Web Services API Administration
Avaya Communication Server 1000

Release 7.6
NN43001-640
Issue 04.01
March 2013

Notice

While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

Documentation disclaimer

“Documentation” means information published by Avaya in varying mediums which may include product information, operating instructions and performance specifications that Avaya generally makes available to users of its products. Documentation does not include marketing materials. Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of documentation unless such modifications, additions, or deletions were performed by Avaya. End User agrees to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End User.

Link disclaimer

Avaya is not responsible for the contents or reliability of any linked websites referenced within this site or documentation provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

Warranty

Avaya provides a limited warranty on its hardware and Software (“Product(s)”). Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya’s standard warranty language, as well as information regarding support for this Product while under warranty is available to Avaya customers and other parties through the Avaya Support website: http://support.avaya.com. Please note that if you acquired the Product(s) from an authorized Avaya reseller outside of the United States and Canada, the warranty is provided to you by said Avaya reseller and not by Avaya. “Software” means computer programs in object code, provided by Avaya or an Avaya Channel Partner, whether as stand-alone products or pre-installed on hardware products, and any upgrades, updates, bug fixes, or modified versions.

Licenses

THE SOFTWARE LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE, HTTP://SUPPORT.AVAYA.COM/LICENSEINFO ARE APPLICABLE TO ANYONE WHO DOWNLOADS, USES AND/OR INSTALLS AVAYA SOFTWARE, PURCHASED FROM AVAYA INC., ANY AVAYA AFFILIATE, OR AN AUTHORIZED AVAYA RESELLER (AS APPLICABLE) UNDER A COMMERCIAL AGREEMENT WITH AVAYA OR AN AUTHORIZED AVAYA RESELLER. UNLESS OTHERWISE AGREED TO BY AVAYA IN WRITING, AVAYA DOES NOT EXTEND THIS LICENSE IF THE SOFTWARE WAS OBTAINED FROM ANYONE OTHER THAN AVAYA, AN AVAYA AFFILIATE OR AN AVAYA AUTHORIZED RESELLER; AVAYA RESERVES THE RIGHT TO TAKE LEGAL ACTION AGAINST YOU AND ANYONE ELSE USING OR SELLING THE SOFTWARE WITHOUT A LICENSE. BY INSTALLING, DOWNLOADING OR USING THE SOFTWARE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE INSTALLING, DOWNLOADING OR USING THE SOFTWARE (HEREINAFTER REFERRED TO INTERCHANGEABLY AS “YOU” AND “END USER”), AGREE TO THESE TERMS AND CONDITIONS AND CREATE A BINDING CONTRACT BETWEEN YOU AND AVAYA INC. OR THE APPLICABLE AVAYA AFFILIATE (“AVAYA”).

Heritage Nortel Software

“Heritage Nortel Software” means the software that was acquired by Avaya as part of its purchase of the Nortel Enterprise Solutions Business in December 2009. The Heritage Nortel Software currently available for license from Avaya is the software contained within the list of Heritage Nortel Products located at http://support.avaya.com/LicenseInfo under the link “Heritage Nortel Products”. For Heritage Nortel Software, Avaya grants Customer a license to use Heritage Nortel Software provided hereunder solely to the extent of the authorized activation or authorized usage level, solely for the purpose specified in the Documentation, and solely as embedded in, for execution on, or (in the event the applicable Documentation permits installation on non-Avaya equipment) for communication with Avaya equipment. Charges for Heritage Nortel Software may be based on extent of activation or use authorized as specified in an order or invoice.

Copyright

Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, or hardware provided by Avaya. All content on this site, the documentation and the Product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, and or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Third Party Components

“Third Party Components” mean certain software programs or portions thereof included in the Software that may contain software (including open source software) distributed under third party agreements (“Third Party Components”), which contain terms regarding the rights to use certain portions of the Software (“Third Party Terms”). Information regarding distributed Linux OS source code (for those Products that have distributed Linux OS source code) and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available in the Documentation or on Avaya’s website at: http://support.avaya.com/Copyright. You agree to the Third Party Terms for any such Third Party Components.

Note to Service Provider

The Product may use Third Party Components that have Third Party Terms that do not allow hosting and may need to be independently licensed for such purpose.

Preventing Toll Fraud

“Toll Fraud” is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf). Be aware that there can be a risk of Toll Fraud associated with your system and that, if Toll Fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Toll Fraud intervention

If you suspect that you are being victimized by Toll Fraud and you need technical assistance or support, call Technical Service Center Toll Fraud Intervention Hotline at +1-800-643-2353 for the United States and Canada. For additional support telephone numbers, see the Avaya Support website: http://support.avaya.com. Suspected security vulnerabilities with Avaya products should be reported to Avaya by sending mail to: securityalerts@avaya.com.

Trademarks

The trademarks, logos and service marks (“Marks”) displayed in this site, the Documentation and Product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, or other third

parties. Users are not permitted to use such Marks without prior written consent from Avaya or such third party which may own the Mark. Nothing contained in this site, the Documentation and Product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party.

Avaya is a registered trademark of Avaya Inc.

All non-Avaya trademarks are the property of their respective owners, and “Linux” is a registered trademark of Linus Torvalds.

Downloading Documentation

For the most current versions of Documentation, see the Avaya Support website: http://support.avaya.com.

Contact Avaya Support

See the Avaya Support website: http://support.avaya.com for product notices and articles, or to report a problem with your Avaya product. For a list of support telephone numbers and contact addresses, go to the Avaya Support website: http://support.avaya.com, scroll to the bottom of the page, and select Contact Avaya Support.

Contents

Chapter 1: New in this release

The following chapter details *Avaya Web Services API Administration*, (NN43001-640) support for Avaya Communication Server 1000 (Avaya CS 1000) Release 7.6.

Features

There are no updates to the feature descriptions in this document.

Other changes

See the following section for information about changes that are not feature-related.

Revision History

March 2013	Standard 04.01. This document is up-issued to support Avaya Communication Server 1000 Release 7.6.
April 2011	Standard 03.02. This document is up-issued to support Avaya Communication Server 1000 Release 7.5.
November 2010	Standard 03.01. This document is up-issued to support Avaya Communication Server 1000 Release 7.5.
June 2010	Standard 02.01. This document is up-issued to support Communication Server 1000 Release 7.0.
August 2009	Standard 01.02. The document is up-issued to include changes in the section Code examples.
May 2009	Standard 01.01. This is a new document to support Communication Server 1000 Release 6.0.

Chapter 2: Customer service

Visit the Avaya Web site to access the complete range of services and support that Avaya provides. Go to HTTP://WWW.AVAYA.COM or go to one of the pages listed in the following sections.

Getting technical documentation

To download and print selected technical publications and release notes directly from the Internet, go to HTTP://WWW.AVAYA.COM/SUPPORT.

Getting product training

Ongoing product training is available. For more information or to register, you can access the Web site at HTTP://WWW.AVAYA.COM. From this Web site, you can locate the Training contacts link on the left-hand navigation pane.

Getting help from a distributor or reseller

If you purchased a service contract for your Avaya product from a distributor or authorized reseller, contact the technical support staff for that distributor or reseller for assistance.

Getting technical support from the Avaya Web site

The easiest and most effective way to get technical support for Avaya products is from the Avaya Technical Support Web site at HTTP://WWW.AVAYA.COM.

Chapter 3: Introduction

This document describes the web services feature available with Avaya Unified Communications Management (Avaya UCM) Common Services (CS) server for Avaya Communication Server 1000 (Avaya CS 1000) and how to write a client application to use these web services. Use this information in conjunction with the online documentation for the various web services.

Prerequisites

An understanding of the Avaya UCM CS framework and web services technology is required. The examples in this document are based on the assumption that you have a working knowledge of the following technologies:

- Java programming language
- Apache Ant build tool
- Java API for XML Web Services (JAX-WS) 2.1.4
- Web services

Avaya also recommends you must have some knowledge of Web Service Definition Language (WSDL).

Important:

Client applications using the web services interface can be developed using any programing language and development toolkit. However, this document covers only developing clients using Java, JAX-WS and Microsoft .NET framework.

Overview

The Web services feature allows remote management and configuration of Avaya CS 1000 and Network Routing Service (NRS) systems. Web services allow application developers to write custom management applications, to accomplish custom or specialized tasks for CS 1000.

Four web services are available depending on what management applications (EM or NRSM) are deployed on a UCM CS server. The following table displays the types of services deployed depending on the type of server that is installed. The deployment type can be Element Manager (EM) or Network Routing Service (NRS) irrespective of whether the server is primary, backup, or member.

Table 1: Types of Services

Service	Description	Deployment type
CS 1000	Provides access to functionality available on the CS 1000. This includes overlay access and selected Signaling Server and Media Card CLI command access.	Element Manager
Phone Provisioning	Provides a programmable way of configuring phones.	Element Manager
Network Routing Service (NRS)	Provides access to functionality available through NRS Manager (NRSM).	NRS
Managed Element Registry	Provides access to query managed elements available in UCM that support one of the other services.	Element Manager and NRS

In this table, the deployment types (as per the Deployment Manager application) are listed, not the hardware type. The web services are deployed when the corresponding management applications (EM or NRSM) are deployed on a server. The web services can be deployed to any server type on which the corresponding management application can be deployed. For more information about how service requests are handled, see Managed Element Registry Service on page 27.

All services are deployed using industry-standard features and are compliant with the following standards:

- Web Services Interoperability (WS-I) Basic Profile 1.1
- WSDL 1.1
- Simple Object Access Protocol (SOAP) 1.1 and 1.2

Each service is exposed using document/literal wrapped binding style. As a result, it is possible to write client application that uses these web services in any language, using any toolkit that is compliant to the same standards. However, all examples in this document are written in Java and use the Java API for XML Web Services (JAX-WS) toolkit, available from Sun, in combination with Apache Ant. Procedures to create clients using other toolkits can vary drastically.

Document conventions

In this document, the text enclosed in angular brackets, such as <server-name> indicates that the text is to be replaced with the relevant text for the setup. For example, server1@avaya.com.

Security

The web services deployed on the UCM CS server use the UCM security model to secure all web service requests. The web service requests are secured by the following methods:

- Secure Sockets Layer (SSL) communication: The client application must use the downloaded UCM certificate for each server.
- Hypertext Transfer Protocol (HTTP) basic authentication: You must configure a valid user name and password on the client, to make a web service call.
- SecureObjectId HTTP request parameter: A Secure Object Id is required for most HTTP requests to the UCM Web server, to identify the intended managed element request.
- Role based permissions in UCM CS server: You must configure the permission to access a web service.

There is no web service available to configure security settings on the UCM CS server. This configuration must occur by using the Web based graphical interface.

SSL Communication

All HTTP requests made to the UCM CS server must be through SSL; this requires the client application to obtain a valid security certificate and use it for all communication. For more information about how to obtain the certificate and create a local keystore, see Creating a keystore on page 19.

The security framework requires all Uniform Resource Locators (URLs) used to access the web services, to end with a .secure extension. For more information about the URLs value for each service, see UCM web services on page 27.

HTTP Basic Authentication

All HTTP requests made to the UCM CS server must contain a valid user name and password to access functionality. If an invalid user name or password is provided, the server returns an HTTP error. For more information about how to configure the user name and password, see Creating a client application on page 21.

Secure Object ID

The UCM CS framework handles Managed Elements, which are logical instances of a system in the network to be managed. For example, a CS 1000 or an NRS. When a new managed element is deployed into the network, the framework considers the element to be a new Secure Object. When the element is created, the UCM CS framework generates a Secure Object ID value. The Secure Object ID is a unique text string, such as a23c8c46248a484f--ea3531e-10ec0658485--7ffd.

The Secure Object ID value uniquely identifies each system and is used in most HTTP requests, to identify the element on which an operation is to be invoked. The secure object ID must be included in the URL for all requests as shown in the following example.

```
https://<server-name>/bccWebService_1_0/SECURE_OBJECT_ID/
com.nortel.ems.CS1000/4589094e-280c-11dd-8079-81412046fbff/
BccWebService.secure
```

In this example, the Secure Object Id identifies the CS 1000 system for which the request is intended.

To retrieve a Secure object ID value, you can use one of the following methods:

1. Retrieve the list of available Managed Elements by using the ManagedElementRegisty service as described in Managed Element Registry Service on page 27. This service is a programmatic way to retrieve the service URLs, which include the Secure Object ID.
2. The URL for each element contains the unique secure object ID value. You can copy the Secure Object ID from the URL and paste in the code directly. However, you must use the correct syntax. For example, the following figure shows the Secure Object ID (circled in red):

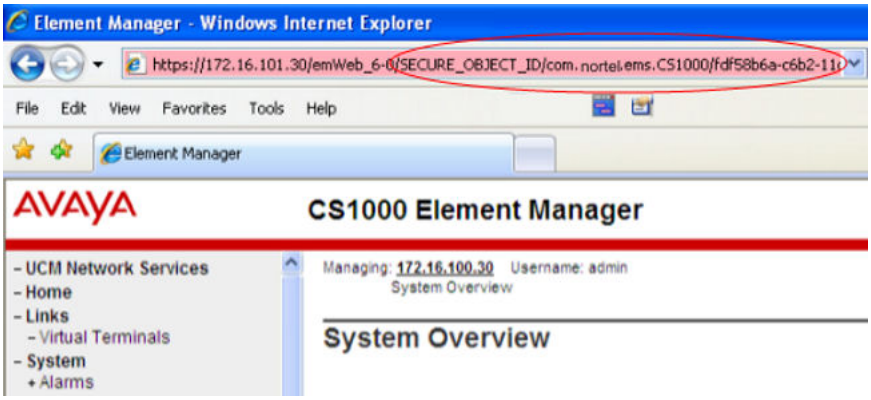

Figure 1: Copying and pasting the Secure Object ID from the URL

Permissions

UCM CS defines the following permissions to control access to web services:

- Web Service – Phone Configuration: Control access to the Phone Provisioning web service and can be configured for any element of CS 1000 type.
- Web Service – Overlay and Signaling Server: Control access to the CS 1000 web service and can be configured for any element of CS 1000 type.
- Web Service – NRSM: Control access to the NRSM web service and configure it for any NRS-element type.

These permissions are disabled by default. When invoking a web service call, if the appropriate permission is not enabled for the required user, an error message is returned. The following is an example of an error message:

```
The server sent HTTP status code 403: Access is denied.
```

Any user with an account on the UCM CS server can access the managed element registry service as long as at least one of the above permissions are enabled for that user.

Error handling

Each method in each web service on the UCM CS framework, throws a single type of exception:

- ElementRegistryServiceException – thrown by the Managed Element Registry Service
- Cs1000ServiceException – thrown by the CS 1000 web service
- BccServiceException – thrown by the Phone Provisioning web service
- NrsmServiceException – thrown by the NRSM web service

In this document these exceptions are referred to as Service Exceptions.

A service exception contains details about the type of error that occurs during execution. Each exception is the same, but have different names to avoid clashes if a client is using multiple services in the same file. A service exception contains the following information:

- A unique error code that indicates errors such as an invalid parameter, or an internal execution error. Each error code consists of a four-letter prefix and a four-digit number such as NRSM0001, or CS1K0030. This value is always present.
- An error description explains the problem and contains a format string {0} which can be used to insert the value that caused the error. For example, “The IP address {0} is invalid”. This value is always present.

- A variable indicating the value which caused the error. For example, it can be an invalid IP address value. This value is a string and not always present.
- A pre formatted message that combines the details of the error as described below.

The error code is the important part of the exception. The error codes are used to look up the cause of the problem. The description and variable are additional information. The description and variable can be combined to create a readable message for the end user by using the MessageFormat.format() method. For example, combine the preceding values as follows.

```
MessageFormat.format( exception.getFaultInfo()
.getErrorDescription(), exception.getFaultInfo()
.getVariable() )
```

the resulting string is “The IP address a.b.c.d is invalid.”

A client can use a separate description and variable to create their own custom or localized version of the error message if desired. This way, the client can use the code to retrieve a custom error message, which can be combined with the variable to create a custom error string.

Generally, the error codes indicate the exact cause of the error. However, when internal errors such as problems accessing a database, or internal programming errors occur, the error code indicates that an internal error has occurred.

All errors are thrown as service exceptions, which contain error codes identifying the source of the problems. When an error occurs, a log message is created. The log message contains details of what caused the error and are most valuable for determining what caused internal errors.

The web services check all input parameters for basic validity such as range validation and values that cannot be null. If an error is detected for an input parameter, an appropriate error code and message is returned . For a list of error codes and messages, see Error code listing on page 37. However, the web services do not perform complex dependency checking between fields such as valid features for a phone type or port numbers for enabled services. The underlying system performs this type of verification. For example, the CS 1000 service verifies that the overlay command is not null, but does not check the validity of the overlay command itself. An error from the underlying system is returned in some form, when these types of errors are detected.

For Microsoft .NET Clients

The error String format described earlier also works for Microsoft .NET clients. For example C# has the method String.format() which does the same type of formatting. So, the following results:

```
String.format(errorDescription,variable)
```

However, it is more difficult to obtain the description and variable values. For more information on Microsoft .NET clients, see Microsoft .NET Clients on page 23.

Performance

Some interface methods described in this document can take a long time to execute under certain circumstances because of slow response time from the underlying server, or time taken to process large requests. Two major concerns arise: execution time and memory usage on the client and the server.

Memory must be closely monitored. When you invoke certain methods in the web services, such as searching for phones in Phone Provisioning, the maximum heap size of the client JVM can be increased to 256 MB or higher to avoid out-of-memory errors on the client. This is due to the large amount of data that can be returned from these calls. The required size of the heap depends on the amount of data that can be retrieved. To set the heap size, an argument such as -Xms500M-Xmx500M can be passed to your client application. This will set the initial and maximum heap sizes.

When large amounts of data are sent or retrieved from a web service call, it can take a long time to process the request. The request will not timeout on the server side, but client applications must be written accordingly to prevent the client application from locking while waiting for a response. The amount of data requested in a single request must be minimal to avoid these issues.

Chapter 4: Building web service clients

Introduction

This chapter describes the general steps to create a client for a web service. For the purpose of an example, the Managed Element Registry and Avaya Communication Server 1000 (Avaya CS 1000) web services are used to give an overview of all steps required in a typical scenario. The steps are the same for all other services. For more information about each service, see UCM web services on page 27.

Perform the following basic steps, to create a web service client:

- Creating a keystore on page 19
- Generating client stub code on page 20
- Creating a client application on page 21

These steps use the Ant build tool for generating and compiling code. This chapter shows only the relevant part of the build file. For a complete code listing, see Code examples on page 47.

Requirements

For this example, ensure you have the following software installed to create a client application:

- Java Development Kit (JDK) 5.0 or later
- JAX-WS 2.1.4 or later
- Apache Ant 1.6.5 or later
- An Avaya Unified Communication Management (Avaya UCM) CS application server with the desired configuration

Creating a keystore

Perform the following steps to create a keystore:

1. Get a copy of the certificate contents.
   - Open a Web browser and navigate to the URL of the server. Log on using a valid user name and password to access the security features (for example, a user with the PowerUser role).

- Under the Security branch in the navigator, click the Certificates link.
- Click the Private Certificate Authority tab.
- Click the Download button to download the certificate contents. Save the file with a .cer extension. For example: D:\keyStores\ucmCertificate.cer.

2. Using the keytool utility available in the JDK, create a keystore which contains the certificate using the command (the JDK bin directory must be in your path, or you must use the full path for the keytool):

```
D:\KeyStores>keytool.exe -import -noprompt -trustcacerts -
alias ucmCert -file ucmCertificate.cer -keyStore ucmKeystore
-storepass password
```

You can change the alias, file, keystore name, and password values to suit your application.

Generating client stub code

The first step is to generate the client stub code that can be used by the client application. To generate the stub code, the wsimport tool which is available with JAX-WS is used with the Ant build tool. As this example accesses both the Managed Element Registry service and the Avaya CS 1000 service, client code for both services must be generated. The following steps are required:

1. Define the Ant task for wscompile. This taskdef requires to create a path element which points to the necessary JAR files from the JAX-WS.

```
<taskdef name="wsimport"classname=
"com.sun.tools.ws.ant.WsImport">
<classpath refid="jaxws.classpath"/>
</taskdef>
```

2. To invoke the wsimport command, create an Ant target. This task will read the WSDL file and use it to generate the client stub code. This generated code is placed in the ‘generated.dir’ directory. The following example shows running the command for the element registry service. Similar commands must be run for any other services.

```
<wsimport keep="true" sourcedestdir="${generated.dir}"
destdir="${classes.dir}" extension=
"true" verbose="true"
fork="true" wsdl="${mer.wsdl.file}">
</wsimport>
```

3. Run the preceding ant task to create the stub code.

For more information on the wsimport tool or details on how to run from the command line, see the documentation available from Sun (https://jax-ws.dev.java.net/).

Creating a client application

Now, you can create an actual client application.

The following three parameters must be configured to invoke a web service method:

1. The location and password of the Avaya UCM keystore.
2. The HTTP basic authentication user name and password. The specified user name and password must be allowed to access the desired service.
3. The URL of the service on which to invoke the methods. This in turn is made of three parts:
   - The IP address and port number of the UCM CS server.
   - The Secure Object ID for the element on which the method must be invoked. This is required for all services except the ManagedElementRegistry service. For more information about secure object ID, see Secure Object ID on page 14.
   - The servlet mapping of the service.

For convenience, the element registry service provides a way to query this URL so that it does not have to be constructed at the client side. Use the managed element registry service to query this information, as it will automatically determine the correct object ID, and server fully qualified domain name based on the element that is queried.

Example

The first step is to setup the keystore location, password and the servers user name and password. This is done by setting environment variables which can be done in the constructor. The security must be configured prior to making the web service calls.

Listing: Configuring security

```
public SampleClient()
{
//Set the location and password for keystore.
System.setProperty("javax.net.ssl.trustStore",
"D://KeyStores//ucmKeystore");
System.setProperty("javax.net.ssl.trustStorePassword",
"password");
}
```

Listing: Determine the URL of the CS 1000 service

```
private String getServiceUrl(final ServiceType serviceType)
{
String url = null;
try
{
final ManagedElementRegistry proxy =
new ManagedElementService()
.getManagedElementServicePort();
((BindingProvider) proxy).getRequestContext().put(
```

```
BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"https://myserver.ca.avaya.com/
managedElementRegistryService_1_0/" +
"ManagedElementRegistryWebService.secure" );
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.USERNAME_PROPERTY, USERNAME );
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.PASSWORD_PROPERTY, PASSWORD );
// Assume there is exactly one CS 1000 element returned.
final ManagedElement element =
(ManagedElement)proxy.getManagedElementsByType(
ElementType.CS_1000).get(0);
for (Service service : element.getServices())
{
if (service.getType() == serviceType)
{
url = service.getUrl();
}
}
return url;
}
catch (ElementRegistryServiceException_Exception e)
{
//...
}
}
```

You must know the address of the UCM CS server you are accessing, to access the managed element registry service. However, that is the only service address that must be known. The element registry service is used to query elements for the element type or name desired to find the desired service type and the URL associated with that service.

Important:

This URL is specific to the element and contains the correct UCM CS server, URL, and secure object ID.

Listing: Calling a CS 1000 service method

```
public void executeOverlayCommand()
{
final Cs1000Management proxy =
new Cs1000Service().getCs1000ServicePort();
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
getServiceUrl(ServiceType.CS_1000));
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.USERNAME_PROPERTY, USERNAME );
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.PASSWORD_PROPERTY, PASSWORD );
try
{
System.out.println(proxy.executeOverlayCommand(
"<?xml version=\"1.0\"?>"+
"<LDOVL>"+
"<LD>21</LD>"+
"<REQ>ltm</REQ>"
"</LDOVL>"));
}
catch (Cs1000ServiceException_Exception e)
{
//...
```

```
}
}
```

This code calls the getServiceUrl() method to obtain the URL for the CS 1000 service available. This URL is then configured as the endpoint address.

Microsoft .NET Clients

You can create clients for these web services by using Microsoft .NET. This section briefly describes the Microsoft .NET client. These examples use C#, Microsoft .NET version 3.5 and Visual C# 2008 Express Edition.

Sample C# client

To create a C# client, visual studio can retrieve the WSDL directly from the server without downloading the WSDL. Once the web service reference is added to your project, the following code example shows how to invoke a service call.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using System.ServiceModel.Description;
namespace UcmWebServiceUserGuide_1_0
{
class SampleClient
{
static void Main(string[] args)
{
servicePointManager.ServerCertificateValidation
Callback =
TrustAllCertificateCallback;
ServicePointManager.Expect100Continue = false;
SampleClient client = new SampleClient();
client.executeOverlayCommand();
}
//Executes a basic CS 1000 overlay command and prints the
//result.
public void executeOverlayCommand()
{
Cs1000Client.Cs1000ManagementClient service =
new Cs1000Client.Cs1000ManagementClient();
service.ClientCredentials.UserName
.User name= "username";
service.ClientCredentials.UserName
.Password = "password";
service.Endpoint.Address =
new System.ServiceModel.EndpointAddress(
getServiceUrl());
Cs1000Client.executeOverlayCommand command =
new Cs1000Client.executeOverlayCommand();
command.overlayCommand =
"<?xml version=\"1.0\"?>"+
"<LDOVL>"+
```

```
"<LD>21</LD>"+
"<REQ>1tm</REQ>"+
"</LDOVL>";
Console.WriteLine (service.executeOverlayCommand (
command) .@return);
}
//Returns the URL for the CS 1000 service on the first
//element found. Assumes that only one CS 1000 element
//is configured.
private String getServiceUrl()
{
String result = null;
ManagedElementClient.ManagedElementRegistryClient
service =
new ManagedElementClient.ManagedElementRegistry
Client();
service.ClientCredentials.UserName.UserName =
"username";
service.ClientCredentials.UserName.Password =
"password";
//Assume there's only one element configured.
ManagedElementClient.managedElement element =
service.getAllManagedElements(
new ManagedElementClient.
getAllManagedElements())[0];
ManagedElementClient.service[] services =
element.services;
for (int i = 0; i <services.Length; i++)
{
if (services[i].type ==
ManagedElementClient.serviceType.CS 1000)
{
result = services[i].url;
}
}
return result;
}
//Accepts all certificates. If you want to really check
//the validity of the certificate against a local keystore
//you can do it here.
//
private static bool TrustAllCertificateCallback(object
sender,
X509Certificate cert, X509Chain chain,
SslPolicyErrors errors)
{
return true;
}
}
}
```

You must be aware of the following issues for this client

1. You need not create a local keystore with the SSL certificate. For example, the callback method TrustAllCertificateCallback() always returns true. The framework calls this method. You can implement this method to retrieve the local keystore and verify the contents of the certificate presented by the server during the transaction.
2. C# does not use a checked exception, and all exceptions caught by the client are wrapped in SoapException. So, the client code is not forced to deal with the exceptions thrown from web service that are sent from the server. This means that no get methods exists for the error code, and description. To obtain the details of

the exception, the client application must catch SoapException and manually extract the exception detail (in XML format) to print the error details.

3. To have the C# client to work correctly, the generated app.config file must be updated to change the security mode on each service and to configure the clientCredentialType as shown in the following example.

```
<security mode="Transport">
<transport clientCredentialType="Basic"
proxyCredentialType="None"
realm=""/>
<message clientCredentialType="UserName"
algorithmSuite="Default"/>
</security>
```

Chapter 5: UCM web services

Introduction

This chapter contains the following topics:

Managed Element Registry Service

You can use the Managed Element Registry service to query managed elements to find the URLs used in the services. This service provides access to the following functionality:

- Get all managed elements
- Get managed elements by name
- Get managed elements by ID
- Get managed elements by type

The methods in this service return only managed elements of type Avaya Communication Server 1000 (Avaya CS 1000) or NRS as those are the only elements that support the web services described in this document.

The data returned from each of these methods contains the exact URL of the service to invoke (for more information about the ‘Service URLs’, see the following chapters), regardless of which server the managed element registry service is deployed on.

To invoke this service, the client application must know the address of the server where the managed element web service is located (can be on more than one server). You can then use this service to query which Phone Provisioning, Avaya CS 1000, or NRS services are available in the network. The query returns Managed Element instances which contain the exact URL to access the desired service.

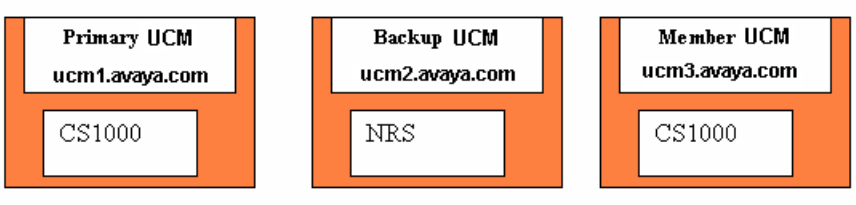

Figure 2: Element registry service

In this diagram, the element registry service is available on all three servers. If the client searches for available CS 1000 elements on the Primary Avaya Unified Communication Management (Avaya UCM) Common Services server, the following two Managed Element entries are returned:

1. ucm1.avaya.com with the URL property:

   ```
   https://ucm1.avaya.com/cs1000WebService_1_0/SECURE_OBJECT_ID/
   com.nortel.ems.CS1000/<object-id>/Cs1000WebService.secure
   ```

2. ucm3.avaya.com with the URL property:

   ```
   https://ucm3.avaya.com/cs1000WebService_1_0/SECURE_OBJECT_ID/
   com.nortel.ems.CS1000/<object-id>/Cs1000WebService.secure
   ```

All query requests are internally redirected to the primary Avaya UCM server. To get the list of the available elements, the results are not dependent on the servers used for performing the query. When the query completes, you can use the URLs to invoke the desired service, so it is not necessary for the client to know which server in the network hosts the service. However, the client needs to search for the required name or type of the service to extract the URL property from the result.

Service URLs

You can download the WSDL for the Managed Element Registry web service from the following URL.

```
https://<server-name>/managedElementRegistryService_1_0/
ManagedElementRegistryWebService.secure?wsdl
```

Use the following endpoint address, to invoke methods on the service:

```
https://<server-name>/managedElementRegistryService_1_0/
ManagedElementRegistryWebService.secure
```

This address does not require secure object ID parameter.

You can view the online documentation from the following URL.

```
https://<server-name>/merWebServiceDocs_1_0/index.html
```

CS 1000 Service

The CS 1000 web service provides access to the following functionality:

- Call server overlays
- Selected Signaling Server and Media Card commands

The Call Server and Signaling Server commands are grouped into a single CS 1000 service because they form a single entity. However, the commands can run on various physical devices. Therefore, Signaling Server or Media Card commands have an IP address parameter which indicates the address of the Signaling Server or Media Card on which to run the command. The IP address is not required for the Call Server (overlay) command. The Call Server IP address is available directly from the UCM CS framework (configured when element is added) and is determined based on the secure object ID in the URL used to invoke the command.

In a co-resident installation (Call Server and Signaling Server on the same hardware), the IP address of the Call Server and Signaling Server are the same, however the commands are routed internally to the correct server.

The following overlays are not supported:

- LD 24
- LD 84
- LD 85

Call server overlays

There is a single method in the CS 1000 web service to run overlay commands on the Call Server. This method has the following signature.

```
String executeOverlayCommand(String overlayCommand)
throws Cs1000ServiceException;
```

This method takes a single parameter, the overlay command string. The XML format for command based overlays is as follows:

```
<?xml version="1.0"?>
<LDOVL>
<LD>overlay number </LD>
<PROMPT NAME-1>value</PROMPT NAME-1>
<PROMPT NAME-2>value</PROMPT NAME-2>
...
<PROMPT NAME-N>value</PROMPT NAME-N>
</LDOVL>
```

For maintenance overlays, the format of the command is slightly different—it has no prompts or responses. The maintenance overlays format is as follows.

```
<?xml version="1.0"?>
<LDOVL>
<LD>overlay number</LD>
<COMMAND>complete command as entered on the CLI</COMMAND>
</LDOVL>
```

The following XML loads overlay 21 and inputs the value ltm at the REQ prompt.

```
final String command =
"<?xml version=\"1.0\"?>" +
"<LDOVL>" +
"<LD>21</LD>" +
"<REQ>ltm</REQ>" +
"</LDOVL>";
System.out.println( service.executeOverlayCommand(command));
```

The result is returned as an XML string. For example, the following string is returned for the above command.

```
<?xml version="1.0"?>
<LDOVL>
<CUSTGRP>
<CUST>00</CUST>
<ROUTGRP>
<TKTP>ATVN</TKTP>
<ROUTE>0</ROUT>
<DES>ATVN_ROUTE</DES>
<MBERGRP>
<TN>002 0 00 01</TN>
<MBER>1</MBER>
<DES>TEST</DES>
</MBERGRP>
<MBERGRP>
<TN>004 00 01 </TN>
<MBER>2</MBER>
<DES>TEST</DES>
</MBERGRP>
</ROUTGRP>
<ROUTGRP>
<TKTP>ATVN</TKTP>
<ROUT>1</ROUT>
<DES>ATVN_VROUTE</DES>
</ROUTGRP>
<ROUTGRP>
<TKTP>ATVN</TKTP>
<ROUT>2</ROUT>
<DES>ATVN_VROUTE</DES>
</ROUTGRP>
<ROUTGRP>
<TKTP>ATVN</TKTP>
<ROUT>3</ROUT>
<DES>ATVN_VROUTE</DES>
</ROUTGRP>
<ROUTGRP>
<TKTP>TIE</TKTP>
<ROUT>4</ROUT>
<DES>TIE_VROUTE</DES>
</ROUTGRP>
```

```
<ROUTGRP>
<TKTP>ATVN</TKTP>
<ROUT>8</ROUT>
<DES>ATVN_ROUTE</DES>
</ROUTGRP>
<CUSTGRP>
```

The output of maintenance commands is slightly different. The returned values are surrounded with <RESULT> tags rather than specific prompt tags.

Access to overlay commands is restricted by using the user name and password properties associated with the CS 1000 element in UCM. If an invalid user name and password is configured in UCM, a service exception occurs containing the error code OVL428, which indicates that the user name and password are invalid.

Refreshing Call Server data

For some overlay commands, the client application can receive the following response from the executeOverlayCommand method.

```
<LDOVL>
<RESULT>WEB4005 TYPE</RESULT>
</LDOVL>
```

The WEB4005 code means that the data you attempted to change needs to be refreshed. To refresh data, run a PRT command before you try to change it. For example, attempting to send the following command.

```
<?xml version='1.0'>
<LDOVL>
<LD>97</LD>
<REQ>CHG</REQ>
<TYPE>SUPL</TYPE>
<SUPL>v184</SUPL>
</LDOVL>
```

Will result in the WEB4005 response. To solve this issue, you must refresh the data by sending the PRT command for the data.

```
<?xml version="1.0">
<LDOVL>
<LD>97</LD>
<REQ>PRT</REQ>
<TYPE>SUPL</TYPE>
</LDOVL>
```

This command will return the following response, which includes a VERSION tag. The VERSION tag contains a number (partial output shown) :

```
<?xml version="1.0">
<LDOVL>
```

```
<VERSION>2</VERSION>
```

```
<SUPL_GRP>
```

```
<SUPL>004</SUPL>
<SUPL>IPMG</SUPL>
<SLOT></SLOT>
...
</LDOVL>
```

In the change command, you must include the value in the VERSION tag as follows.

```
<?xml version="1.0">
<LDOVL>
<LD>97</LD>
```

```
<VERSION>2</VERSION>
```

```
<REQ>CHG</REQ>
<TYPE>SUPL</TYPE>
<SUPL>v184</SUPL>
</LDOVL>
```

Signaling Server and Media Card CLI commands

The CS 1000 web service provides access to run selected Signaling Server and Media Card CLI commands. These commands return the same information as that would be returned if the equivalent commands were invoked from Element Manager. For example, see the following commands:

```
service.getElectionInfo( "1.1.1.1" );
```

A string similar to the following will be returned:

```
Node ID : 556
Node Master   : Yes
Up Time       : 10 days, 1 hours, 11 mins, 40 secs
TN            : 00 00
Host Type     : Signaling Server
TLAN IP Addr  : 47.11.249.251
ELAN IP Addr  : 47.11.255.225
Election Duration       : 15
Wait for Result time    : 31
Master Broadcast period : 30
===== master tps =====
Host Type   TN            TLAN IP Addr
Signaling Server  00 00         47.11.249.251
Next timeout    : 29 sec
AutoAnnounce    : 1
Timer duration  : 60 (Next timeout in 8 sec)
====== all tps ======
Num  TN            Host Type   ELAN MAC           TLAN IP Addr     ELAN IP Addr
Up Time       NumOfSets  TimeOut

001  00 00         Signaling Server  00:02:b3:c5:52:0a  47.11.249.251
47.11.255.225    010 01:11:40  0          0

====== All cards in node configuration are registered ======
```

The client application determines how to display or parse this information.

Some commands in this interface work only on Signaling Servers, while others work only on Media Cards.

Service URL

You can download the WSDL for the CS 1000 Web service from the following URL.

```
https://<server-name>/cs1000WebService_1_0/Cs1000WebService
.secure?wsdl
```

Use the following endpoint address, to invoke methods on the service.

```
https://<server-name>/cs1000WebService_1_0/SECURE_OBJECT_ID/
com.nortel.ems.CS1000/<object-id>/Cs1000WebService.secure
```

You can view the online documentation from the following URL.

```
https://<server-name>/cs1000WebServiceDocs_1_0/index.html
```

Phone Provisioning Service

The basic client configuration service provides access to functionality available through overlays 10 and 11 on the CS 1000 Call Server. This service provides the same functionality from under the Phones link in Element Manager including the ability to perform the following tasks:

- Add phones
- Delete phones
- Modify phones
- Search for phones
- Move phones
- Swap phones

Important:

You must access the Phones link within Element Manager Web application before you use the Phone Provisioning web service. You must access the application from the Web to register the name and address of the element in the Phone Provisioning database. Alternatively, call retrieveSystem() method on the Phone Provisioning web service interface to initialize the database. This initialization occurs only after the first time either action occurs after a new installation.

Using the Phone Provisioning web service interface can cause the Phone Provisioning database to become unsynchronized with the Call Server configuration. This occurs when the Call Server enters default values for phone properties, which are not provided by the user. For example, if you add a new phone using the web service without configuring keys or classes of

service, the Call Server automatically creates default keys and classes of service. In this case, the new phone is added to the Phone Provisioning database without keys or classes of service. However, the phone added to the Call Server contains some values configured. As the web service and Phone Provisioning Web application can access only the database, the client cannot see the actual phone that exists on Call Server. Two methods are available to resynchronize the data:

- retrieveAndReconcileTelephones – retrieves all phones from the Call Server and ensures that the database is synchronized with all phones.

  ⚠ Caution:

  If a large number of phones are configured, this can take a long time to complete; use it with caution.

- retrieveSpecificTelephones – retrieves specific phones from the Call Server and ensures that the database is synchronized for those phones. Client applications must use this in most cases. You can quickly retrieve a subset of phones (the phones that are added or updated) rather than all phones, (provided the number of selected phones is small).

Service URL

You can download the WSDL for the Phone Provisioning Web service from the following URL.

```
https://<server-name>/bccWebService_1_0/BccWebService
.secure?wsdl
```

Use the following endpoint, to invoke methods on the service.

```
https://<server-name>/bccWebService_1_0/SECURE_OBJECT_ID/com.nortel.ems.CS1000/
<object-id>/BccWebService.secure
```

You can view the online documentation from the following URL.

```
https://<server-name>/bccWebServiceDocs_1_0/index.html
```

NRSM Service

The NRSM web service provides access to the following operations:

- Add, modify, or delete domains.
- Add, modify, or delete endpoints.
- Add, modify, or delete routes.
- Add, modify, or delete translators.
- Add, modify, or delete collaborative servers.

- Search for domains, endpoints, routes, and translators.
- Perform basic maintenance operations such as switching databases.

You cannot use the NSRM web service to perform routing tests, or other advanced maintenance operations available in the Web application.

Service URL

You can download WSDL for NRSM Web service from the following URL.

```
https://<server-name>/nrsmWebService_1_0/NrsmWebService.
secure?wsdl
```

Use the following endpoint, to invoke methods on the service.

```
https://<server-name>/nrsmWebService_1_0/SECURE_OBJECT_ID/
com.nortel.ems.NRS/<object-id>/NrsmWebService.secure
```

You can view the online documentation from the following URL.

```
https://<server-name>/nrsmWebServiceDocs_1_0/index.html
```

Chapter 6: Error code listing

Every web services described in this document have a unique set of error codes and messages. These codes and messages are returned through a thrown service exception, when an error occurs. All services define an “Internal Error” code indicating an error on the server and which client has no control. For all the error messages returned, an equivalent log is generated on the server. For more information about error handling see Error handling on page 15

Contents

This chapter contains the following topics:

Managed Element Registry

The Managed Element Registry service defines the following error codes and messages.

Table 2: Managed Element Registry error codes

Error code	Description
MER0001	Access to the requested functionality is denied.
MER0002	An internal error occurred. See logs for details.
MER0003	Invalid ID: {0}. The ID cannot be null.
MER0004	Invalid list of properties. The list cannot be null.
MER0005	Invalid name: {0}. The name cannot be null and must be between 1 and 32 characters long.
MER0006	Invalid description: {0}. The description cannot be null.
MER0007	Invalid type: {0}. The type must be a valid type supported by this system.

Error code	Description
MER0008	Invalid property name:{0}. Properties must be one of the valid properties for the managed element type.
MER0009	An error occurred while accessing persistent storage. See logs for details.

Avaya Communication Server 1000

The Avaya CS 1000 web service defines the following error codes and messages.

Table 3: Communication Server 1000 error codes

Error code	Description
CS1K0001	An internal error occurred see logs for more details.
CS1K0002	Invalid Signaling Server address: {0}.
CS1K0003	Invalid call server address: {0}.
CS1K0004	Invalid IP address: {0}.
CS1K0005	Invalid count value: {0}. Count must be between 1 and 65535.
CS1K0006	Invalid action value: {0}. The only valid value for action is 99.
CS1K0007	Invalid CS 1000 user name: {0}. The password for the managed element must be set to a valid value.
CS1K0008	Invalid terminal number: {0}.
CS1K0009	Invalid number of hops: {0}. Number of hops must be between 1 and 40.
CS1K0010	Invalid password: {0}. Password must be between 6 and 14 characters and only contain the characters 0-9, * or #.
CS1K0011	Invalid number of password uses: {0}. Number of uses must be more than 1.
CS1K0012	Invalid timeout value: {0}. Timeout must be greater than 0.
CS1K0013	Invalid protocol: {0}. Protocol must be "", "SIP", or "H323".
CS1K0014	Invalid vtrkShow start value: {0}. Start value must be greater than 0.
CS1K0015	Invalid vtrkShow range value: {0}. Range value must be greater than 1.
CS1K0016	Invalid overlay command: {0}.
CS1K0017	Invalid range values: {0}. Range start must be greater or equal to 0. Range end must be greater than range start and range values must be between 0 and 9999.

Error code	Description
CS1K0018	Access to execute the requested operation is denied.
CS1K0019	Invalid Secure Object ID value in HTTP request. Please ensure that the endpoint address has the correct format.
CS1K0020	Invalid channel number: {0}. Channel number must be less than 4301.
CS1K0021	Invalid file name or path specified: {0}.
CS1K0022	Invalid value specified: {0}. Must consist of all positive numbers.
CS1K0023	Invalid filter value specified: {0}.
CS1K0024	Invalid hour value specified: {0}. Hour value must be between 0-23.
CS1K0025	Invalid minute value specified: {1}. Minutes value must be between 0-59.
CS1K0026	Invalid soft client value specified: {0}.
CS1K0027	Invalid command name specified: {0}.
CS1K0028	Cannot access system of different release: {0}. Please ensure that the system accessed has the correct release information.
CS1K0029	Invalid Customer Number value: {0}. Must be between 0 and 99.
CS1K0030	Invalid SIP gateway application. Cannot be null.
CS1K0031	Invalid channel state. Cannot be null.
CS1K0032	An error occurred while processing the request: {0}.

Apart from these error codes; if an error occurs when you run the overlay command on the Call Server or Signaling Server, the relevant SCH code or error message is returned as the String value. If an error occurs when you run a Signalling Server command, a Cs1000ServiceException is thrown with an error code CS1K0032, along with an error message from the Signalling Server.

Phone Provisioning

The Phone Provisioning web service defines the following error codes and messages.

Table 4: Phone provisioning error codes

Error code	Description
BCC0001	Access denied.
BCC0002	An internal error occurred, see logs for details.
BCC0004	Invalid Terminal Number value: {0}. TN must be of the form lll s cc uu.

Error code	Description
BCC0005	Invalid Designation value: {0}. Must be between 1 and 6 characters long.
BCC0006	Invalid Customer Number value: {0}. Must be a configured customer number.
BCC0008	The list of Terminal Numbers provided is invalid: {0}.
BCC0009	Invalid Search View value: {0}.
BCC0010	Invalid DN value: {0}. DN must be one or more digits.
BCC0011	Invalid Phone object provided: {0}.
BCC0013	Invalid Phone Type value: {0}. Value must not be null.
BCC0014	Invalid Phone Zone value: {0}. Value must be 1 to 31 to 5 digits long and have a value of 0-2550-8000. Note that 000, 00100000, 00001 etc. are also valid.
BCC0015	Invalid Phone Key value: {0}. Value must not be null.
BCC0016	Invalid Key Number: {0}. Value must be greater than 0.
BCC0017	Invalid Key Feature: {0}. Value cannot be null.
BCC0018	Invalid Key Feature ID value: {0}. Value cannot be empty or null.
BCC0019	Invalid Voice Mailbox class of service value: {0}. Value must be between 0 and 127.
BCC0020	Invalid Voice Mailbox action. Value cannot be null.
BCC0021	Invalid CPND name value: First name and last name values cannot both be blank.
BCC0022	Invalid Phone Feature ID value: {0}. Value must not be null or blank.
BCC0023	Invalid Phone Feature value: {0}. Value cannot be null.
BCC0024	Invalid Phone class of service value: {0}. Values cannot be empty.
BCC0025	Invalid Prime DN key: {0}.
BCC0026	Invalid feature value used during update: {0}.
BCC0029	Invalid feature criteria value: {0}. Value cannot be null.
BCC0030	Invalid search parameters. At least one value must be non null.
BCC0032	Invalid phone input. Cannot enable prime DN key and keys together.
BCC0033	An error occurred while accessing the system information with the following message. Make sure that the element ID provided in the URL is correct.
BCC0034	An error occurred while executing the operation with the following message: {0}.

Error code	Description
BCC0035	Invalid phone feature parameter: {0}.
BCC0037	Invalid Secure Object ID value in HTTP request. Please ensure that the endpoint address has the correct format.
BCC0040	Terminal number not found: {0}.
BCC0041	Invalid key expansion module value: {0}. Value must be a valid integer.
BCC0042	Invalid single line feature ID: {0).
BCC0043	Invalid single line feature. Feature cannot be null.
BCC0044	Invalid key based accessories value: {0}. Value must be a valid integer.
BCC0045	Invalid display based accessories value: {0}. Value must be a valid integer.
BCC0046	Invalid MLNG value: {0}.
BCC0047	Invalid KBA and DBA values. At least one must be equal to 0: {0}.
BCC0048	Invalid AOM value: {0}. Value must be a valid integer.
BCC0049	Cannot access system of different release: {0}. Please ensure that the system accessed has the correct release information.
BCC0050	Invalid Status value: {0}. Value cannot be null.
BCC0051	Invalid Card Type value: {0}. Value cannot be null if TN status is UNUSED.
BCC0052	Invalid Unit Type value: {0}. Value cannot be null if TN status is UNUSED and card type is not ANALOG.
BCC0053	Invalid range values: {0}. Start and end values must both be valid and the same format, or both be null.
BCC0054	Invalid DN range: {0}. Start value must be provided, and be less than or equal to the end value.
BCC0055	Invalid Terminal Number value: {0}. Value must be one of the valid TN formats lll,lll s, lll s cc or lll s cc uu.
BCC0056	Zone is not a supported property for the given phone type.
BCC0057	Keys are not supported on the given phone type.
BCC0058	Key expansion module is not a supported property for the given phone type.
BCC0059	Key based accessories is not a supported property for the given phone type.
BCC0060	Display based accessories is not a supported property for the given phone type.
BCC0061	Add on modules is not a supported property for the given phone type.

Error code	Description
BCC0062	Single line features are not supported for the given phone type.
BCC0063	Prime DN key is not supported for the given phone type.
BCC0064	Invalid CPND language value. Cannot be null.
BCC0065	Invalid CPND name format value. Cannot be null.
BCC0066	Invalid tenant value: {0}. Value must be between 1 and 511.
BCC0067	An error occurred while initializing the system. This may be due to problems accessing the call server. See log file for details.

Important:

If the Phone Provisioning web service is not able to access the CS 1000 system, it throws a BccServiceException with the error code BCC0067. There are several reasons, but the most common cause is that all TTY ports on the call server are currently in use, or the call server is offline. For more information about the specifics of this error, see the Phone Provisioning web service log file.

NRSM

The NRSM web service defines the following error codes and messages.

Table 5: NRSM error codes

Error code	Description
NRSM1000	Collaborative Server cannot be null.
NRSM1001	Default Route cannot be null.
NRSM1002	Gateway Endpoint cannot be null.
NRSM1003	L0 Domain cannot be null.
NRSM1004	L1 Domain cannot be null.
NRSM1005	NRS Server cannot be null.
NRSM1006	Post Translator cannot be null.
NRSM1007	Route cannot be null.
NRSM1008	Service Domain cannot be null.
NRSM1009	System Wide Settings cannot be null.
NRSM1010	User Endpoint cannot be null.
NRSM1011	Authentication State cannot be null.

Error code	Description
NRSM1012	Collaborative Server Domain Type cannot be null.
NRSM1013	Database Instance cannot be null.
NRSM1014	DN Type cannot be null.
NRSM1015	H323 Support cannot be null.
NRSM1017	Redundant State cannot be null.
NRSM1019	Server Role cannot be null.
NRSM1020	Sip Support cannot be null.
NRSM1022	An internal error occurred while processing your request. See log file on server for details.
NRSM1023	Access to execute the requested operation is denied.
NRSM1024	Invalid service domain name: {0}. Value must be between 1 and 30 characters long.
NRSM1025	Invalid L1 domain name: {0}. Value must be between 1 and 30 characters long.
NRSM1026	Invalid L0 domain name: {0}. Value must be between 1 and 30 characters long.
NRSM1027	Invalid gateway endpoint name: {0}. Value must be between 1 and 30 characters long.
NRSM1028	Invalid user endpoint name: {0}. Value must be between 1 and 30 characters long.
NRSM1029	Invalid originating endpoint name: {0}. Value must be between 1 and 30 characters long.
NRSM1030	Invalid terminating endpoint name: {0}. Value must be between 1 and 30 characters long.
NRSM1031	Unable to find post translator with the specified properties.
NRSM1032	Invalid IP address: {0}.
NRSM1033	Invalid route cost value: {0}. Value must between 1 and 255.
NRSM1034	Invalid phone context: {0}. Value cannot contain newlines, apostrophes or spaces and must be less than 250 characters long.
NRSM1035	Invalid routing string value: {0}. Value must be between 1 and 24.
NRSM1036	Invalid DN prefix: {0}. Value must be up to 8 digits.
NRSM1038	Invalid digits to start value: {0}. Value must be between 1 to 24 digits.
NRSM1039	An error occurred retrieving the system wide settings values. See logs for details.
NRSM1040	Unable to find the specified service domain name: {0}.

Error code	Description
NRSM1041	Unable to find the specified L1 domain name: {0}.
NRSM1042	Unable to find the specified L0 domain name: {0}.
NRSM1043	Unable to find the specified gateway endpoint name:
NRSM1044	Unable to find the specified route.
NRSM1045	Unable to find the specified user endpoint name: {0}.
NRSM1046	Unable to find the specified collaborative server address: {0}.
NRSM1047	Invalid description value: {0}. Value must be less than 120 characters, and cannot contain newlines, carriage returns, commas, or apostrophes.
NRSM1048	Invalid password value. Value must be alphanumeric and less than 25 characters long.
NRSM1049	Invalid E.164 area code: {0}. Value must be up to 8 digits.
NRSM1050	Invalid E.164 country code: {0}. Value must be up to 8 digits.
NRSM1051	Invalid E.164 access code: {0}. Value must be up to 8 digits.
NRSM1052	Invalid E.164 access code length: {0}. Value must be between 0 and 99.
NRSM1053	Invalid special number: {0}. Value must be up to 30 digits.
NRSM1054	Invalid special number dialing code length: {0}. Value must be between 0 and 31.
NRSM1055	Invalid emergency service access prefix: {0}. Value must be up to 30 digits.
NRSM1056	Invalid special number label: {0}. Value must be alphanumeric and less than 30 characters.
NRSM1057	Invalid unqualified number label: {0}. Value must be alphanumeric and less than 30 characters.
NRSM1058	Invalid port number: {0}. Value must be between 0 and 65535.
NRSM1059	Invalid DN value: {0}. Value must be up to 30 digits.
NRSM1061	Invalid DN prefix: {0}. Value must start with a digit, consist of up to 30 digits, -, # or ? characters.
NRSM1062	Invalid number of digits to remove: {0}. Value must be between 0 and 99.
NRSM1063	Invalid digits to add: {0}. Value must be between 1 and 24 digits and can be prefixed by 1 or more '+' characters.
NRSM1064	Invalid alias name: {0}. Value must be alphanumeric and less than 31 characters long.
NRSM1065	Invalid host name: {0}. Value must be alphanumeric and less than 19 characters long. Can contain '-' or '.'.

Error code	Description
NRSM1066	Invalid control priority: {0}. Value must be between 0 and 63.
NRSM1067	Invalid realm name: {0}. Value must be alphanumeric and less than 19 characters long. Can contain '-' or '.'.
NRSM1068	Invalid H.323 LRQ response timeout: {0}. Value must be between 1 and 10.
NRSM1069	Invalid public SIP name: {0}. Value must be 0 and 96 alphanumeric characters.
NRSM1070	Invalid public SIP number: {0}. Value must be up to 8 digits. Can include '-' if not the first character.
NRSM1071	Invalid MTU value: {0}. Value must be up to 5 digits.
NRSM1072	Invalid session cache: {0}. Value must be between 1024000 and 4096000.
NRSM1073	Invalid session cache timeout: {0}. Value must be between 600 and 3600.
NRSM1074	Invalid renegotiation in byte value: {0}. Value must be between 1024000 and 32768000.
NRSM1075	Invalid NCS timeout value: {0}. Value must be between 1 and 30.
NRSM1076	Invalid registration time expiry: {0}. Value must be between 30 and 3600.
NRSM1077	Invalid auto backup time: {0}. Value must be of the format HH:MM.
NRSM1078	Invalid auto backup path: {0}. Value must be a maximum of 120 characters long, and cannot contain newlines.
NRSM1079	Invalid auto backup username: {0}. Value must be 1 to 30 characters long.
NRSM1080	Invalid NCS port number: {0}. Value must be between 1024 and 65535.NRSM1081=Cannot access system of different release: {0}. Please ensure that the system accessed has the correct release information.
NRSM1082	Invalid Secure Object ID value in HTTP request. Please ensure that the endpoint address has the correct format.
NRSM1083	Invalid SIP Mode value: {0}. Value cannot be null.
NRSM1084	Cannot change the type, service domain, l1 domain or l0 domain when updating a collaborative server.
NRSM1085	Unable to find the specified default route.
NRSM1090	An error occurred while processing the request: {0}.

Alternatively; other error codes, which are not defined by the web service can be returned directly by the underlying NRSM logic. These errors indicate complex errors such as field or feature dependencies and are prefixed with WC.

Chapter 7: Code examples

This chapter lists the entire content of the Java sample code referenced in this document.

SampleClient.java

```
package com.avaya.ucm.ws.client;
import java.text.MessageFormat;
import javax.xml.ws.BindingProvider;
import com.avaya._1_0.cs1000webservice.Cs1000Management;
import com.avaya._1_0.cs1000webservice.Cs1000Service;
import com.avaya._1_0.cs1000webservice
.Cs1000ServiceException_Exception;
import com.avaya._1_0.managedelementregistry
.ElementRegistryServiceException_Exception;
import com.avaya._1_0.managedelementregistry.ElementType;
import com.avaya._1_0.managedelementregistry
.ManagedElement;
import com.avaya._1_0.managedelementregistry
.ManagedElementRegistry;
import com.avaya._1_0.managedelementregistry
.ManagedElementService;
import com.avaya._1_0.managedelementregistry.Service;
import com.avaya._1_0.managedelementregistry.ServiceType;
public final class SampleClient
{
private static final String USERNAME= "admin";
private static final String PASSWORD = "admin12_Admin";
/**
* Constructor. Sets up keystore location and password.
*/
public SampleClient()
{
//Set the location and password for keystore.
System.setProperty("javax.net.ssl.trustStore",
"D://KeyStores//ucmKeystore" );
System.setProperty("javax.net.ssl.trustStorePassword",
"password" );
}
/**
* Executes a basic CS 1000 overlay command and prints the
result.
*/
public void executeOverlayCommand()
{
final Cs1000Management proxy =
new Cs1000Service().getCs1000ServicePort();
//Set endpoint address
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
getServiceUrl( ServiceType.CS_1000 ) );
//Set username and password
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.USERNAME_PROPERTY, USERNAME );
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.PASSWORD_PROPERTY, PASSWORD );
try
```

```
{
System.out.println( proxy.executeOverlayCommand(
"<?xml version=\"1.0\"?> +
"<LD0VL>" +"
"<LD>21</LD>" +
"<REQ>1tm</REQ>" +
"</LD0VL>" ) ) ;
}
catch (Cs1000ServiceException e)
{
final String errorCode = e.getFaultInfo()
.getErrorCode();
final String errorDescription=e.getFaultInfo()
.getErrorDescription();
final String errorValue = e.getFaultInfo()
.getVariable();
System.out.println( "Error code: " + errorCode );
System.out.println( MessageFormat.format( error
Description,
errorValue ) );
}
}
/**
*Gets the web service endpoint URL for the service for the
element.
* with given name using the managed element registry service.
*
@param serviceType the type of service to get the URL for.
* @return the URL for the requested service type for the
element.
*/
private String getServiceUrl( final ServiceType serviceType)
{
String url = null;
try
{
final ManagedElementRegistry proxy =
new ManagedElementService()
.getManagedElementServicePort();
((BindingProvider) proxy).getRequestContext().put(
BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
https://otm-hp13.ca.avaya.com/+
"managedElementRegistryService_1_0/" +
"ManagedElementRegistryWebService.secure" );
((BindingProvider) proxy).getRequestContext().put(BindingProvider.
USERNAME_PROPERTY, USERNAME );
((BindingProvider) proxy).getRequestContext().put(BindingProvider.
PASSWORD_PROPERTY, PASSWORD );
//Assume there is exactly one CS 1000 element returned.
final ManagedElement element =
(ManagedElement)proxy.getManagedElementsByType(
ElementType.CS_1000 ).get( 0 );
for (Service service : element.getServices())
{
if (service.getType() == serviceType)
{
url = service.getUrl();
}
}
}
catch (final ElementRegistryServiceException
_Exception e)
{
final String errorCode = e.getFaultInfo()
.getErrorCode();
```

```
final String errorDescription = e.getFaultInfo()
.getErrorDescription();
final String errorValue = e.getFaultInfo()
.getVariable();
System.out.println( "Error code: " + errorCode );
System.out.println( MessageFormat
.format(errorDescription,
errorValue ) );
}

return url;
}
public static void main( String[] args )
{
final SampleClient client = new SampleClient();
client.executeOverlayCommand();
}
}
```

build.xml

```
<?xml version="1.0"?>
<project name="ucmWebServiceUserGuideExample_6_0" default=
"default">
<!-- Environment specific properties -->
<property name="jaxws.lib.dir"location=
"D:/JAX-WS-2.1.4/jaxws-ri/lib"
description="Location of JAX-WS. Must be 2.1.4"/>
<property name="jdk.lib.dir" location=
"C:\jdk1.5.0_12\lib"
description="Location of JDK lib directory" />
<property name="server.name" value=
"otm-hp9.ca.avaya.com"
description="Name of UCM server" />
<property name="keystore.location"
value="D://KeyStores//ucmKeystore"
description="Location of keystore" />
<property name="keystore.password"
value="password"
description="Password for keystore" />
<property name="mer.wsdl.file"
location="D"/wsdls/ManagedElementRegistryWebService
_6_0.wsdl"/>
<property name="cs1000.wsdl.file"
location="D:/wsdls/Cs1000WebService_6_0.wsdl"/>
<property name="bcc.wsdl.file"
location="D:/wsdls/BssWebService_6_0.wsdl"/>
<!-- Paths local to the project -->
<property name="src.dir" location="src"
description="Source code directory." />
<property name="classes.dir" location="build/classes"
description="Directory where compiled files go." />
<property name="generated.dir" location="generated"
description="Directory where generated files go." />
<!--
This path contains all JAR files required for running
wsimport
and compiling the source.
-->
<path id="jaxws.classpath">
<fileset dir="${jaxws.lib.dir}">
<include name="**/*.jar" />
</fileset>
<fileset dir="${jdk.lib.dir}">
```

```
<include name="**/*.jar" />
</fileset>
</path>
<!-- Define wsimport task -->
<taskdef name="WsImport" classname=
"com.sun.tools.ws.ant.WsImport">
<classpath refid="jaxws.classpath" />
</taskdef>
<!-- Define targets -->
<target name="default" depends="clean, init,
buildClientStubs, compile">
</target>
<target name="init>
<mkdir dir="${generated.dir}" />
mkdir dir="${classes.dir}" /
</target>
<target name="clean">
<delete dir="${generated.dir}" />
<delete dir="${classes.dir}" />
</target>
<!--
Builds all the required client side artifacts for
the element registry,
CS 1000 and BCC web services.
-->
<target name="buildClientStubs" depends="init">
<wsimport keep="true" sourcedestdir="${generated.dir}"
destdir="${classes.dir}" extension=
"true" verbose="true"
fork="true" wsdl="${mer.wsdl.file}">
</wsimport>
<wsimport keep="true" sourcedestdir="$(generated.dir)"
destdir="#{classes.dir}" extensions=
"true" verbose="true"
fork="true" wsdl="${cs1000.wsdl.file}">
</wsimport>
<wsimport keep="true" sourcedestdir="$(generated.dir)"
destdir="${classes.dir}" extensions=
"true" verbose="true"
fork="true" wsdl="${bcc.wsdl.file}">
</wsimport>
</target>
<target name="compile">
<javac srcdir="${src.dir}" destdir="${classes.dir}"
classpath="${generated.dir}"
<classpath refid="jaxWS.classpath" />
</javac>
</target>
</project>
```

Chapter 8: Web service API documentation

Contents

This section contains the following topics:

MER Web Service API

The MER web service API defines methods to search for Managed Elements on the Avaya Unified Communication Management (Avaya UCM) CS framework. You cannot add, update, or delete the elements as this is a read-only service. Only elements that support the Avaya Communication Server 1000 (Avaya CS 1000), Phone Provisioning or NRS web service are returned.

The managed element objects returned from this call contain all information needed to retrieve the WSDL for the supported services and the URL required to invoke those services.

All methods in this interface can generate a ElementRegistryServiceException indicating an error. For more information about how to handle this exception, and the possible error codes that it can contain, see the user guide.

Table 6: MER web service API methods

Method summary	
`java.util.List<ManagedElement>`	`getAllManagedElements()` Obtain a list of all Managed Elements currently configured on the system.
`ManagedElement`	`getManagedElementById` `(java.lang.String id)`

Method summary	
	Obtains the Managed Element with the specified ID.
`java.util.List<ManagedElement>`	`getManagedElementsByName (java.lang.String name` Obtains a list of Managed Elements with the specified name.
`java.util.List<ManagedElement>`	`getManagedElementsByType (ElementType type)` Obtains an list of Managed Elements with the specified type.

Avaya CS 1000 Web Service API

You can use the CS 1000 web service API methods to execute commands on a CS 1000 system. This includes general overlay commands on the Call Server and specific OAM CLI commands on the Signaling Server. Valid values for parameters are determined by restrictions enforced by the CS 1000.

Not all OAM CLI commands provided by this interface are supported on all hardware types. If a command is unsupported, an exception indicating that the command is unknown is thrown. Refer to the hardware documentation to determine which commands are supported.

All methods in this interface can generate a CS 1000 Service Exception indicating an error. For more information about how to handle this exception, and the possible error codes that it can contain, see the user guide.

Table 7: CS 1000 web service API methods

Method summary	
`java.lang.String`	`balanceIpSetRegistrationLoad (java.lang.String ipAddress)` Runs a loadBalance command on the Signaling Server or Media Card.
`java.lang.String`	`clearNodeTempPassword (java.lang.String ipAddress)` Runs an nodeTempPwdClear command on the Signaling Server or Media Card.

Method summary	

`void`	`deleteCallersList`
	`(java.lang.String ipAddress, int customerNum, java.lang.String directoryNum)`
	Runs a deletePDRLCL command on the server to delete callers list entries.

`void`	`deletePersonalDirectory`
	`java.lang.String ipAddress, int customerNum, java.lang.String directoryNum)`
	Runs a deletePDRLCL command on the server to delete personal directory entries.

`void`	`deleteRedialList`
	`java.lang.String ipAddress, int customerNum, java.lang.String directoryNum)`
	Runs a deletePDRLCL command on the server to delete redial list entries.

`void`	`deleteUserPreferences`
	`java.lang.String ipAddress, int customerNum, java.lang.String directoryNum)`
	Runs a deletePDRLCL command on the server to delete user preference entries.

`java.lang.String`	`disableFirmwareDownloadTurboMode`
	`(java.lang.String ipAddress, int delayInMinutes)`
	Runs an uftpTurboMode command on the Signaling Server to disable firmware download turbo mode on the Signaling Server immediately or after the specified duration.

`java.lang.String`	`disableNodePassword`
	`(java.lang.String ipAddress)`
	Runs an nodePwdDisable command on the Signaling Server or Media Card.

`java.lang.String`	`disableServicesForcefully`
	`java.lang.String ipAddress)`
	Execute a forceDisServices command on the Signaling Server or Media Card.

Method summary	
`java.lang.String`	`disableServicesGracefully (java.lang.String ipAddress)` Runs disServices command on the Signaling Server or Media Card.
`java.lang.String`	`disableSipCtiTrace (java.lang.String ipAddress)` Runs SIPCTITrace command on the Signaling Server to disable trace.
`java.lang.String`	`disableSipCtiTraceLevelOutput (java.lang.String ipAddress)` Runs SIPCTITraceLevel command on the Signaling Server to disable the trace level output.
`java.lang.String`	`disableTpsForcefully (java.lang.String ipAddress)` Runs a forceDisTps command on the Signaling Server or Media Card.
`java.lang.String`	`disableTpsGracefully (java.lang.String ipAddress)` Runs a disTps command on the Signaling Server or Media Card.
`java.lang.String`	`disableVirtualTrunksForcefully java.lang.String ipAddress)` Runs a forcedVTRK command on the Signaling Server.
`java.lang.String`	`disableVirtualTrunksGracefully (java.lang.String ipAddress)` Runs a disVtrk command on the Signaling Server.
`java.lang.String`	`enableFirmwareDownloadTurboMode (java.lang.String ipAddress, int delayInMinutes)` Runs an uftpTurboMode command on the Signaling Server to enable firmware download turbo mode on

Method summary	
	the Signaling Server immediately or after the specified duration.

`java.lang.String`	`enableNodePassword`
	`(java.lang.String ipAddress)`
	Runs an nodePwdEnable command on the Signaling Server or Media Card.

`java.lang.String`	`enableServices`
	`(java.lang.String ipAddress)`
	Runs an enlServices command on the Signaling Server or Media Card.

`java.lang.String`	`enableSipCtiTrace`
	`(java.lang.String ipAddress)`
	Runs SIPCTITrace command on the Signaling Server to enable SIPCTI trace.

`java.lang.String`	`enableSipCtiTraceForIncomingMessages`
	`(java.lang.String ipAddress)`
	Runs SIPCTITrace command on the Signaling Server to enable SIPCTI trace for incoming messages.

`java.lang.String`	`enableSipCtiTraceForOutgoingMessages`
	`(java.lang.String ipAddress)`
	Runs SIPCTITrace command on the Signaling Server to enable SIPCTI trace for outgoing messages.

`java.lang.String`	`enableSipCtiTraceLevelOutput`
	`(java.lang.String ipAddress)`
	Runs SIPCTITraceLevel command on the Signaling Server to enable the trace level output.

`java.lang.String`	`enableTps`
	`(java.lang.String ipAddress)`
	Runs an enlTps command on the Signaling Server or Media Card.

Method summary	

`java.lang.String`	`enableVirtualTrunks`
	`(java.lang.String ipAddress)`
	Runs an enlVTRK command on the Signaling Server

`java.lang.String`	`executeOverlayCommand`
	`(java.lang.String overlayCommand)`
	Runs an overlay command on the Call Server.

`java.lang.String`	`executeRemotePingUsingIp`
	`(java.lang.String ipAddress, java.lang.String setIpAddress, java.lang.String remoteIpAddress, int count)`
	Runs a rping command on the Signaling Server using a phones IP address.

`java.lang.String`	`executeRemotePingUsingTn`
	`(java.lang.String ipAddress, java.lang.String setTerminalNumber, java.lang.String remoteIpAddress, int count)`
	Runs a rping command on the Signaling Server using a phones Terminal Number.

`java.lang.String`	`executeRemoteTraceRouteUsingIp`
	`(java.lang.String ipAddress, java.lang.String setIpAddress, java.lang.String remoteIpAddress, int maxNumHops)`
	Runs an rTraceRoute command on the Signaling Server using a phones IP address.

`java.lang.String`	`executeRemoteTraceRouteUsingTn`
	`java.lang.String ipAddress, java.lang.String setTerminalNumber, java.lang.String remoteIpAddress, int maxNumHops)`
	Runs an rTraceRoute command on the Signaling Server using a phones terminal number.

`java.lang.String`	`getAllChannelInfo` `(java.lang.String ipAddress)`

Method summary	
	Runs vgwShowAll command only on the Media Card.

`java.lang.String`	`getAllVirtualTrunkInfo`
	`(java.lang.String ipAddress, int start, int range)`
	Runs a vtrkShow command on the Signaling Server with no parameters.

`java.lang.String`	`getCardRoleInfo`
	`(java.lang.String ipAddress)`
	Runs cardRoleShow command on the Signaling Server.

`java.lang.String`	`getDchannelStatus`
	`java.lang.String ipAddress, java.lang.String channelNum)`
	Runs DCHStatus command on the Signaling Server.

`java.lang.String`	`getDspSoftwareVersion`
	`(java.lang.String ipAddress)`
	Runs a dspSWVersionShow command on the Signaling Server or Media Card.

`java.lang.String`	`getEchoServerInfo`
	`(java.lang.String ipAddress)`
	Runs an echoServerShow command on the Signaling Server with no parameters.

`java.lang.String`	`getEchoServerInfoAndResetCount`
	`(java.lang.String ipAddress)`
	Runs an echoServerShow command on the Signaling Server.

`java.lang.String`	`getElectionInfo`
	`(java.lang.String ipAddress)`
	Runs an electShow command on the Signaling Server.

Method summary	

`java.lang.String`	`getFirmwareDownloadTurboModeInfo`
	`(java.lang.String ipAddress)`
	Runs an uftpTurboModeShow command on the Signaling Server.

`java.lang.String`	`getFirmwareInfoForAllSets`
	`(java.lang.String ipAddress)`
	Runs an isetFWShow command on the Signaling Server or Media Card.

`java.lang.String`	`getFirmwareInfoForSelectedSets`
	`(java.lang.String ipAddress, java.lang.String query)`
	Runs an isetFWGet command on the Signaling Server or Media Card for selected phones.

`java.lang.String`	`getHelp`
	`(java.lang.String ipAddress, java.lang.String command)`
	Runs a Help command on the Signaling Server.

`java.lang.String`	`getHostTableInfo`
	`(java.lang.String ipAddress)`
	Runs a hosts command on the Signaling Server.

`java.lang.String`	`getIpInfo`
	`(java.lang.String ipAddress)`
	Runs an ipInfoShow command on the Signaling Server or Media Card.

`java.lang.String`	`getIpSetInfo`
	`(java.lang.String ipAddress, int rangeStart, int rangeEnd)`
	Runs an isetShow command on the Signaling Server or Media Card.

`java.lang.String`	`getIpSetInfoUsingIp` `(java.lang.String ipAddress, java.lang.String setIpAddress)`

Method summary	
	Runs an isetInfoShow command on the Signaling Server or Media Card using a phones IP address.

`java.lang.String`	`getIpSetInfoUsingTn`
	`(java.lang.String ipAddress, java.lang.String setTerminalNumber)`
	Runs an isetInfoShow command on the Signaling Server or Media Card using a phones terminal number.

`java.lang.String`	`getIpSetNatInfo`
	`(java.lang.String ipAddress, int rangeStart, int rangeEnd)`
	Runs an isetNATShow command on the Signaling Server or Media Card.

`java.lang.String`	`getIpStatus`
	`(java.lang.String ipAddress)`
	Runs an netstat command on the Signaling Server.

`java.lang.String`	`getItgCardInfo`
	`(java.lang.String ipAddress)`
	Runs an itgCardShow command on the Signaling Server or Media Card.

`java.lang.String`	`getLocationInfoForSets`
	`(java.lang.String ipAddress, int rangeStart, int rangeEnd)`
	Runs an isetLocShow command on the Signaling Server or Media Card.

`java.lang.String`	`getLocationInfoForSetsThatNeedsUpdate`
	`(java.lang.String ipAddress, int rangeStart, int rangeEnd)`
	Runs an isetLocNeedUpdateShow command on the Signaling Server or Media Card.

`java.lang.String`	`getNodePasswordSettings`
	`(java.lang.String ipAddress)`
	Runs an nodePwdShow command on the Signaling Server or Media Card.

Method summary	

`java.lang.String`	`getNumDsps`
	`(java.lang.String ipAddress, com.avaya.esm.mgmt.CS 1000.webservice.ChannelState channelState)`
	Runs a DSPNumShow command on the Signaling Server or Media Card.

`java.lang.String`	`getPbxLinkStatus`
	`(java.lang.String ipAddress)`
	Runs a pbxLinkShow command on the Signaling Server or Media Card.

`java.lang.String`	`getRoutingInfo`
	`(java.lang.String ipAddress)`
	Runs a route command on the Signaling Server or Media Card.

`java.lang.String`	`getRtcpStatusInfoUsingIp`
	`(java.lang.String ipAddress, java.lang.String endpointIpAddress)`
	Runs a RTPStatShow command on the Signaling Server using IP address.

`java.lang.String`	`getRtcpStatusInfoUsingTn`
	`(java.lang.String ipAddress, java.lang.String endpointTn)`
	Runs a RTPStatShow command on the Signaling Server using terminal number.

`java.lang.String`	`getRudpInfo`
	`(java.lang.String ipAddress)`
	Runs a rudpShow command on the Signaling Server.

`java.lang.String`	`getServicesStatus`
	`(java.lang.String ipAddress)`
	Runs a servicesStatusShow command on the Signaling Server or Media Card.

Method summary	

`java.lang.String`	`getSipCtiTraceInfo`
	`(java.lang.String ipAddress)`
	Runs a SIPCTITraceShow command on the Signaling Server.

`java.lang.String`	`getSipGatewayCallingNumInfo`
	`java.lang.String ipAddress, com.avaya.esm.mgmt.CS 1000.webservice.GwApplication appName, java.lang.String callingOrCalledNum, java.lang.String numberingPlanIndicator, java.lang.String numType)`
	Runs a SIPGwShownum command on the Signaling Server with a numbering plan indicator.

`java.lang.String`	`getSipGatewayChannelInfo`
	`(java.lang.String ipAddress, com.avaya.esm.mgmt.CS 1000.webservice.GwApplication appName, java.lang.String channelNum)`
	Runs SIPGwShowch command on the Signaling Server.

`java.lang.String`	`getSipGatewayInfo`
	`(java.lang.String ipAddress,com.avaya.esm.mgmt.CS 1000.webservice.GwApplication appName)`
	Runs a SIPGwShow command on the Signaling Server.

`java.lang.String`	`getSipGatewayNumInfo`
	`(java.lang.String ipAddress, com.avaya.esm.mgmt.CS 1000.webservice.GwApplication appName, java.lang.String callingOrCalledNum)`
	Runs a SIPGwShownum command on the Signaling Server.

`java.lang.String`	`getSystemInfo`
	`(java.lang.String ipAddress)`
	Runs an ifConfig command on the Signaling Server or Media Card.

Method summary	

`java.lang.String`	`getSystemResourceInfo`
	`(java.lang.String ipAddress)`
	Runs a sysResShow command on the Signaling Server.

`java.lang.String`	`getTaskInfo`
	`(java.lang.String ipAddress)`
	Runs a ps command on the Signaling Server.

`java.lang.String`	`getTotalNumSipCtiSessions`
	`(java.lang.String ipAddress)`
	Runs a SIPCTISessionShow command on the Signaling Server.

`java.lang.String`	`getUpgradePolicyInfo`
	`(java.lang.String ipAddress)`
	Runs a umsPolicyShow command on the Signaling Server.

`java.lang.String`	`getVirtualTrunkInfoForProtocol`
	`(java.lang. String ipAddress, java.lang.String protocol, int start, int range)`
	Runs a vrtShow command on the Signaling Server with the specified parameters.

`java.lang.String`	`getVirtualTrunksNetworkMonitorInfo`
	`(java.lang.String ipAddress)`
	Runs an vtrkNetMonShow command on the Signaling Server.

`java.lang.String`	`redirectSipCtiTraceOutputToFile`
	`(java.lang.String ipAddress, java.lang.String fileName)`
	Runs a SIPCTIOutput command on the Signaling Server to redirect the SIPCTI trace output to specific file.

`java.lang.String`	`redirectSipCtiTraceOutputToRtplog` `(java.lang.String ipAddress)`

Method summary	
	Runs a SIPCTIOutput command on the Signaling Server to redirect the SIPCTI trace output log to rtplog.

`java.lang.String`	`redirectSipCtiTraceOutputToTty`
	`(java.lang.String ipAddress)`
	Runs a SIPCTIOutput command on the Signaling Server to redirect the SIPCTI trace output to tty.

`java.lang.String`	`renoveAllSipCtiSessions`
	`(java.lang.String ipAddress)`
	Run a SIPCTIStop command on the Signaling Server to remove all DN and remove the SIPCTI sessions.

`java.lang.String`	`removeSingleSipCtiSession`
	`(java.lang.String ipAddress, java.lang.String directoryNum)`
	Run a SIPCTIStop command on the Signaling Server to remove one DN and remove the SIPCTI session.

`java.lang.String`	`setFirmwareDownLoadTurboModeIdleTimeOut`
	`(java.lang.String ipAddress, int timeoutInMinutes)`
	Runs an uftp TurboModeTimeoutSet command on the Signaling Server.

`java.lang.String`	`setFirmwareDownLoadTurboModeStartTime`
	`(java.lang.String ipAddress, int hour, int minutes, int durationMinutes)`
	Runs an uftp TurboMode command on the Signaling Server to schedule firmware download turbo mode start time and duration on the Signaling Servers in the node.

`java.lang.String`	`setNodePassword`
	`java.lang.String ipAddress, java.lang.String password`
	Runs an nodePwdSet command on the Signaling Server or Media Card.

Method summary	

`java.lang.String`	`setNodeTempPassword`
	`(java.lang.String ipAddress, java.lang.String password, int numUses, int timeoutInHours)`
	Runs an nodeTempPwdSet command on the Signaling Server or Media Card.

`java.lang.String`	`setSipCtiTraceFilter`
	`(java.lang.String ipAddress, java.lang.String filter)`
	Runs a SIPCTITrace command on the Signaling Server to configure the filter for TR87 body.

`java.lang.String`	`setSipCtiTraceFilerUsingDn`
	`(java.lang.String ipAddress, java.lang.String directoryNum)`
	Runs a SIPCTITrace command on the Signaling Server to configure the directory number as the filter for SIPCTI trace.

`java.lang.String`	`setSipTraceFooftClientUri`
	`(java.lang.String ipAddress, java.lang.String softClientUri)`
	Runs a SIPCTITrace command on the Signaling Serer to configure the URI for the soft client.

`java.lang.String`	`setUserResponseForFrimwareUpgrade`
	`(java.lang.String ipAddress, int timeoutInMinutes)`
	Runs an uftp Auto Upgrade TimeoutSet command on the Signaling Server.

`java.lang.String`	`startFirmwareDownloadTurboMode`
	`(java.lang.String ipAddress, int delayInMinutes)`
	Runs uftp TurboMode command on the Signaling Server to start firmware download turbo mode on all Signaling Servers in the node immediately or after the specified start duration.

Method summary	

`java.lang.String`	`stopFrimwareDownloadTurboMode`
	`(java.lang.String ipAddress, int delayInMinutes)`
	Runs uftp TurboMode command on the Signaling Server to stop firmware download turbo mode on all Signaling Servers in the node immediately or after the specified stop duration.

Phone Provisioning Web Service API

You can use the Phone Provisioning web service API methods to manage telephones on the CS 1000. These methods provide phone functions such as search, add, move, delete, and modify. Valid values for parameters are determined by restrictions enforced by the Phone Provisioning application.

All methods in this interface can generate a BccServiceException indicating an error. For more information about how to handle this exception, and the possible error codes that it can contain, see the user guide.

After you install a new system, you can access the Phone Provisioning service methods after initializing Phone Provisioning database. You can initialize the Phone Provisioning database either by starting Phone Provisioning Web application (clicking the Phones link in Element Manager) or by calling the initializeSystem() web service method. After a new installation, if you call the method before the Phone Provisioning database initializes, then BccServiceException generates an error code.

Table 8: Phone provisioning web service API methods

Method Summary	

`void`	`addPhone`
	`(Phone phone)`
	Adds the specified Phone to the system.

`void`	`deletePhone`
	`(java.lang.String terminalNum`
	Deletes the Phone with the terminal number.

`java.util.List<java.lang.Integer>`	`getAllCustomerNumbers()` Obtains all the customer numbers present in the current system

Method Summary	

`java.util.List<java.lang.String>`	`getDirectoryNumbers`
	`(Status status, int customer, java.lang.String dnRangeStart, java.lang.String dnRangeEnd)`
	Obtains a list of directory numbers based on the input parameters.

`Phone`	`getPhone`
	`(java.lang.String terminalNum)`
	Obtains the Phone with the given terminal number.

`java.util.List<PhoneSummary>`	`getPhoneSummaries`
	`(java.lang.Integer customer, java.lang.String primeDn, java.lang.String terminalNum, PhoneType phoneType, java.lang.String designation)`
	Performs a phone search with the given search criteria.

`java.util.List<java.lang.String>`	`getTerminalNumbers`
	`Status status, CardType cardType, UnitType unitType, java.lang.String tnRangeStart, java.lang.String tnRangeEnd)`
	Obtains a list of terminal numbers based on the input parameters.

`void`	`initializeSystem()` `Retrieve system information and stores it in the local database.`

`void`	`movePhone`
	`(java.lang.String oldTerminalNum, java.lang.String newTerminalNum)`
	Moves a phone from the old terminal number to the new terminal number.

`void`	`retrieveAndReconcilePhones()` Retrieves all phones from the call server to ensure that the database is in sync.

Method Summary	

`void`	`retrieveSpecificPhones`
	`(int customer, java.lang.String terminalNum, PhoneType phoneType, java.lang.String designator, CardDensity cardDensity, java.lang.Integer tenant, java.util.Calendar modifiedSince)`
	Retrieves the specified phones as determined by the parameters to ensure that the database is in sync.

`void`	`swapPhones`
	`(java.lang.String terminalNum1, java.lang.String terminalNum2)`
	Swaps the phones with the given TNs.

`void`	`updatePhone`
	`(Phone phone)`
	Updates the specified Phone.

NRSM Web Service API

The NRSM Web Service API defines the web service methods for the NRS Management functionality to update and retrieve data necessary to maintain NRS system. All entries in the NRS system are uniquely identified by using a hierarchy of components combined with a unique value such as a name value within the hierarchy. Therefore, many of the methods require you to specify a parent name identification of an element.

Perform the following tasks, to use this interface:

- Add a new entry to the NRS database. These operations take a complex type which includes all information required to add the entry.
- Update an existing entry in the database. These operations take a complex type which includes all the updated values for the entry. Also, you can use a number of additional parameters to identify the updated entry. For example, if you are updating an L1 domain, you must provide a valid service domain, and L1 domain name to identify exactly what you intend to update.
- Delete an existing entry in the database. These operations take a hierarchy of names that identify the element to delete.
- Use get to retrieve all the given type, or some subsets. You can narrow the resulting set, based on the amount of detail you provide. All get methods return a zero length array, if

no results are available. Get methods that return a single object return null if the requested object is not found.

- Perform maintenance operations that generally do not require parameters nor return a value.

You must perform all add, update and delete operations on the standby database. Perform get methods on the database as specified by the DatabaseInstance parameter.

All parameters for all methods must be valid values or an exception is thrown. All method parameters are mandatory and must be non-null, unless stated otherwise. Valid values for parameters are determined by restrictions enforced by the NRSM application. For non mandatory property, if it is null due to an empty string or the value starts or ends with an extra space (for example, " ", " abc" or "abc "), then the extra spaces are removed automatically and are saved to the database.

All methods in this interface can generate a NrsmServiceException indicating an error. For more information about how to handle this exception, and the possible error codes that it can contain, see the user guide.

Table 9: NRSM web service API methods

Method summary	

`void`	`addCollaborativeServer`
	`(CollaborativeServer collaborativeServer)`
	Adds a Collaborative Server to the system.

`void`	`addDefaultRoute`
	`(DefaultRoute defaultRoute)`
	Adds a Default Route to the system.

`void`	`addGatewayEndpoint`
	`(GatewayEndpoint gatewayEndpoint)`
	Adds a Gateway Endpoint to the system.

`void`	`addL0Domain`
	`(L0Domain l0Domain)`
	Adds a L0 domain to the system.

`void`	`addL1Domain`
	`(L1Domain l1Domain)`
	Adds a L1 domain to the system.

Method summary	

`void`	`addPostTranslator`
	`(PostTranslator postTranslator)`
	Adds a Post Translator to the system.

`void`	`addRoute`
	`(Route route)`
	Adds a Routing Entry to the system.

`addServiceDomain(ServiceDomain serviceDomain)`	`addServiceDomain`
	`(ServiceDomain serviceDomain)`
	Adds a service domain to the system.

`void`	`addUserEndpoint`
	`(UserEndpoint userEndpoint)`
	Adds a User Endpoint to the system.

`void`	`commit()` Performs the commit on database.
`void`	`cutover()` Performs a cutover on the database.

`void`	`deleteCollaborativeServer`
	`(java.lang.String collaborativeServerAddress)`
	Deletes the specified Collaborative Server from the system.

`void`	`deleteDefaultRoute`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName, DnType dnType)`
	Deletes the specified Default Route from the system.

`void`	`deleteGatewayEndpoint` `(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String endpointName)`

Method summary	
	Deletes the specified Gateway Endpoint from the system.

`void`	`deleteL0Domain`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Deletes the specified L0 domain from the system.

`void`	`deleteL1Domain`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Deletes the specified L1 domain from the system.

`void`	`deletePostTranslator`
	`(java.lang.String serviceDomainName, java.lang.String originatingEndpointName, java.lang.String terminatingEndpointName, java.lang.String targetPhoneContext, int routingStringLength, java.lang.String digitToStart)`
	Deletes the specified Post Translator from the system.

`void`	`deleteRoute`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName, java.lang.String dnPrefix, DnType dnType)`
	Deletes the specified Route from the system.

`void`	`deleteServiceDomain`
	`(java.lang.String serviceDomainName)`
	Deletes a service domain from the system.

Method summary	

`void`	`deleteUserEndpoint`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String userEndpointName`
	Deletes the specified User Endpoint from the system.

`void`	`disableGatekeeper()` Disables the Gatekeeper.
`void`	`disableNcs()` Disables the Network Connection Server.
`void`	`disableSipProxyServer()` Disables the SIP Proxy Server.
`void`	`enableGatekeeper()` Enables the Gatekeeper.
`void`	`enableNcs()` Enables the Network Connection Server.
`void`	`enableSipProxyServer()` Enables the SIP Proxy Server.

`ActiveGatewayEndpoint`	`getActiveGatewayEndpoint`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName)`
	Returns an Active Gateway Endpoint with the specified Gateway Endpoint name from the Active database.

`ActiveGatewayEndpoint[]`	`getActiveGatewayEndpoints`
	`InL0Domain(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Returns all configured active gateway endpoints configured on the Active database in the specified L0 domain.

Method summary	

`ActiveGatewayEndpoint[]`	`getActiveGatewayEndpointsInL1Domain`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns all configured active gateway endpoints configured on the Active database in the specified L1 domain.

`ActiveGatewayEndpoint[]`	`getActiveGatewayEndpointsInServiceDomain`
	`(java.lang.String serviceDomainName)`
	Returns all configured Active Gateway Endpoints configured on the Active database in the specified Service Domain.

`ActiveUserEndpoint`	`getActiveUserEndpoint`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String userEndpointName)`
	Returns the Active User Endpoint with the specified name from the active database.

`ActiveUserEndpoint[]`	`getActiveUserEndpointsInL0Domain`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Returns all configured active user endpoints configured on the active database for the specified L0 domain.

`ActiveUserEndpoint[]`	`getActiveUserEndpointsInL1Domain`
	`(java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns all configured active user endpoints configured on the active database for the specified L1 domain.

Method summary	

`ActiveUserEndpoint[]`	`getActiveUserEndpointsInServiceDomain`
	`(java.lang.String serviceDomainName)`
	Returns all configured active user endpoints configured on the active database for the specified service domain.

`ActiveGatewayEndpoint[]`	`getAllActiveGatewayEndpoints()` `getAllActiveGatewayEndpoints()` Returns all configured Active Gateway Endpoints configured on the Active database for the entire system.

`CollaborativeServer[]`	`getAllCollaborativeServers`
	`(DatabaseInstance databaseInstance)`
	Returns all configured collaborative servers configured on the specified database for the entire system.

`DefaultRoute[]`	`getAllDefaultRoutes`
	`(DatabaseInstance databaseInstance)`
	Returns all configured default routes configured on the specified database for the entire system.

`GatewayEndpoint[]`	`getAllGatewayEndpoints` `(DatabaseInstance databaseInstance)`

`L0Domain[]`	`getAllL0Domains`
	`(DatabaseInstance databaseInstance)`
	Returns all configured L0 domains configured on the specified database.

`L1Domain[]`	`getAllL1Domains`
	`(DatabaseInstance databaseInstance)`
	Returns all configured L1 domains configured on the specified database.

Method summary	

`PostTranslator[]`	`getAllPostTranslators`
	`(DatabaseInstance databaseInstance)`
	Returns all configured post translators configured on the specified database.

`Route[]`	`getAllRoutes`
	`(DatabaseInstance databaseInstance)`
	Returns all configured routes configured on the specified database.

`ServiceDomain[]`	`getAllServiceDomains`
	`(DatabaseInstance databaseInstance)`
	Returns all the configured service domains from the specified database.

`UserEndpoint[]`	`getAllUserEndpoints`
	`(DatabaseInstance databaseInstance)`
	Returns all configured user endpoints configured on the active database.

`CollaborativeServer`	`getCollaborativeServer`
	`(DatabaseInstance databaseInstance, java.lang.String collaborativeServerAddress)`
	Returns a Collaborative Server with the specified IP from the database.

`CollaborativeServer[]`	`getCollaborativeServersInL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Returns all configured collaborative servers configured on the specified database in the L0 domain.

`CollaborativeServer[]`	`getCollaborativeServersInL1Domain` `(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`

Method summary	
	Returns all configured collaborative servers configured on the specified database in the L1 domain.

`CollaborativeServer[]`	`getCollaborativeServersInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns all configured collaborative servers configured on the specified database in the service domain.

`DatabaseStatus`	`getDatabaseStatus` Gets the current database status.

`DefaultRoute`	`getDefaultRoute`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName, DnType dnType`
	Returns the Default Route with the specified ID from the database.

`DefaultRoute[]`	`getDefaultRoutesInGatewayEndpoint`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName)`
	Returns all configured default routes configured on the specified database in the gateway endpoint.

`DefaultRoute[]`	`getDefaultRoutesInL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Returns all configured default routes configured on the specified database in the L0 domain.

Method summary	

`DefaultRoute[]`	`getDefaultRoutesInL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns all configured default routes configured on the specified database in the L1 domain.

`DefaultRoute[]`	`getDefaultRoutesInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns all configured default routes configured on the specified database in the service domain.

`ServerStatus`	`getGatekeeperStatus()` Determines if the Gatekeeper is enabled on this server.

`GatewayEndpoint[]`	`getGatewayEndpointsInL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String endpointName)`
	Returns all configured gateway endpoints configured on the specified database in the L1 domain.

`GatewayEndpoint[]`	`getGatewayEndpointsInL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String endpointName)`
	Returns all configured gateway endpoints configured on the specified database in the L1 domain.

`GatewayEndpoint[]`	`getGatewayEndpointsInServiceDomain` `(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String endpointName)`

Method summary	
	Returns all configured gateway endpoints configured on the specified database in the service domain.

`L0Domain`	`getL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Returns the L0 domain with the specified ID from the specified database.

`L0Domain[]`	`getL0DomainsInL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns all configured L0 domains configured on the specified database in the service domain.

`L0Domain[]`	`getL0DomainsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns all configured L0 domains configured on the specified database in the service domain.

`L1Domain`	`getL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns the L1 Domain with the specified name and parent service domain from the database.

`L1Domain[]`	`getL1DomainsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns all configured L1 domains configured on the specified database in the service domain.

`ServerStatus`	`getNcsStatus()` Determines if the NCS is enabled on this server.

Method summary	
`NrsServer`	`getNrsServer()` Returns the NRS Server configuration information from the active database.

`int`	`getNumCollaborativeServersInL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainId)`
	Returns the total number of collaborative servers configured on the specified database in the L0 domain.

`int`	`getNumCollaborativeServersInL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns the total number of collaborative servers configured on the specified database in the L1 domain.

`int`	`getNumCollaborativeServersInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the total number of collaborative servers configured on the specified database in the service domain.

`int`	`getNumDefaultRoutesInGatewayEndpoint`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName)`
	Returns the total number of default routes configured on the specified database in the gateway endpoint.

`int`	`getNumDefaultRoutesInL0Domain` `(DatabaseInstance databaseInstance,`

Method summary	
	`java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)` Returns the total number of default routes configured on the specified database in the L0 domain.
`int`	`getNumDefaultRoutesInL1Domain (DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)` Returns the total number of default routes configured on the specified database in the specified L1 domain.
`int`	`getNumDefaultRoutesInServiceDomain (DatabaseInstance databaseInstance, java.lang.String serviceDomainName` Returns the total number of default routes configured on the specified database in the service domain.
`int`	`getNumGatewayEndpointsInL0Domain (DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)` Returns the total number of gateway endpoints configured on the specified database in the L0 domain.
`int`	`getNumGatewayEndpointsInL1Domain (DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)` Returns the total number of gateway endpoints configured on the specified database in the L1 domain.

Method summary	

`int`	`getNumGatewayEndpointsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the total number of gateway endpoints configured on the specified database in the service domain.

`int`	`getNumL0DomainsInL1Domain`
	`(DatabaseInstance databaseInstance,java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns the total number of L0 domains configured on the specified database in the specifiedL1 domain.

`int`	`getNumL0DomainsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the total number of L0 domains configured on the specified database in the service domain.

`int`	`getNumL1DomainsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the total number of L1 domains configured on the specified database in the service domain.

`int`	`getNumPostTranslatorsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the total number of post translators configured on the specified database in the service domain.

`int`	`getNumRoutesInGatewayEndpoint` `(DatabaseInstance databaseInstance,`

Method summary	
	`java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName)` Returns the total number of routes configured on the specified database in the gateway endpoint.

`int`	`getNumRoutesInL0Domain`
	
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	
	Returns the total number of routes configured on the specified database in the L0 domain.

`int`	`getNumRoutesInL1Domain`
	
	`(DatabaseInstance databaseInstance java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	
	Returns the total number of routes configured on the specified database in the L1 domain.

`int`	`getNumRoutesInServiceDomain`
	
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	
	Returns the total number of routes configured on the specified database in the service domain.

`int`	`getNumUserEndpointsInL0Domain`
	
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName`
	
	Returns the total number of user endpoints configured on the specified database in the L0 domain.

`int`	`getNumUserEndpointsInL1Domain`
	
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`

Method summary	
	Returns the total number of user endpoints configured on the specified database in the L1 domain.

`int`	`getNumUserEndpointsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the total number of user endpoints configured on the specified database in the service domain.

`PostTranslator`	`getPostTranslator`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String originatingEndpointName, java.lang.String terminatingEndpointName, java.lang.String targetPhoneContext, int routingStringLength, java.lang.String digitToStart)`
	Returns the Post Translator with the specified properties from the database.

`PostTranslator[]`	`getPostTranslatorsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns all configured post translators configured on the specified database in the service domain.

`java.lang.String`	`getPrimaryNrsIpAddress()` Return the IP address of the primary NRS.
`Route[]`	`getRoutesInGatewayEndpoint` `(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String gatewayEndpointName, java.lang.String dnPrefix, DnType dnType)`

Method summary	
	Returns all configured routes configured on the specified database in the gateway endpoint.

`Route[]`	`getRoutesInL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String dnPrefix, DnType dnType)`
	Returns all configured routes configured on the specified database in the L0 domain.

`Route[]`	`getRoutesInL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String dnPrefix, DnType dnType)`
	Returns all configured routes configured on the specified database in the L1 domain.

`Route[]`	`getRoutesInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String dnPrefix, DnType dnType)`
	Returns all configured routes configured on the specified database in the service domain.

`java.lang.String`	`getSecondaryNrsIpAddress()` Returns IP address of the alternate NRS.

`ServiceDomain`	`getServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns the service domain with the specified service domain name from the database.

`ServerStatus`	`getSipProxyServerStatus()` Determines if the SIP Proxy Server is enabled on this server.
`java.lang.String`	`getSoftwareVersion()`

Method summary	
	Returns the software version of the NRS server.
`SystemWideSetting`	`getSystemWideSettings()` Returns the System Wide Settings configuration from active database.

`int`	`getTotalNumCollaborativeServers`
	`(DatabaseInstance databaseInstance)`
	Returns the number of collaborative servers configured on the specified.

`int`	`getTotalNumDefaultRoutes`
	`(DatabaseInstance databaseInstance)`
	Returns the number of default routes configured on the database.

`int`	`getTotalNumGatewayEndpoints`
	`(DatabaseInstance databaseInstance)`
	Returns the number of gateway endpoints configured on the database.

`int`	`getTotalNumL0Domains`
	`(DatabaseInstance databaseInstance)`
	Returns the number of L0 domains configured on the database.

`int`	`getTotalNumL1Domains (DatabaseInstance databaseInstance)` Returns the number of L1 domains configured on the specifieddatabase.

`int`	`getTotalNumPostTranslators`
	`(DatabaseInstance databaseInstance)`
	Returns the number of post translators configured on the database.

`int`	`getTotalNumRoutes`
	`(DatabaseInstance databaseInstance)`
	Returns the number of routes configured on the database.

Method summary	

`int`	`getTotalNumServiceDomains`
	`(DatabaseInstance databaseInstance)`
	Returns the number of configured service domains from the specified database.

`int`	`getTotalNumUserEndpoints`
	`(DatabaseInstance databaseInstance)`
	Returns the total number of user endpoints configured on the database.

`UserEndpoint`	`getUserEndpoint`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName, java.lang.String userEndpointName)`
	Returns the User Endpoint with the specified ID from the database.

`UserEndpoint[]`	`getUserEndpointsInL0Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName, java.lang.String l0DomainName)`
	Returns all configured user endpoints configured on the specified database for the specified L0 domain.

`UserEndpoint[]`	`getUserEndpointsInL1Domain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName, java.lang.String l1DomainName)`
	Returns all configured user endpoints configured on the specified database for the specified L1 domain.

`UserEndpoint[]`	`getUserEndpointsInServiceDomain`
	`(DatabaseInstance databaseInstance, java.lang.String serviceDomainName)`
	Returns all configured user endpoints configured on the specified database for the specified service domain.

Method summary	
`void`	`restartGatekeeper()` Restarts the Gatekeeper.
`void`	`restartNcs()` Restarts the Network Connection Server.
`void`	`restartSipProxyServer()` Restarts the SIP Proxy Server.
`void`	`revert()` Performs a revert operation on the database.
`void`	`rollback()` Performs a rollback action on database.

`void`	`updateCollaborativeServer`
	`(java.lang.String originalAddress, CollaborativeServer collaborativeServer)`
	Updates an existing Collaborative Server on the system.

`void`	`updateDefaultRoute`
	`(DnType dnType, DefaultRoute defaultRoute)`
	Updates an existing Default Route on the system.

`void`	`updateGatewayEndpoint`
	`(java.lang.String endpointName, GatewayEndpoint gatewayEndpoint)`
	Updates a Gateway Endpoint on the system.

`void`	`updateL0Domain`
	`(java.lang.String l0DomainName, L0Domain domain)`
	Updates an L0 domain on the system.

`void`	`updateL1Domain(java.lang.String domainName, L1Domain l1Domain)` Updates an L1 domain on the system.

Method summary	

`void`	`updateNrsServer`
	`(NrsServer nrsServer)`
	Updates an NRS Server on the system.

`void`	`updatePostTranslator`
	`(java.lang.String originatingEndpointName, java.lang.String terminatingEndpointName, java.lang.String targetPhoneContext, int routingStringLength, java.lang.String digitToStart, PostTranslator postTranslator)`
	Updates a Post Translator on the system.

`void`	`updateRoute`
	`(java.lang.String dnPrefix, DnType type, Route route)`
	Updates a Routing Entry on the system.

`void`	`updateServiceDomain`
	`(java.lang.String domainName, ServiceDomain serviceDomain)`
	Updates an existing service domain on the system.

`void`	`updateSystemWideSettings`
	`(SystemWideSetting systemWideSettings)`
	Updates the System Wide Setting on the system.

`void`	`updateUserEndpoint`
	`(java.lang.String endpointName, UserEndpoint userEndpoint)`
	Updates a User Endpoint on the system.