no cluster configured.
  - Primary
  - String backupAddress - Address of the backup node for identification
  - String backupState - The backup node can have the following states:

    - Backup
    - Failed
    - Waiting
    - Joining
    - Unknown

SystemInfo

General information about the CC-SG.

- Elements
    - String firmwareVersion – The version of the software running on the CC-SG.
    - String WSAPIVersion – The version of the WS API on the CC formatted as follows:
    - Major.Minor.Point
    - Major increments with big, new feature sets or changes to existing services. The client should be rebuilt based on the current WSDL.
    - Minor increments with small functionality additions. A client rebuild may not be necessary.
    - Point increments for patches and bug fixes. No client rebuilding needed.
    - String serial – The serial number identifying the unit.
    - String platform – Hardware identification.
    - xsd:dateTime date – The current date.
    - int nodesUsed – The number of nodes configured.
    - int nodesAvailable – The maximum number of nodes licensed for configuration.
    - ClusterStatus - clusterStatus.

ImportStatus

Describes the current status of an import request.

- Elements
    - String id – The unique task identifier.
    - String status – Current state of the import task. See messages for details.
    - Processing – The import is currently running.
    - Success – The import has completed, possibly with non-fatal errors.
    - Failure – The import has completed unsuccessfully.
    - KeyValue messages[] – Feedback from processing the import request data.

- key – Status of this message.
- ---- Error - A fatal problem.
- ---- Warning – A non-fatal issue.
- value – The message string in the following format with CSV line number when applicable:
- ---- [LINE_NUMBER:]MESSAGE

Services

getSystemInfo

Retrieve information about the CC-SG.

- parameters
  - String sessionID
- return value
  - SystemInfo

importCSV

Import configuration data in a background task. Use getImportStatus() or see the Admin client task manager for updates on the task status.

Permission: CC Setup And Control

- Parameters
  - String sessionID
  - String type – The type of CSV data to import corresponding to the types of imports in the Admin client. Case insensitive. Each type has an additional permission requirement.
  - Categories
  - ---- Permission: User Security Management
  - Devices
  - ---- Permission: Device, Port and Node Management
  - Nodes
  - ---- Permission: Device, Port and Node Management
  - Power Strips
  - ---- Permission: Device, Port and Node Management
  - Users
  - ---- Permission: User Management
  - String CSV – The CSV data to import.
- Return value
  - ImportStatus

getImportStatus

Retrieve the current status of an import task.

Permission: CC Setup And Control

- Parameters
    - String sessionID
    - String taskID – Task identifier for the import task.
- Return value
    - ImportStatus

Authentication and Authorization Services

This set of services is for logging into and out of CC-SG.

```
http://CC_IP_ADDRESS/CommandCenterWebServices/
AuthenticationAndAuthorizationServicePort?wsdl
```

Data Types

AuthenticationAndAuthorizationException

All errors are returned via exception as defined under Common Data Types (on page 10).

SessionInfo

Describes a user connected to the CC-SG.

- Elements
    - String id – The unique session identifier.
    - String user – The name of the account.
    - String groups[] – The names of the account's group memberships.
    - String address – The client's originating address.
    - String type – Login type.
    - xsd:dateTime startTime – When the session was created.

ConnectionInfo

An individual user's connection to an interface.

- Elements

    - String id – Connection identifier.
    - String sessionId – Session used to make the connection. See getSessions().
    - String mode – Method used to make the connection (Direct or Proxy).
    - String address - The client's originating address (can be different than that of SessionInfo when launching an interface via URL).
    - Calendar startTime – When the connection was made.

InterfaceConnectionInfo

An interface that has connections.

- Elements
    - String name – Interface's name
    - String nodeName – Name of the node containing the interface.
    - String deviceName – Name of the parent device (if any).
    - ConnectionInfo connections[]

Services

getSessions

Retrieve all sessions.

Requires User Management permission.

- Parameters
  - String sessionID
- Return value
  - SessionInfo[]

signOn()

This operation is used to login to CC-SG and authenticate with the CC-SG user database or an external database. This is the first method which should be used once a SSL/TLS session has been established. This operation achieves a SSO (Single Sign On) effect. The WS-API can continue to make requests without signing in again. The signon is used to authenticate any user credentials that is using the WS-API client as a proxy.

- parameters
  - String *username* - user name logging into the WS-API client
  - String *password* - associated password of the username being used by the WS-API client.
- return value
  - String sessionID

signoff()

Log out a WS API user (generated by signOn()). The application can have multiple users logged in.

- String *username* - user name that is logging out via the WS-API client
- String *sessionID*
- return value void

forceSignOff

Close the specified sessions.

Requires User Management permission.

There is a short delay from a successful sign off until the session is removed from the sessions list.

A Super User session cannot be forced to close.

This function will not close the caller's session (specified by sessionID). Use signOff() to close the caller's session.

- Parameters

    - String sessionID
    - String sessions[] - Session identifiers to close. If null or the first entry is * (i.e. {"*"}), then the CC-SG will close all sessions.
    - String message – Optional message to display to the user of the closed session. If null or empty, the CC-SG will display its default message.
- Return value
    - KeyValue[] - key is the session ID. value is the status message: "Success" means that the session is now closing; all other messages indicate an error.

closeInterfaceConnections

Close the specified interface connections (does not close parent sessions).

Requires Device, Port and Node Management permission.

A Super User connection cannot be closed by a non Super User.

- Parameters
    - String sessionID
    - String connections[] – Connection identifiers to close. If null or the first entry is * (i.e. {"*"}), then the CC-SG will close all connections.
- Return value
    - KeyValue[]- key is the connection ID. value is the status message: "Success" means that the connection is now closing; all other messages indicate an error.

getInterfaceConnections

Retrieve all the active interface connections.

Requires Device, Port and Node Management permission.

- Parameters
    - String sessionID
- Return value
    - InterfaceConnectionInfo[]

Unsupported Authentication and Authorization Services

The following Authentication and Authorization services are not supported and should not be used.

- authenticate()
- getAllUserGroups()
- getAllUserGroupsCount()

Device Management Services

This set of services is for getting information about devices that the CC-SG is managing.

http(s)://CC_IP_ADDRESS/CommandCenterWebServices/DeviceManagementServicePort?wsdl

Data Types

DeviceData

Contains information about a device.

- Elements
  - String name
  - String type – Product name.
  - String address – Host or IPv4 address.
  - String gateway – IPv4 gateway address.
  - String subnet – IPv4 subnet mask.
  - String addressIPv6 – IPv6 address.
  - String gatewayIPv6 – IPv6 gateway address.
  - String prefixLengthIPv6 – IPv6 routing data.
  - String model -- The device's model.
  - Integer portCount -- The port count reflects the number of feature ports such as KVM, serial, power, and outlet.
  - Integer discoveryPort – Discovery port.
  - Integer httpPort – HTTP port.
  - Integer httpsPort – HTTPS port.
  - Integer heartbeat – Heartbeat (seconds)
  - Boolean allowDirectDeviceAccess – Allow Direct Device Access feature.
  - String encryption – Type of encryption used for communications with this device.
  - String firmware – Firmware version on this device.
  - String serial – Serial number of this device.
  - String description

DeviceManagementException

All errors are returned via exception as defined under Common Data Types (on page 10).

DeviceStatus

Describes the current state of the appliance.

- Elements
  - String name
  - String status – Possible values:

- Up – Available for use.
- Down – Unavailable for use.
- Paused – Management of this appliance is paused.
- Locked – Unavailable because of appliance upgrade.

Services

getDevice

Retrieve one or more devices.

- Parameters
  - String sessionID
  - String name – A standard wild card search field. See Standard Wild Card Search Field (on page 11).
- Return value
  - DeviceData[]

deleteDevice

Stop management and remove the device from the CC.

- Parameters
  - String sessionID
  - String name
  - boolean force – Some situations, like the device being unavailable, require confirmation before the device can be deleted. Set this parameter to true to indicate confirmation.

getDeviceStatus

Retrieve the status of appliances.

- parameters
  - String sessionID
  - String name - Appliance name to find. A standard wild card search field. See Standard Wild Card Search Field
- return value
  - DeviceStatus[]

setUserPassword

This operation allows update of a user's password on a managed device.

Permission: "Device Configuration and Upgrade Management", or "Port and Node Management"

- parameters

- String sessionID
- String device – Name of the device to access
- String user – Name of the user to modify.
- String password – The user’s new password.

Node Management Services

This set of services is for modifying nodes in CC-SG.

```
http://CC_IP_ADDRESS/CommandCenterWebServices/
NodeManagementServicePort?wsdl
```

Data Types

AccessMethod

Description of a means of accessing a node.

- Elements
    - String methodName - the name of the application used for access. Some examples of the application name are "SSH Client (SSH)", “MPC”, "RemoteDesktop Viewer (RemoteDesktop)" and "iLO RemoteConsole".
    - String methodType - “inband” or “outband” - Inband refers to any interface that uses only the TCP/IP network to directly connect to the node. Outband is based on reaching a Raritan device product via TCP/IP and from the device connecting to the KVM or serial port of the node.
    - String InterfaceID - CC-SG generated string which uniquely identifies the interface within CC-SG
    - String InterfaceName - CC-SG generated string which identifies the interface within CC-SG
    - String applicationId - CC-SG generated string which uniquely identifies the access application type within CC-SG for out-of-band access designated for use within CC-SG. String is null if not applicable to the interface.
    - boolean userAuthorizedForMethod - value of the authorization for the user to access this interface with this application. If the user has permission, the value is TRUE.

AssociationData

Represents a category based label placed on the node.

- Elements
    - String category - The unique name identifying the category.
    - String element - The value from the category that labels this node.

InterfaceAvailabilityAndStatus

Describes the current state of the interface.

- Elements

- String interfaceID
- String availability – Text description of the availability field. For example, if an operation is still in progress, availability will indicate "Processing." Availability may be Idle, Busy, or Processing.
- String status – Text description of the status field. For example, Up or Down.

InterfaceData

Description of the node's attachment to the end point.

- Elements
    - String ip - host IP address for the interface. This field is filled in for in-band interfaces only; otherwise, it is the empty string.
    - String hostname - hostname for the interface (for in-band interfaces only) based on query using the supplied host IP address.
    - String portName i.e the name for the port for out-of-band interfaces only; otherwise it is the empty string.
    - String portID - the Raritan port ID for out-of-band interfaces. This is a unique generated value that occurs as part of configuration of a Raritan devices' ports. This field is empty for in-band interfaces.
    - String deviceName i.e the name of the Raritan device. This field is filled in only for out-of-band interfaces; otherwise, it the empty string.
    - String name -
    - String id - Unique identifier referenced by AccessMethod
    - String description - User description of the interface
    - String type - Function of the interface
    - Integer portNumber - Physical port number of the device.

NodeData

Description of a node's configuration.

- Elements

- String name
- InterfaceData[] interfacesData – The interfaces attached to this node
- AssociationData[] associations - Category values with which the node is associated

NodePowerStatus

Describes the power status of a node through its interfaces.

- Elements
  - PowerInterfaceStatus [] powerInterfaceStatus – Contains an entry for each of the node's power interfaces.

NodeStatus

Current status of a node's interfaces.

- Elements
  - String name
  - InterfaceAvailabilityAndStatus[] interfaces – The status of each of the node's interfaces.

PowerInterfaceStatus

Describes the interface and power operation statuses of a power interface.

- Elements
  - InterfaceAvailabilityAndStatus availabilityAndStatus – The general interface status.
  - String operation – The most recent power operation.
  - boolean inProgress – true if the operation is still running.
  - boolean successful – true if the operation has finished successfully. If both successful and inProgress are false, then the operation failed.
  - String failureReason – If the operation failed, then this field will typically contain a description of the failure.
  - xsd:dateTime timeStamp – The time that the CC-SG updated the power operation status.

URLObject

Components to form a URL to access the CC-SG.

- Elements

- String protocol - the protocol used - either http or https
- String port - the TCP port to be used for connecting to the interface: port 80 or port 443.
- String path - the path to the actual webservice servlet
- String tokenKey - the name of the property to be used for the token. The value is always "sessionID"
- String tokenValue - the actual property value corresponding to the tokenKey from above

Constructing a URL from URLObject

Combine the string elements of the returned URLObject *(italicized)* in the following order with other data (plain):

protocol + *://* + CCSG Address + : + *port* + *path* + *?* + *tokenKey* + = + *tokenValue*

Given the following URLObject data:

- *protocol* - http
- *port* - 80
- *path* - /CommandCenterWeb/index_frames.jsp
- *tokenKey* - sessionID
- *tokenValue* - 03AC4A3B1EE2CB665564BEB1ACAA8401

The constructed URL should look similar to this one:

http://10.0.0.101:80/CommandCenterWeb/index_frames.jsp?
sessionID=03AC4A3B1EE2CB665564BEB1ACAA8401

Note: A single question mark (?) delimits parameters from the document path. Parameters themselves are separated from one another using an ampersand (&). The path from getInterfaceURL() will already contain parameters, so you must append the sessionID using an ampersand delimiter in that case.

NodeManagementException

All errors are returned via exception as defined under Common Data Types (on page 10).

Services

getCCSGAppletURL

This operation retrieves the full URL to the CC-SG Admin Client applet in order to launch the main CC-SG client. When this URL is opened in a browser it will display the main CC-SG client, if the sessionID is valid, or the login screen, if the sessionID is invalid.

- parameters
  - String sessionID
- return value URLObject

getCCSGHTMLClientURL

This operation retrieves the full URL for the CC-SG Access Client HTML pages to launch the main access CC-SG page. When this URL is opened in a browser it will display the main access CC-SG page if the sessionID is valid, or the login page if the sessionID is invalid.

- parameters
  - String sessionID - authentication token granted for use by CCSG in SignOn()
  - return value URLObject

getInterfaceURL

This operation retrieves a URL used to connect directly to an interface and launch the associated client. This URL can be opened in a web browser to initiate the connection. The signOn method must have previously been called for this user.

- Parameters
  - String sessionID
  - String interfaceID CC-SG generated string which uniquely identifies the interface within CC-SG
  - String unused-deprecated and ignored
- return value:
  - URLObject

getAccessMethodsForNode

This operation retrieves all the available access methods (applications) for a given node in the form of an array. Each element in the array has an indicator to denote whether the passed in username has access to particular applications.

- parameters

    - String *nodeName* - name based on the configured name in CC-SG. The node name that the user enters in the CC-SG is not guaranteed to be unique so the system appends a number in parentheses to make the name unique. For example, the name “MyServer” becomes “MyServer(1) if another node is already using that name.
    - String *username* - CC-SG user for which the Access Methods are to be returned.
    - return value AccessMethod [ ]

getAccessMethodsForNodeByInterfaceID

This operation retrieves the interface information that defines a particular access method based on the unique interface ID.

- parameters
    - String *interfaceID*
    - String *username.* This service uses the username to determine whether that user has permission to access the node and, thereby, the AccessMethod (indicated by AccessMethod.userAuthorizedForMethod).
- return value
    - AccessMethod [ ]

getNodeByName

Retrieve nodes by name.

- parameters
    - String sessionID
    - String name - Node name to find. A standard wild card search field. See Standard Wild Card Search Field (on page 11).
- return value
    - NodeData[]

getNodeByAttribute

Retrieve a node by an attribute name having the specified value.

Permission: “Device, Port and Node Management”, “Node In-band Access”, or “Node Out-of-band Access”

- parameters

    - String sessionID
    - String attribute – Specify the search field. Case insensitive.
        - CIM Serial
    - String value – Find the node with its attribute matching this value.
- return value
    - NodeData[]

getNodeByInterfaceName

Retrieves nodes by the name of the interface.

- parameters
    - String sessionID
    - String interfaceName - Interface name to find. A standard wild card search field. See Standard Wild Card Search Field (on page 11).
- return value
    - NodeData[]

getNodeByAssociation

Find a node based on the category label applied to it.

- parameters

    - String sessionID
    - String category - The unique name identifying the category
    - String element - The value from the category that labels this node
- return value
    - NodeData[]

getNodesForUser

Retrieves all nodes available to the specified user.

- parameters
    - String *username*
- return value
    - NodeData [ ]

getNodeStatus

Retrieve the status and availability of every interface for a node.

- Parameters
    - String sessionID
    - String name – A standard wild card search field. See Standard Wild Card Search Field (on page 11).
- Return value
    - NodeStatus[]

addAssociationToNode

Associate the node with one or more category values

- parameters
    - String sessionID
    - String nodeName - The unique name of the node to modify
    - AssociationData[]associations
- return value
    - boolean true on success

deleteAssociationFromNode

Disassociate the node from one or more category values

- parameters

    - String sessionID
    - String nodeName - The unique name of the node to modify
    - AssociationData[]associations
- return value
    - boolean true on success

renameNode

This operation changes the name of the specified node to the new name.

- parameters
    - String sessionID
    - String currentNodeName - The name of the node that one wants to modify.
    - String newNodeName - The name to take the place of the current one.
- return value
    - boolean true

getNodePower

Returns the power status of each interface of the node, including the status of the latest power operation. The user must have permission to access the node.

In addition to the normal states, the availability state of each interface will be set to "Processing" if the state is pending a change because of a power operation.

- parameters
    - String sessionID
    - String nodeName – Return the status of the interfaces of the node with this name.
- return value
    - NodePowerStatus

setNodePower

Initiate a power operation on the interfaces of a node. This command requires the user to possess node permissions allowing power control of the node.

Note: Power operations are in progress after setNodePower() returns. The session must remain valid during the lifetime of the operation, otherwise, the operation will fail. Do not call signOff() until the operation is complete (see getNodePower()).

- parameters
    - String sessionID
    - String nodeName – The name of the node on which to operate.
    - String[]powerInterfaceID – An array of each of the node's interfaces on which to perform the power operation. Each interface must be a member of the node.

- String powerOperation – The power operation to perform on the interfaces. It must be one of the following strings, as supported by the specified power interface. All operations are not supported by all power interfaces. See Raritan's CommandCenter Secure Gateway Administrators Guide.
  - power on
  - power off
  - power cycle
  - graceful shutdown
  - suspend
- Integer sequenceInterval – The interval, in seconds, between successive operations of the specified power interfaces. Applicable to power on and power off only.
- String reasonForAccess – Text used to track access by user, required when Node Auditing is enabled for the user's group.

- return value
  - boolean true

User Management Services

This set of services is for adding, modifying, and removing users from the CC-SG database as is typically done by system administrators.

```
http(s)://CC_IP_ADDRESS/CommandCenterWebServices/
UserManagementServicePort?wsdl
```

Data Types

CCSGUser

The settings of a user on the CC-SG.

- Elements
  - String name – The unique user name used when logging into the CC-SG.
  - String password – The user's password (required if remoteAuthentication is false). This value will always be null when returned from getUser().
  - boolean loginEnabled – The user may use the account when true.
  - boolean remoteAuthentication – true if the user will be remotely authenticated.
  - boolean passwordExpirationEnabled – When true, the user will have to change their password periodically as indicated by passwordExpirationPeriod.
  - Integer passwordExpirationPeriod – The user will have to reset their password after this many days (required if passwordExpirationEnabled is true). This value will always be clear if this feature is disabled.
  - boolean forcePasswordChange - When true, the user will have to change their password on the next login. When the user is first added with addUser(), forcePasswordChange will be forced to true. The value of forcePasswordChange can be modified with editUser().
  - String fullName – The user’s full name to be used when generating reports and notifications to more easily identify the user. (optional)

- String emailAddress – (optional)
- String phoneNumber – (optional)
- String groups[ ] – An array of the name(s) of the group(s) to which this user is a member. These are the groups from which this user's permissions will be assigned.

UserManagementException

All errors are returned via exception as defined under Common Data Types (on page 10).

Services

getUser

Returns configuration data for the specified user.

- parameters
  - String sessionID
  - String userName - The login name of the desired user.
- return value
  - CCSGUser - The requested user.

getAllUsers

Returns an array containing the CCSGUser data for each local user defined on the CC-SG, similar to the Admin client’s All Users Data report.

- parameters
  - String sessionID
- return value
  - CCSGUser []

addUser

Add a new user configuration to the CC-SG.

- parameters
  - String sessionID
  - CCSGUser user – The new user's settings.
- return value
  - boolean true

editUser

Change an existing user's settings, excluding groups.

- parameters
  - String sessionID
  - CCSGUser user – The new user's settings.
- return value
  - boolean true

deleteUser

Remove the user with the provided name from the CC-SG.

Note: You cannot delete the SuperUser.

- parameters
  - String sessionID
  - String userName - The name of the user to delete.
- return value
  - boolean true

addUserToGroup

Assign a user to a group to control their access of the CC-SG.

Note: You cannot add users to or delete users from the SuperUser group.

- parameters
  - String sessionID
  - String userName - The name of the user to modify.
  - String[]groupName - An array of group names of which the user shall be a member.
- return value
  - boolean true

deleteUserFromGroup

Remove a user from a group to control their access of the CC-SG.

Note: If this operation deletes all of a user's groups, then the user itself shall be deleted.

- parameters
  - String sessionID
  - String userName - The name of the user to modify.
  - String[]groupName - An array of group names in which the user shall no longer be a member.
- return value
  - boolean true

Logging Management Services

This set of services is for retrieving log records from the CC-SG database.

```
http(s)://CC_IP_ADDRESS/CommandCenterWebServices/
LoggingManagementServicePort?wsdl
```

Data Types

ReportRecord

The components of a CC-SG log record. This data encompasses multiple types of logs such that not all elements will be populated for a particular record instance.

- Elements
  - Integer recordNumber – The ordinal number of the record relative to the requested report. Numbering begins at one.
  - xsd:dateTime entryDateTime – The date and time of the record.
  - String userName – The user to which the entry corresponds.
  - String userIPAddress – The IP address of the corresponding user.
  - String messageType – The message type of the report entry. The following are supported:
  - Access Audit
  - Access Connection
  - Authentication
  - Error
  - Power
  - Tasks
  - User Maintenance
  - String message – The message text describing the user activity.
  - String deviceName – The managed device the report entry corresponds to, provided for message type Access Connection.
  - String nodeName – The node the report entry corresponds to, provided for message type Access Connection.
  - String portName – The managed device port that the report entry corresponds to, provided for message type Access Connection.
  - String interfaceName – The node interface the report entry corresponds to, provided for message type Access Connection.
  - String interfaceType – The node interface type the report entry corresponds to, provided for message type Access Connection.
  - String accessMode – The access mode used.
  - String reasonForAccess – The reason for access provided by the user.

ReportData

This data describes a report generated using runReport(). The information can be used to manage the report and retrieve further results.

- Elements

- String reportID – Identifier used to reference the report.
- Integer totalNumberOfRecords – The total number of records available for this report. This value is not necessarily the same as the number of ReportRecord entries included.
- ReportRecord[] records – The first set of results for the report.

LoggingManagementException

All errors are returned via exception as defined under Common Data Types (on page 10).

Services

runReport

Returns a log report formed using the request parameters.

The client should call deleteReport() for every call to runReport(). See deleteReport (on page 36). If deleteReport() is not called, the session manager will remove remaining report tasks when the session is closed by any of the following methods.

1. The client calls signOff() to close the session.
2. The session times out and the system closes it.
3. An administrator closes the session from the Active Users report using the CC-SG Admin Client.

- parameters
  - String sessionID
  - xsd:dateTime startDateTime – Return records after this date and time. If null, the value will be the start of the current day.
  - xsd:dateTime endDateTime – Return records up until this date and time. If null or beyond the CC-SG's current time, the value will be the current time.
  - String userName – Restrict results to those of this user name. A standard wild card search field. See Standard Wild Card Search Field (on page 11). The empty string will be ignored like null.
  - String userIPAddress – Restrict results to those of this IP address. A standard wild card search field. See Standard Wild Card Search Field (on page 11). The empty string will be ignored like null.
  - String messageType – Search for messages in the specified category. If null or empty, include all message types. The following are supported:
  - Access Audit
  - Access Connection
  - Authentication
  - Error
  - Power
  - Tasks
  - User Maintenance
  - String message – Restrict results to those containing this message. A standard wild card search field. See Standard Wild Card Search Field (on page 11). The empty string will be ignored like null.

    - Integer numberOfRecordsToGet – The maximum number of records to retrieve. If not specified, the service retrieves all available records, as reported by totalNumberOfRecords. When specified, can be used to limit the number of records retrieved. There is a maximum limit of 10,000 records.
    - return value
    - ReportData

getReportRecords

Retrieve a set of records from a previously generated report starting at the specified record index up to the maximum number of records indicated.

- parameters
    - String sessionID
    - String reportID
    - Integer startingAtRecord – The index of the first record to retrieve. Indexing begins at zero.
- Integer numberOfRecords – The maximum number of records to retrieve. If not specified, the service retrieves all available records, as reported by totalNumberOfRecords. When specified, can be used to limit the number of records retrieved. There is a maximum limit of 10,000 records.
- return value
    - ReportRecord[]

getReportDocument

Retrieve accounting reports on resources like device ports. The required permission is specific to each report.

- parameters
    - String sessionID
    - String type – Specify the category of accounting. Case insensitive.
    - ▪ Device
    - String report – The accounting asset. Case insensitive. See Reports and Options below for report descriptions.
    - ▪ Query
    - KeyValue[] parameters – Some reports accept parameters to determine the accounting results. Keys are case insensitive. See Reports and Options below for the options each report accepts.
- Return value
    - String – The requested report in CSV format.

Reports and Options:

Together, type and report specify the report to generate.

Type Value	ReportValue	Description
Device	Query	Retrieve accounting of device ports (See Query Port Report in Admin Guide).

Type Value	ReportValue	Description
		Permission: Device, Port and Node Management Accepts the following parameters: Paused – True includes paused ports; false or missing excludes paused ports. (optional)

deleteReport

Delete a previously requested report; otherwise, reports are deleted after the user session is terminated.

- parameters

String sessionID

- String reportID
- return value
- boolean true

Category Management Services

This set of services is for getting and setting categories for CC-SG nodes.

```
http(s)://CC_IP_ADDRESS/CommandCenterWebServices/
CategoryManagementServicePort?wsdl
```

Data Types

CategoryData

Stores information about a category and its elements.

- Elements
  - String name
  - boolean node - Applicable to nodes
  - boolean device - Applicable to devices
  - String type - Data type of elements: "String" or "Integer"
  - String[]elements - The values of the elements

CategoryManagementException

All errors are returned via exception as defined under Common Data Types (on page 10).

Services

getCategory

Retrieve one or more categories including element values.

- parameters
  - String sessionID
  - String name - The name of the category. A standard wild card search field. See Standard Wild Card Search Field (on page 11).
- return value
  - CategoryData[]

addCategory

Add a new category to CC-SG.

- parameters
  - String sessionID
  - String name - Unique name to identify the category
  - boolean node - Applicable to nodes
  - boolean device - Applicable to devices
  - String type - Data type of elements: "String" or "Integer"
- return value
  - boolean true on success

editCategory

Facilitates changes to the editable components of a category.

- parameters

    - String sessionID
    - String currentName - the name of the existing category
    - String newName - the new name for the category (same as currentName if no change)
    - boolean forNode
    - boolean forDevice
- return value
    - boolean true on success

deleteCategory

Remove one category.

- parameters
    - String sessionID
    - String name
- return value
    - boolean true on success

addElementToCategory

Add one or more values to the specified category

- parameters
    - String sessionID
    - String category - The unique name identifying the category
    - String[]elements - Each entry in this array is a value for the category.
- return value
    - boolean true on success

renameElementInCategory

Change the value of a category's element.

- parameters
    - String sessionID
    - String category - The unique name identifying the category
    - String currentElement - The current value of the element
    - String newElement - The new value intended for the element
- return value
    - boolean true on success

deleteElementFromCategory

Remove one or more values from the specified category.

- parameters

    - String sessionID
    - String category - The unique name identifying the category
    - String[]elements - Each array entry is a value to be removed from the category
- return value
    - returns boolean true on success

Appendix A Certificate Management

CC-SG web services require HTTPS which may require your client to trust the CC-SG certificate explicitly or via the local trust store. The CC-SG requires WS-API clients to present the client certificate created during configuration. This appendix contains some tips on managing these certificates. See the respective software's documentation for more details.

In This Chapter

Java keytool

► *Viewing a certificate:*

```
keytool -printcert -file CC-SG.pem
```

► *Add a certificate to a new key store:*

```
keytool -importcert -alias [ADDRESS] -file CC-SG.cert -keystore jssecacerts -storepass yourpassword
```

► *View the contents of a key store:*

```
keytool -list -keystore jssecacerts
```

OpenSSL

► *Viewing a certificate:*

```
openssl x509 -in CC-SG.pem -text
```

► *Changing a certificate's format:*

```
openssl x509 -in CC-SG.cert -out CC-SG.pem -outform PEM -inform DER
```

Saving the CCSG's Server Certificate

► *To save the certificate in the browser:*

Browsers typically offer an option to save the server certificate for users to verify the server's identity.

1. Open a URL to your CC-SG using HTTPS.

https://CCSGADDRESS

2. Click on the padlock, which is usually next to the URL, to access the server certificate.
3. Save the certificate. Please consult documentation for your specific browser.
   - Firefox has a download section. Get the PEM (cert).
   - In Edge, select Copy to File under the Details tab.

► *To save the certificate from the Admin Client:*

1. Choose Administration > Security > Certificate.
2. Select "Export current certificate and private key".
3. Click Export to file.
4. Copy the PEM formatted certificate text field to a file.

Installing the Client Certificate into a Key Store (Microsoft Windows)

1. Save the PKCS12 certificate file on your client computer. See Add Web Services API Client Configuration on CC-SG (on page 7).
2. Double click the client certificate file to open the Certificate Import Wizard.
3. Install the certificate in a place where your client will be able to access it, such as Personal.

You can also use Microsoft Management Console (MMC) to manage certificates with more control than the methods above. For example, add the client certificate into a specific store such as Current User or Local Computer.

Appendix A Web Services Development in Java

This section focuses on CC-SG specific topics regarding WS client development in Java.

In This Chapter

Choose a WS Library

The Java API for XML Web Services (JAX-WS) is the common standard for deploying and consuming Web Services in Java. Glassfish contains a JAX-WS Reference Implementaion under Metro which can be found at https://javaee.github.io/metro-jax-ws/. This document is based on version 2.1.5, but you may use a different version or an entirely different WS system to suit your needs.

The Glassfish implementation includes a tool called wsimport that one can use to generate WS client code for one's application using WSDL files from the WS server. Such tools are invaluable in WS development. Please consult the documentation of your chosen JAX-WS system for details on wsimport, other tools, and non-CCSG related information.

Certificates information for Java users

Create a truststore for the trusted server certificate, or add the certificate to a truststore you already use.

► *To tell a Java client about the trusted server certificate:*

You can do this via -Djavax.net.ssl.trustStore=jssecacerts.

Save the client certificate from the CC-SG's WS client configuration window. The file is in PKCS12 format. You can pass the client certificate to a Java client using the following parameters:

`-Djavax.net.ssl.keyStore=new_client.p12`

`-Djavax.net.ssl.keyStorePassword=pass` (Where `pass` is the password you entered in the client configuration page.)

`-Djavax.net.ssl.keyStoreType=pkcs12`

Setting the CCSG Address

Downloading the WSDL files from port 8080 of the CC-SG is the default source of the WSDL files, however, their contents will reflect port 8080 of your CC-SG. Further, you might wish to use your WS client with a different CC-SG or you might change the CC-SG's address.

Each WSDL file contains an element like the following:

```
<soap:address location="https://10.0.0.101:9443/
CommandCenterWebServices/AuthenticationAndAuthorizationServicePort"/>
```

Before generating the stubs, you can edit this address within each file to reflect the WS port of 9443, the WS protocol of HTTPS, and the address of your CC-SG. After generating and building the source, your client will already know the address, protocol, and port to use to access WS on your CC-SG.

Manually editing the WSDL files requires less coding. Or, you can tell your WS stubs the URL of your own choice at run time.

► *Sample method to create a URL to access a CC'-SGs web services:*

```
public static void set_service_end_point( Service service,
BindingProvider port )

{

Pattern pattern = Pattern.compile( "CC_SG_" );

Matcher matcher =
pattern.matcher( service.getServiceName().getLocalPart() );

String service_name = matcher.replaceFirst( "" );

String ccsg_port = "9443";

String ccsg_address = "10.0.0.101";

port.getRequestContext().put(

BindingProvider.ENDPOINT_ADDRESS_PROPERTY,

"https://" + ccsg_address + ":" + ccsg_port +

"/CommandCenterWebServices/" +

service_name + "Port?wsdl" );

}
```

Call the method from your application for each service object. This example uses AuthenticationAndAuthorizationService:

```
CCSGAuthenticationAndAuthorizationService service = new
CCSGAuthenticationAndAuthorizationService();

AuthenticationAndAuthorizationService service_port =
service.getAuthenticationAndAuthorizationServicePort();
```

```
set_service_end_point( service, (BindingProvider)service_port );
```

This procedure must be done for each service, like NodeManagementService, to connect to the intended CC-SG.

Calling a Web Service

JAX-WS requires an instance of the service class for the desired service generated by wsimport. The service object contains a port object that one can use to call the service methods as shown below:

```
CCSGAuthenticationAndAuthorizationService service = new
CCSGAuthenticationAndAuthorizationService();

AuthenticationAndAuthorizationService service_port =
service.getAuthenticationAndAuthorizationServicePort();

try

{

String sessionID = port.signOn( user, password );

} catch ( security.service.webservice.bl.cc.raritan.com AuthenticationAndAuthorizationException ex )

{

System.out.println( "AuthenticationAndAuthorizationException: " + ex.getFaultInfo().getMessage() );

System.out.println( "\t" + ex.getFaultInfo().getCode() );

System.out.println( "signOn() for user " + user + " failed." );

}
```

Sample Application for Java

```
/*

* RCSfile: ...

* Revision: ...

* Date: ...

*

* This source code is owned by Raritan, Inc. and is confidential

* and proprietary information distributed solely pursuant to a

* confidentiality agreement or other confidentiality obligation.
* It is intended for informational purposes only and is distributed

* "as is" with no support and no warranty of any kind.

*

* Copyright (c) 2009 Raritan, Inc. All rights reserved.

* Reproduction of any element without the prior written consent of

* Raritan, Inc. is expressly forbidden.

*/

import
security.service.webservice.bl.cc.raritan.com.AuthenticationAndAuthorizationService;

import
security.service.webservice.bl.cc.raritan.com.CCSGAuthenticationAndAuthorizationServic

import security.service.webservice.bl.cc.raritan.com.types.*;

import
node.service.webservice.bl.cc.raritan.com.NodeManagementService;

import
node.service.webservice.bl.cc.raritan.com.CCSGNodeManagementService;

import node.service.webservice.bl.cc.raritan.com.types.*;

// change server address

import javax.xml.ws.Service;

import javax.xml.ws.BindingProvider;

import java.util.regex.Pattern;

import java.util.regex.Matcher;

// user input

import java.io.*;

public class SampleClient

{

public static String ccsg_address = "10.0.0.101", ccsg_port = "9443";
```

```
static void
auth_exception_handler( security.service.webservice.bl.cc.raritan.com.AuthenticationAn
ex,

String name )

{

System.out.println( "AuthenticationAndAuthorizationException: " +
ex.getFaultInfo().getMessage() );

System.out.println( "\t" + ex.getFaultInfo().getCode() );

// ex.printStackTrace();

}

public static void set_service_end_point( Service service,
BindingProvider port )

{

Pattern pattern = Pattern.compile( "CC_SG_" );

Matcher matcher =
pattern.matcher( service.getServiceName().getLocalPart() );

String service_name = matcher.replaceFirst( "" );

if( ccsg_port.length() < 1 )

ccsg_port = "9443";

if( ccsg_address.length() > 0 )

{

port.getRequestContext().put(

BindingProvider.ENDPOINT_ADDRESS_PROPERTY,

"https://" + ccsg_address + ":" + ccsg_port +

"/CommandCenterWebServices/" +

service_name + "Port?wsdl" );

}

}
```

```
static String get_input( String message )

{

System.out.println(message);

BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

String name = null;

try {

name = reader.readLine();

} catch (IOException ioe)

{

System.err.println("Could not read input.");

return null;

}

if( name.equals("") )

return null;

return name;

}

public static void main (String[] args)

{

String user = "gregor";

String password = "pass123";

String session = "";

String current_name, new_name;

CCSGAuthenticationAndAuthorizationService service = new
CCSGAuthenticationAndAuthorizationService();

AuthenticationAndAuthorizationService port =

service.getAuthenticationAndAuthorizationServicePort();
```

```
set_service_end_point( service, (BindingProvider)port );

CCSGNodeManagementService node_service = new
CCSGNodeManagementService();

NodeManagementService node_service_port =
node_service.getNodeManagementServicePort();

set_service_end_point( node_service,
(BindingProvider)node_service_port );

try

{

session = port.signOn( user, password );

} catch
( security.service.webservice.bl.cc.raritan.com.AuthenticationAndAuthorizationExceptio
ex )

{

auth_exception_handler( ex, "signOn()" );

System.exit(1);

}

current_name = get_input( "Enter the name of the node to change: " );

new_name = get_input( "Enter the new name: " );

if( current_name != null && new_name != null )

{

try

{

if( node_service_port.renameNode( session, current_name, new_name ) )

System.out.println( "Node name successfully changed to " + new_name );

else

System.err.println( "renameNode() failed without an exception." );
}
catch( node.service.webservice.bl.cc.raritan.com.NodeManagementException
ex )

{

System.out.println( "NodeManagementException: " +
ex.getFaultInfo().getMessage() );

System.out.println( "\t" + ex.getFaultInfo().getCode() );

// ex.printStackTrace();

}

}

else

System.err.println( "Could not change node name without the current
and new names." );

try

{

port.signOff( user, session );

} catch
( security.service.webservice.bl.cc.raritan.com.AuthenticationAndAuthorizationExceptio
ex )

{

auth_exception_handler( ex, "signOff()" );

}

}

}
```

Appendix A Web Services Development in Visual Studio

The following sections describe how to create a Web Services client for the CCSG written in C# using Windows Communication Foundation (WCF). This description is based on Microsoft Visual Studio 2022 with .NET desktop development (.NET Framework 4.7.2) and Windows Communication Foundation installed.

In This Chapter

Creating a CC-SG Web Service in a Project

1. Create a Console App (.NET Framework).
2. Select Project > Add Service Reference
3. Set the address to the URL of the CC-SG service WSDL.
   - For example: http://CCSGADDRESS/CommandCenterWebServices/AuthenticationAndAuthorizationServicePort?wsdl

   Note: If you change the namespace from the default (ServiceReference1), reference it later as the endpoint contract.

4. Right click app.config in Solution Explorer and select Edit WCF Configuration.
5. Under Bindings, select the binding matching the service.
   - For example: CC_SG_AuthenticationAndAuthorizationServiceSoapBinding
6. Copy the name and delete the binding.
7. Right click Bindings and select New Binding Configuration.
8. Select customBinding,
9. Set Name to that of the deleted binding.
   - For example: CC_SG_AuthenticationAndAuthorizationServiceSoapBinding
10. Remove httpTransport.
11. Add httpsTransport.
12. Open httpsTransport.
13. Set RequireClientCertificate to True.
14. Open textMessageEncoding.
15. Set MessageVersion to Soap11.
16. Select the matching service under Client->Endpoints.
17. Change the Binding to customBinding.
18. Verify that BindingConfiguration matches the new customBinding's name.

19. Edit the Address to use HTTPS, your CCSG's address, and port 9443 (rather than 8080).

Note: You must edit the URL and set the binding for each service that you use. The same binding can be used for every CCSG service.

---

20. Save the configuration.
21. Edit the C# source.
    - Instantiate the client.
    - Tell the client which client certificate to use for identity.
22. Access the CC-SG API (signOn in this example).

Sample Application for C#

```
using System;
using System.Security.Cryptography.X509Certificates;
// For testing with self signed server certificate:
//using System.Net;
//using System.Net.Security;
namespace CCWSClient
{
internal class Program
{
static void Main(string[] args)
{
try
{
// Uncomment for testing: accept default self signed certificate.
//ServicePointManager.ServerCertificateValidationCallback =
// delegate (object obj, X509Certificate certificate, X509Chain chain, SslPolicyErrors
errors)
// {
// X509Certificate server_certificate =
// X509Certificate.CreateFromCertFile(@"C:\cc_certificate.cer”);
// return server_certificate.Equals(certificate);
// };
CCAuth.AuthenticationAndAuthorizationServiceClient auth_service =
new CCAuth.AuthenticationAndAuthorizationServiceClient();
// Prove identity to CC-SG using client certificate.
auth_service.ClientCredentials.ClientCertificate.SetCertificate(
StoreLocation.CurrentUser, StoreName.My, X509FindType.FindBySubjectName,
"FQDN"); // Same as entered to create client certificate in Admin client.
Console.WriteLine("Connecting to CC-SG...");
auth_service.Open();
// Authenticate and show the session ID
Console.WriteLine(auth_service.signOn("user", "password"));
}
catch (Exception e)
{
Console.WriteLine(e.ToString());
}
}
}
}
```

Appendix A Web Services Development in a Shell

This appendix describes web services development in a shell. Web Services are built on HTTP which can be accessed from a low level. All of the API operations and data are contained within XML. The XML is normally and most easily handled by a WS development tool kit, but it is possible to process manually or by your own preferred methods. You can find information on using WSDL to construct SOAP messages online. This guide is an example to get you started."

The signOn() request is formatted as:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://
schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<S:Body>
<ns2:signOn xmlns:ns2="http://com.raritan.cc.bl.webservice.service.security/types">
<String_1>testuser</String_1> <!-- Enter user name. -->
<String_2>password</String_2> <!-- Enter password. -->
</ns2:signOn>
</S:Body>
</S:Envelope>
```

In This Chapter

Using wget and curl

HTTPS capable software, such as wget and curl, can handle transport of the XML data to and from the CC-SG. All clients must present the client certificate to the CC-SG. See Add Web Services API Client Configuration on CC-SG (on page 7).

If needed, OpenSSL can reformat the PKCS12 certificate and key to a format the HTTPS software can use. For curl and wget, the P12 file can be converted to separate PEM formatted certificate and key files as follows:

```
openssl pkcs12 -in client.p12 -nokeys -out client.pem -nodes
openssl pkcs12 -in client.p12 -nocerts -out key.pem -nodes
```

Given the XML, client certificate, and client key above, wget can be used with the following arguments:

- Post file as text/xml to the CC-SG.

```
--post-file=signOn.xml --header="Content-Type: text/xml"
```

- Reduced security is useful for testing (i.e. the CC-SG is using a self signed certificate).

```
--no-check-certificate
```

- Present client certificate to the CC-SG.

```
--certificate=client.pem --certificate-type=PEM
```

- Use matching private key.

```
--private-key=key.pem --private-key-type=PEM
```

- Put the result into a file.

```
-O response.xml
```

- URL to reach the service.

```
https://CCSGADDRESS:9443/CommandCenterWebServices/
AuthenticationAndAuthorizationServicePort
```

curl can do the same operation with the following arguments:

- Post text/xml to the CC-SG.

```
-X POST -H "Content-Type: text/xml"
```

- Reduced security is useful for testing (i.e. the CC-SG is using a self signed certificate).

```
-k
```

- Send contents of file.

```
-d @signOn.xml
```

- Present client certificate to the CC-SG.

```
--cert ./client.pem --cert-type PEM
```

- Use matching private key.

```
--key ./key.pem --key-type PEM
```

- URL to reach the service.

```
https://CCSGADDRESS:9443/CommandCenterWebServices/
AuthenticationAndAuthorizationServicePort
```

► *Successful response:*

A successful response contains the expected session ID from signOn().

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ns2:signOnResponse xmlns:ns2="http://com.raritan.cc.bl.webservice.service.security/
types">
<result>SESSIONID</result>
</ns2:signOnResponse>
</soap:Body>
</soap:Envelope>
```

► *Error response:*

A typical error response will include the message and code of the exceptions defined in this document.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<soap:Body>
<soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>Fault occurred while processing.</faultstring>
<detail>
<ns1:AuthenticationAndAuthorizationException xmlns:ns1="http://
com.raritan.cc.bl.webservice.service.security/types">
<message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns2="http://
com.raritan.cc.bl.webservice.service.security/types" xsi:nil="true"/>
<code xmlns:ns2="http://com.raritan.cc.bl.webservice.service.security/
types">INVALID_CREDENTIALS</code>
</ns1:AuthenticationAndAuthorizationException>
</detail>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

► *Bash script using curl:*

```
#!/bin/bash
CCSG=10.0.0.103 # Address of CC-SG
USER=testuser
PASSWORD=password
TARGETID=269 # The ID of the target interface available from Bookmark Node Interface
# and InterfaceData returned from API operations like getNodesForUser().
CERT=./client.pem
KEY=./key.pem
#DEBUG=true
AUTHSERV=AuthenticationAndAuthorizationServicePort
NODESERV=NodeManagementServicePort
CONTENT="Content-Type: text/xml"
EXITCODE=100
set -E
trap "[ \"\$?\" -ne $EXITCODE ] || echo Script failed. && exit 1" ERR
if [ "$DEBUG" = true ]; then
set -x
else
exec 2>/dev/null
fi
function get_needle()
{
NEEDLE="$1"
HAYSTACK="$2"
STRIP="$3"
if grep -q "${NEEDLE}" <<< "${HAYSTACK}"; then
SED="s/.*\(<${NEEDLE}>.*<\/${NEEDLE}>\).*/\1/"
if [ "$STRIP" = true ]; then
SED="s/.*<${NEEDLE}>\(.*\)<\/${NEEDLE}>.*/\1/"
fi
sed -e "$SED" <<< "$HAYSTACK"
else
echo "Could not find $NEEDLE in result. Quitting."
exit $EXITCODE
fi
}
function curl_rpc()
{
SERVICE="$1"
XML="<S:Envelope xmlns:S=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:SOAP-
ENV=\"http://schemas.xmlsoap.org/soap/envelope/\">
<SOAP-ENV:Header/>
<S:Body>
$2
</S:Body>
</S:Envelope>"
OUT=$(curl -X POST -H "$CONTENT" -k -d @- --cert $CERT --cert-type PEM --key $KEY --
key-type PEM https://$CCSG:9443/CommandCenterWebServices/$SERVICE ${DEBUG:+-v} <<<
"$XML")
get_needle result "$OUT"
}
function signOnOff()
{
OP="$1"
USER="$2"
PASS_SESS="$3"
XML="<ns2:sign${OP} xmlns:ns2=\"http://com.raritan.cc.bl.webservice.service.security/
types\">
<String_1>${USER}</String_1>
<String_2>${PASS_SESS}</String_2>
</ns2:sign${OP}>"
OUT=$(curl_rpc $AUTHSERV "$XML")
get_needle result "$OUT" true
}
function signOn()
{
signOnOff On "$1" "$2"
}
function signOff()
{
signOnOff Off "$1" "$2"
}
function getInterfaceURL()
{
SESSIONID="$1"
INTERFACEID="$2"
XML="<ns2:getInterfaceURL xmlns:ns2=\"http://com.raritan.cc.bl.webservice.service.node/
types\">
<String_1>${SESSIONID}</String_1>
<String_2>${INTERFACEID}</String_2>
<String_3 xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xsi:nil=\"true\"/>
</ns2:getInterfaceURL>"
OUT=$(curl_rpc $NODESERV "$XML")
OUT=$(get_needle result "$OUT" true)
PROTOCOL=$(get_needle protocol "$OUT" true)
PORT=$(get_needle port "$OUT" true)
URLPATH=$(get_needle path "$OUT" true)
TOKENKEY=$(get_needle tokenKey "$OUT" true)
TOKENVALUE=$(get_needle tokenValue "$OUT" true)
echo "${PROTOCOL}://${CCSG}:${PORT}${URLPATH}&${TOKENKEY}=${TOKENVALUE}"
}
SESSIONID="$(signOn $USER $PASSWORD)"
echo Session ID: $SESSIONID
URL=$(getInterfaceURL $SESSIONID $TARGETID)
if [ -n "$URL" ]; then
printf "Connect to target with interface %s using URL:\n%s\n" $TARGETID $URL
else
printf "Failed to build URL.\nSigning off: %s" $(signOff $USER $SESSIONID)
```

Appendix A Web Services Development in C

This example uses Axis2C for handling the WS communications and was tested with 1.6.0 on Ubuntu 10.04 LTS - the Lucid Lynx.

In This Chapter

Sample Application for C

This example uses Axis2C for handling the WS communications (tested with 1.6.0 on Ubuntu 10.04 LTS - the Lucid Lynx.

```
/*
* This source code is owned by Raritan, Inc. and is confidential
* and proprietary information distributed solely pursuant to a
* confidentiality agreement or other confidentiality obligation.
* It is intended for informational purposes only and is distributed
* "as is" with no support and no warranty of any kind.
*
* Copyright (c) 2012 Raritan Computer, Inc. All rights reserved.
* Reproduction of any element without the prior written consent of
* Raritan, Inc. is expressly forbidden.
*/
#include <stdio.h>
#include <axiom.h>
#include <axis2_util.h>
#include <axiom_soap.h>
#include <axis2_client.h>
#include <axis2_svc_client.h>
#define TRUE 1
#define FALSE 0
#define CC_ADDRESS "10.0.0.102"
#define CCWS_URL "https://" CC_ADDRESS ":9443/CommandCenterWebServices/"
#define CCWS_URL_AUTH CCWS_URL "AuthenticationAndAuthorizationServicePort"
#define CCWS_NS_AUTH "http://com.raritan.cc.bl.webservice.service.security/types"
#define CC_WS_SIGNON "signOn"
#define CC_WS_SIGNOFF "signOff"
#define CC_WS_RESULT "result"
#define CCWS_URL_NODE CCWS_URL "NodeManagementServicePort"
#define CCWS_NS_NODE "http://com.raritan.cc.bl.webservice.service.node/types"
#define GET_CC_APPLET_URL "getCCSGAppletURL"
char g_debug_enable = FALSE;
char*
my_strcpy_alloc(
const char *source
)
{
int len = strlen( source ) + 1;
char *dest = malloc( len );
snprintf( dest, len, "%s", source );
return dest;
}
void
set_service_destination(
const axutil_env_t * env,
axis2_svc_client_t *client,
const char *destination
)
{
axis2_options_t *options = axis2_svc_client_get_options( client, env );
axis2_endpoint_ref_t *endpoint_ref = axis2_endpoint_ref_create( env, destination );
axis2_options_set_to( options, env, endpoint_ref );
}
axiom_node_t *
send_message(
const axutil_env_t * env,
axis2_svc_client_t *client,
axiom_node_t *message
)
{
axiom_node_t *response;
axis2_char_t *temp = NULL;
if( g_debug_enable &&
(temp = axiom_node_to_string(message, env)) )
{
printf("\nSending message:\n%s\n", temp);
AXIS2_FREE(env->allocator, temp);
}
response = axis2_svc_client_send_receive( client, env, message );
if( response )
{
if( g_debug_enable )
{
temp = axiom_node_to_string(response, env);
if (temp)
printf("\nResponse message:\n%s\n", temp);
AXIS2_FREE(env->allocator, temp);
}
}
else
{
AXIS2_LOG_ERROR(env->log, AXIS2_LOG_SI,
"Stub invoke FAILED: Error code:" " %d :: %s",
env->error->error_number,
AXIS2_ERROR_GET_MESSAGE(env->error));
printf("No response!\n");
}
return response;
}
axiom_node_t *
signOnOff(
const axutil_env_t * env,
axis2_svc_client_t *client,
char is_signOn,
const char *user,
const char *second_parameter
)
{
axiom_node_t *node = NULL;
axiom_element_t *echo_om_ele = NULL;
axiom_node_t *text_om_node = NULL;
axiom_element_t *text_om_ele = NULL;
axiom_namespace_t *ns = NULL;
set_service_destination( env, client, CCWS_URL_AUTH );
ns = axiom_namespace_create(env, CCWS_NS_AUTH, "ns2");
echo_om_ele = axiom_element_create(env, NULL, (is_signOn ? CC_WS_SIGNON :
CC_WS_SIGNOFF ), ns, &node);
text_om_ele = axiom_element_create(env, node, "String_1", NULL, &text_om_node);
axiom_element_set_text(text_om_ele, env, user, text_om_node);
```

```
text_om_ele = axiom_element_create(env, node, "String_2", NULL, &text_om_node);
axiom_element_set_text(text_om_ele, env, second_parameter, text_om_node);
return send_message( env, client, node );
}
char* // session
signOn(
const axutil_env_t * env,
axis2_svc_client_t *client,
const char *user,
const char *password
)
{
printf( CC_WS_SIGNON "()\n" );
axiom_element_t *response_element;
axiom_node_t *session_node;
axiom_element_t *session_element;
axiom_node_t *response_node = signOnOff( env, client, TRUE, user, password );
axis2_char_t *session;
if( !response_node )
{
printf( "signOn FAILED!\n" );
return NULL;
}
response_element = (axiom_element_t *) axiom_node_get_data_element( response_node,
env );
if( axutil_strcmp( axiom_element_get_localname( response_element, env ), CC_WS_SIGNON
"Response" ) != 0 )
{
printf( "signOn FAILED: Invalid response.\n" );
return NULL;
}
session_node = axiom_node_get_first_element( response_node, env );
if( !session_node )
{
printf( "signOn FAILED: No session node.\n" );
return NULL;
}
session_element = (axiom_element_t *) axiom_node_get_data_element( session_node, env );
if( axutil_strcmp( axiom_element_get_localname( session_element, env ),
CC_WS_RESULT ) != 0 )
{
printf( "signOn FAILED: No result in response.\n" );
return NULL;
}
session = axiom_element_get_text( session_element, env, session_node );
printf( "\tSession ID: %s\n", session );
return my_strcpy_alloc( session );
}
void
signOff(
const axutil_env_t * env,
axis2_svc_client_t *client,
const char *user,
const char *session
)
{
printf( CC_WS_SIGNOFF "()\n" );
signOnOff( env, client, FALSE, user, session );
}
void
getCCSGAppletURL(
const axutil_env_t * env,
axis2_svc_client_t *client,
const char *session
```

```
)
{
axiom_node_t *node = NULL;
axiom_element_t *echo_om_ele = NULL;
axiom_node_t *text_om_node = NULL;
axiom_element_t *text_om_ele = NULL;
axiom_namespace_t *ns = NULL;
axiom_element_t *response_element;
axiom_node_t *child_node;
axiom_element_t *temp_element;
axiom_node_t *response_node;
axis2_char_t *temp;
char *path = NULL;
char *port = NULL;
char *protocol = NULL;
char *tokenKey = NULL;
char *tokenValue = NULL;
printf( GET_CC_APPLET_URL "()\n" );
// create request
set_service_destination( env, client, CCWS_URL_NODE );
ns = axiom_namespace_create(env, CCWS_NS_NODE, "ns2");
echo_om_ele = axiom_element_create(env, NULL, GET_CC_APPLET_URL, ns, &node);
text_om_ele = axiom_element_create(env, node, "String_1", NULL, &text_om_node);
axiom_element_set_text(text_om_ele, env, session, text_om_node);
response_node = send_message( env, client, node );
// extract URL from response
if( !response_node )
{
printf( "FAILED!\n" );
return;
}
response_element = (axiom_element_t *) axiom_node_get_data_element( response_node,
env );
if( axutil_strcmp( axiom_element_get_localname( response_element, env ),
GET_CC_APPLET_URL "Response" ) != 0 )
{
printf( "FAILED: Invalid response.\n" );
return;
}
// result node
child_node = axiom_node_get_first_child(response_node, env);
if( !child_node )
{
printf( "FAILED: No result node.\n" );
return;
}
temp_element = (axiom_element_t *) axiom_node_get_data_element( child_node, env );
if( axutil_strcmp( axiom_element_get_localname( temp_element, env ), CC_WS_RESULT ) !=
0 )
{
printf( "FAILED: No result in response.\n" );
return;
}
// URL components within result
child_node = axiom_node_get_first_child(child_node, env);
while (axiom_node_is_complete(response_node, env) && child_node)
{
temp_element = (axiom_element_t *) axiom_node_get_data_element( child_node, env );
char *name = axiom_element_get_localname( temp_element, env );
char *text;
text = axiom_element_get_text( temp_element, env, child_node );
if( axutil_strcmp( name, "path" ) == 0 )
path = text;
else if( axutil_strcmp( name, "port" ) == 0 )
port = text;
else if( axutil_strcmp( name, "protocol" ) == 0 )
protocol = text;
else if( axutil_strcmp( name, "tokenKey" ) == 0 )
tokenKey = text;
else if( axutil_strcmp( name, "tokenValue" ) == 0 )
tokenValue = text;
else
printf( "\tMissed data: %s (%s)\n", name, text );
child_node = axiom_node_get_next_sibling(child_node, env);
}
printf( "\t%s://%s:%s%s?%s=%s\n", protocol, CC_ADDRESS, port, path, tokenKey,
tokenValue );
}
int
main(
int argc,
char **argv)
{
const axutil_env_t *env = NULL;
axis2_options_t *options = NULL;
axis2_svc_client_t *client = NULL;
char user[] = "gregor";
char password[] = "pass123";
char *session;
env = axutil_env_create_all("ccwstest.log", AXIS2_LOG_LEVEL_TRACE);
options = axis2_options_create(env);
client = axis2_svc_client_create(env, "/usr/lib/axis2/");
if (!client)
{
printf( "client creation failed.\n" );
AXIS2_LOG_ERROR(env->log, AXIS2_LOG_SI,
"Client creation failed: Error code:" " %d :: %s",
env->error->error_number,
AXIS2_ERROR_GET_MESSAGE(env->error));
return -1;
}
axis2_svc_client_set_options(client, env, options);
// CC uses SOAP 1.1
axis2_options_set_soap_version( options, env, AXIOM_SOAP11 );
axis2_svc_client_engage_module( client, env, AXIS2_MODULE_ADDRESSING );
// CC services
session = signOn( env, client, user, password );
getCCSGAppletURL( env, client, session );
// Note: Do not signOff() until the applet session is complete.
if( session )
signOff( env, client, user, session );
if (client)
{
axis2_svc_client_free(client, env);
client = NULL;
}
if (env)
{
axutil_env_free((axutil_env_t *) env);
env = NULL;
}
return 0;
}
```

Index