CommandCenter Secure Gateway

WS-API Programming Guide

Release 4.3

CC-WSAPI

December 2009

255-80-0004-00

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、情報処理装置等電波障害自主規制協議会（ＶＣＣＩ）の基準に基づくクラス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。

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.

Contents

Chapter 1 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 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

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 2)
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 2) for instructions on creating the client certificate. See ***Certificates*** (on page 4).

   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 25)

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 4)
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. You can add up to five WS-API clients. 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: 1024 is the default.

c. Validity Period (days): Maximum 4 numeric characters.

d. Country Code: CSR tag is Country Name.

e. State or Province: Maximum 64 characters. Type in the whole state or province name. Do not abbreviate.

f. City/Locality: CSR tag is Locality Name. Maximum 64 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.

k. Challenge Password: Maximum 64 characters.

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

---

l. 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.

Access Information

WSDL URLs

```
http://CC_IP_ADDRESS:8080/CommandCenterWebServices/AuthenticationAndAuthorizationServicePort?wsdl

http://CC_IP_ADDRESS:8080/CommandCenterWebServices/NodeManagementServicePort?wsdl

http(s)://CC_IP_ADDRESS:8080/CommandCenterWebServices/UserManagementServicePort?wsdl

https://CC_IP_ADDRESS:8080/CommandCenterWebServices/LoggingManagementServicePort?wsdl
```

Certificates

The CC-SG's Web Services require mutual certificates such that both the CC-SG and the WS client present a certificate. Once one configures a client on the CC-SG, the CC-SG will know about both certificates. The client also needs to know about both certificates.

The CC-SG's server certificate on port 9443 is generated by the CC-SG itself. 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.

Chapter 2 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.

Authentication and Authorization Services

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

```
http://CC_IP_ADDRESS:8080/CommandCenterWebServices/AuthenticationAndAuthorizationServicePort?wsdl
```

Data Types

AuthenticationAndAuthorizationException

Exception returned for all errors specific to authentication and authorization.

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()

This operations signs off (log out) a particular user from CC-SG. 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**

Unsupported Authentication and Authorization Services

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

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

Node Management Services

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

`http://CC_IP_ADDRESS:8080/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.

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.

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.

NodeManagementException

Exception returned for all errors specific to node management.

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.

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

NodeManagementException

Exception returned for all errors specific to node management.

Services

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 [ ]**

getInterfaceURL

This operation retrieves the URL required to connect to an interface, using access method (application), provided by CC-SG. This URL can be used as the path for a web browser or any HTML aware application to initiate the connection via the chosen application and destination. 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 accessApplicationId - CC-SG generated string which uniquely identifies the application type within CC-SG
- return value:
  - **URLObject**

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

getNodesForUser

This operation retrieves all available nodes for a particular user, by username. For each node that a user has access to, its interfaces are checked and all relevant data recorded for both in-band and out-of-band interfaces. All interfaces for one node are then inserted into one object - **NodeData**, which besides the interfaces data, also contains the name of the particular node, for identification. An array list of all nodes that the user has access to and their interfaces is returned on success.

- parameters
  - **String** *username*

- return value **NodeData [ ]**

  Custom class **NodeData** has the following properties

  - **String** *name*, i.e. the name of the node
  - **InterfaceData [ ]** *interfacesData*, i.e. data about interfaces contained in the node

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

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

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

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:8080/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

Exception returned for all errors specific to user management.

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 user defined on the CC-SG.

- 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

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

```
http(s)://CC_IP_ADDRESS:8080/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

Exception returned for all errors specific to logging management.

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

Services

runReport

Returns a log report formed using the request parameters.

Reports created via the WS API will show up as scheduled reports with the name WS Report Task in the CC-SG Admin Client in the Administration > Tasks section. CC-SG Admin Client users can see and delete these reports.

The WS system should handle deleting the tasks created using the WS API. The client should call deleteReport() for every call to runReport(). See ***deleteReport*** (on page 22). 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. Wildcards allowed. (Optional: may be null or empty.)
  - String userIPAddress – Restrict results to those of this IP address. Wildcards allowed. (Optional: may be null or empty.)
  - 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. The only supported wildcard is the asterisk. Optional: may be null or empty.
    - 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[]

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

Appendix A Certificate Management

This appendix contains some tips on managing certificates. See the respective company'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 from a Web Browser (IE6)

1. Using the WS-API client machine that is configured in the CC-SG, open a CC-SG service URL via HTTPS and port 9443.

For example:
https://10.0.0.101:9443/CommandCenterWebServices/AuthenticationAndAuthorizationServicePort?wsdl

2. A Security Alert appears. Click View Certificate.
3. Click Details then Copy to File.
4. Use the Certificate Export Wizard to save the certificate to a file.

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

1. Save the PKCS12 certificate file on your client computer. See ***Add Web Services API Client Configuration on CC-SG.*** (see "Add Web Services API Client Configuration on CC-SG" on page 2)
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.

Using Microsoft Management Console to Manage Certificates

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 B 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

As of this writing, Java API for XML Web Services (JAX-WS) 2.1 is the most recent specification for Java Web Services. Glassfish contains a JAX-WS implementation under Metro which can be found at https://jax-ws.dev.java.net/. 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/CommandCenterWebSer
vices/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 Computer,
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 Computer, Inc. All
rights reserved.
 * Reproduction of any element without the prior
written consent of
 * Raritan Computer, Inc. is expressly forbidden.
 */


import
security.service.webservice.bl.cc.raritan.com.Authent
icationAndAuthorizationService;
import
security.service.webservice.bl.cc.raritan.com.CCSGAut
henticationAndAuthorizationService;
import
security.service.webservice.bl.cc.raritan.com.types.*
;


import
node.service.webservice.bl.cc.raritan.com.NodeManagem
entService;
```

```
import
node.service.webservice.bl.cc.raritan.com.CCSGNodeMan
agementService;

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.Authent
icationAndAuthorizationException 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.getAuthenticationAndAuthorizationServic
ePort();
    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.Authent
icationAndAuthorizationException 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.NodeManagem
entException 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.Authent
icationAndAuthorizationException ex )
    {
      auth_exception_handler( ex, "signOff()" );
    }


  }
}
```

Appendix C Web Services Development in C#

The following sections describe how to create a Web Services client for the CCSG written in C#. This description is based on Microsoft Visual Studio 2008 and IE6 running on Windows XP. Other methods of Web Service creation are available in Visual Studio 2008, but this document solely covers services through Windows Communication Foundation (WCF) and .Net Framework Version 3.5.

In This Chapter

Using a CC-SG Web Service in a Project

1. Choose Project > Add Service Reference using the CC-SG WSDL URL for the intended service, such as AuthenticationAndAuthorization.
2. Right click *app.config* in Solution Explorer. You might have to open *WCF Service Configuration Editor* under *Tools* first. Choose *Edit WCF Configuration*.
   a. Under *Bindings*, select *New Binding Configuration*.
   b. Choose *customBinding*.
   c. Remove *httpTransport* extension if it is already entered as a *Stack Element*.
   d. Add *httpsTransport,* then edit it to set *RequireClientCertificate* to *True*.
   e. Select the *textMessageEncoding* extension and set *MessageVersion* to *Soap11*.
   f. Select the service under *Client->Endpoints*.
   g. Change the *Binding* to *customBinding*.
   h. Change *BindingConfiguration* to the *customBinding* that you just created.
   i. 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.

   j. Save the configuration.
3. Edit the C# source back in Visual Studio 2008.

4. Set a call back for *ServerCertificateValidationCallback* so that the client will accept the CCSG's server certificate. A simple method is to trust the certificate if it matches the CCSG server certificate that you have explicitly saved from the CCSG. See Saving the CCSG's Server Certificate Using IE6.

```
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);
   };
```

5. Create an instance of the service proxy.
6. Tell the proxy which client certificate to use through the *ClientCredentials* member. You should save the client certificate into a store. See Installing the Client Certificate into the Store. Specify the store where you installed the certificate along with a name that the search method can use to find the certificate. In this example, 10.0.0.150 is the address and FQDN of the client host and the certificate is stored in the *Personal* section of the *Current User* store.

```
auth_service.ClientCredentials.ClientCertificate.SetCertificate(
   StoreLocation.CurrentUser,
   StoreName.My,
   X509FindType.FindBySubjectName,
   "10.0.0.150");
```

7. Create an instance of the service's parameters class (signOn for instance).
8. Set the parameters.
9. Use the proxy and parameters object to access the service.
10. You can receive the response through the returned object (signOnResponse for example).

Sample Application for C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography.X509Certificates;
using System.Net;
using System.Net.Security;
using System.ServiceModel;


namespace CCWSClient_WCF
{
    class Program
    {
        static string cc_address = "10.0.0.101";
        static string user = "gregor";
        static string password = "pass123";
        static string session_id;


        static void
quit(CCAuthentication.AuthenticationAndAuthorizationS
erviceClient auth_service)
        {
            if (auth_service != null)
            {
                try
                {
                    CCAuthentication.signOff sign_off
= new CCAuthentication.signOff();
                    sign_off.String_1 = user;
                    sign_off.String_2 = session_id;
                    CCAuthentication.signOffResponse
response = auth_service.signOff(sign_off);
```

```
                    Console.WriteLine("signOff
Response: " + response.result);
                }
                catch (Exception exception)
                {

Console.WriteLine(exception.ToString());
                }
            }


            Console.WriteLine("Press a key to
exit.");
            Console.ReadKey(true);
            Environment.Exit(0);
        }


        static void Main(string[] args)
        {

ServicePointManager.ServerCertificateValidationCallba
ck =
                delegate(Object obj, X509Certificate
certificate, X509Chain chain, SslPolicyErrors errors)
                {
                    X509Certificate
server_certificate =
X509Certificate.CreateFromCertFile(@"C:\cc_certificat
e.cer");
                    return
server_certificate.Equals(certificate);
                };



CCAuthentication.AuthenticationAndAuthorizationServic
eClient auth_service =
```

```
                new
CCAuthentication.AuthenticationAndAuthorizationServic
eClient();


auth_service.ClientCredentials.ClientCertificate.SetC
ertificate(

                StoreLocation.CurrentUser,

                StoreName.My,

                X509FindType.FindBySubjectName,

                "10.0.0.150");


            // service parameters

            CCAuthentication.signOn sign_on = new
CCAuthentication.signOn();

            sign_on.String_1 = user;

            sign_on.String_2 = password;

            // access the service

            Console.WriteLine("Connecting to: " +
cc_address);

            try

            {

                auth_service.Open();

                CCAuthentication.signOnResponse
response = auth_service.signOn(sign_on);

                session_id = response.result;

                Console.WriteLine("signOn response: "
+ response.result);

            }

            catch (Exception exception)

            {


Console.WriteLine(exception.ToString());

                quit(null);

            }
```

```
            // renameNode service
            CCNode.NodeManagementServiceClient
node_service = new
CCNode.NodeManagementServiceClient();

node_service.ClientCredentials.ClientCertificate.SetC
ertificate(
                StoreLocation.CurrentUser,
                StoreName.My,
                X509FindType.FindBySubjectName,
                "10.0.0.150");
            CCNode.renameNode rename = new
CCNode.renameNode();
            rename.String_1 = session_id;
            Console.Write("Name of node to change:
");
            rename.String_2 = Console.ReadLine();
            Console.Write("New node name: ");
            rename.String_3 = Console.ReadLine();
            try
            {
                CCNode.renameNodeResponse response =
node_service.renameNode(rename);
                Console.WriteLine("renameNode
response: " + response.result);
            }
            catch (Exception exception)
            {

Console.WriteLine(exception.ToString());
                quit(auth_service);
            }

            quit(auth_service);
```

```
                    }
                }
            }
```

Index

U

W

U.S./Canada/Latin America

Monday - Friday
8 a.m. - 6 p.m. ET
Phone: 800-724-8090 or 732-764-8886
For CommandCenter NOC: Press 6, then Press 1
For CommandCenter Secure Gateway: Press 6, then Press 2
Fax: 732-764-8887
Email for CommandCenter NOC: tech-ccnoc@raritan.com
Email for all other products: tech@raritan.com

China

Beijing
Monday - Friday
9 a.m. - 6 p.m. local time
Phone: +86-10-88091890

Shanghai
Monday - Friday
9 a.m. - 6 p.m. local time
Phone: +86-21-5425-2499

GuangZhou
Monday - Friday
9 a.m. - 6 p.m. local time
Phone: +86-20-8755-5561

India

Monday - Friday
9 a.m. - 6 p.m. local time
Phone: +91-124-410-7881

Japan

Monday - Friday
9:30 a.m. - 5:30 p.m. local time
Phone: +81-3-3523-5991
Email: support.japan@raritan.com

Europe

Europe
Monday - Friday
8:30 a.m. - 5 p.m. GMT+1 CET
Phone: +31-10-2844040
Email: tech.europe@raritan.com

United Kingdom
Monday - Friday
8:30 a.m. to 5 p.m. GMT
Phone +44(0)20-7090-1390

France
Monday - Friday
8:30 a.m. - 5 p.m. GMT+1 CET
Phone: +33-1-47-56-20-39

Germany
Monday - Friday
8:30 a.m. - 5:30 p.m. GMT+1 CET
Phone: +49-20-17-47-98-0
Email: rg-support@raritan.com

Melbourne, Australia

Monday - Friday
9:00 a.m. - 6 p.m. local time
Phone: +61-3-9866-6887

Taiwan

Monday - Friday
9 a.m. - 6 p.m. GMT -5 Standard -4 Daylight
Phone: +886-2-8919-1333
Email: support.apac@raritan.com