A MITEL
PRODUCT
GUIDE

# OpenScape Voice V11

## Interface Manual Volume 8 Assistant API

Interface Manual: Volume 8, Assistant API

Description
04/2025

A31003-H80B0-T108-01-7618

Mitel

## Notices

The information contained in this document is believed to be accurate in all respects but is not warranted by Mitel Europe Limited. The information is subject to change without notice and should not be construed in any way as a commitment by Mitel or any of its affiliates or subsidiaries. Mitel and its affiliates and subsidiaries assume no responsibility for any errors or omissions in this document. Revisions of this document or new editions of it may be issued to incorporate such changes. No part of this document can be reproduced or transmitted in any form or by any means - electronic or mechanical - for any purpose without written permission from Mitel Networks Corporation.

## Trademarks

The trademarks, service marks, logos, and graphics (collectively “Trademarks”) appearing on Mitel’s Internet sites or in its publications are registered and unregistered trademarks of Mitel Networks Corporation (MNC) or its subsidiaries (collectively “Mitel), Unify Software and Solutions GmbH & Co. KG or its affiliates (collectively “Unify”) or others. Use of the Trademarks is prohibited without the express consent from Mitel and/or Unify. Please contact our legal department at iplegal@mitel.com for additional information. For a list of the worldwide Mitel and Unify registered trademarks, please refer to the website: http://www.mitel.com/trademarks.

# Contents

# 1 Introduction

The administration of the OpenScape Voice system is done by the OpenScape Voice Assistant, a plug-in of the Common Management Platform (CMP) which is a web based management application of the OpenScape solution. In addition to the web based graphical user interface the OpenScape Voice Assistant also offers a web services API. All operations and data structures used by this API are described in the form of a WSDL (Web Service Description Language) file The document at hand describes the content of this WSDL file for the OpenScape Voice Assistant API version 7.0 and the usage of the provided web services.

To use this document, the user should have knowledge about the general architecture of the OpenScape Voice and how it is managed. In addition, the user should have a general understanding of the SOAP (**S**imple **O**bject **A**ccess **P**rotocol) protocol.

## 1.1 The OpenScape Voice Assistant API implementation

The OpenScape Voice Assistant API is implemented as Java Web Services using the **Apache** implementation of the '**S**imple **O**bject **A**ccess **P**rotocol' (**SOAP**) called **Axis**. It is automatically installed on the same server as the OpenScape Voice V7 Assistant as Axis services running on a **Tomcat** web server.

## 1.2 The WSDL file

The OpenScape Voice Assistant API WSDL can be found and viewed at the URL

```
https://<yourOSVAssistant>/HiPath8000AssistantAPIv310/
services/HiPath8000AssistantAPI?wsdl
```

# 2 The OpenScape Voice Assistant Operations

## 2.1 General Information

The OpenScape Voice Assistant API operations are listed at the following URL:

`https://<yourOSVAssistantAddress>/HiPath8000AssistantAPIv310/services`

OpenScape Voice Assistant API operations are logically grouped as

- Session-Related Commands,
- System-Related Commands,
- Office Code-Related Commands,
- Directory Number-Related Commands,
- Numbering Plan-Related Commands,
- Global Numbering Plan Resources,
- Business Group-Related Commands,
- Subscriber-Related Commands,
- Feature Profile-Related Commands,
- DLS-Related Commands,
- Switch-Related Commands.

The methods, input and return parameters and values for these parameters are explained in detail in the following sections.

Most of the API operations can be used by authorized users only. Therefore, a session token should be requested and this token shall be added to every subsequent API call. The session token request is done via **openSession(username, password)** method. For finishing the session, this session should be closed with **closeSession(sessionToken)** method.

**Predefined User Profiles**

OpenScape Voice Assistant users have user profiles assigned to them. The profiles specify the access rights of the users. Depending on the profiles and the features assigned to the profiles, the user is either allowed or not allowed to perform certain operations. If a method is not allowed for a user, the API will return the proper error message. The administration of the profiles and the users is performed using the OpenScape Voice Assistant GUI.

It is recommended to create a dedicated user account for the API access. This can be done using the OpenScape Voice Assistant application.

**Multiple OpenScape Voice Switch Support**

The OpenScape Voice Assistant API supports the management of multiple OpenScape Voice switches. For this purpose, all switch specific API methods require the use of the switch name parameter. For example, the user can list business groups of an OpenScape Voice with name *switchName1* with a call to the list business group method as follows:

```
listBusinessGroups(sessionToken, switchName1)
```

**Multiple DLS Support**

A list of all available DLS servers (with active API) can be displayed to the user. This list contains several pieces of information, including DLS Server Name, DLS ID, DLS Status and Version, DLS URL, API URL and DLS Client Session ID.

DLS servers can be assigned to OpenScape Voice Business Groups using the create/modify BusinessGroup methods. When a DLS server is assigned to a business group, all subscriber specific methods of the OpenScape Voice Assistant will implicitly handle the update of DLS devices in a consistent manner according to the respective OpenScape Voice subscriber configuration.

## 2.2 Types of Operations of the API

There are typically five types of API operations:

- **create**

  Create and modify operations get the bean as an input parameter. They usually return **ResultStatus** bean. The input bean points to the valid bean like *OfficeCodeBean*, *BusinessGroupBean* …etc.

- **delete**

  Delete operations get the unique property of the bean like name or id as an input parameter. They also return **ResultStatus** bean.

- **get**

Get operations have the unique property of the bean like name or id as an input parameter. They return response bean of the operation, which includes **ResultStatus** bean and required bean.

- **modify**

  Create and modify operations get the bean as an input parameter. They return **ResultStatus** bean. The input bean points to the valid bean like *OfficeCodeBean*, *BusinessGroupBean* etc.

and

- **list**

  List operations usually do not have any input parameter except session token. They return response bean of the operation, which includes **ResultStatus** bean and required list of beans.

Office codes, directory numbers, business groups, subscribers, feature profiles are created, deleted, get, modifed and listed via API. Each of these operations requires session token as input parameter.

For enumerated values, constants are used. There are many types of constants like *ConstantsApplyRecall*, *ConstantsAuthorizationCode*…etc. Proper constant value should be set to the enumerated field.

For a better understanding of the usage of the available API operations, we present you some scenarios in Section , “API Usage Scenarios” after describing the methods in detail.

**ResultStatus**

*ResultStatus* bean contains the execution state of the operation. If any error has occurred during validation of parameters or during execution of the operation, the *currentState* field of *ResultStatus* includes the error or failure message. If the operation is executed successfully, this field contains an empty string.

## 2.3 Session-Related Commands

### 2.3.1 Open a Session (openSession)

Nearly all operations of the OpenScape Voice Assistant API need a session token. This token has to be retrieved via **openSession(username, password)** method.

### 2.3.2 Close a Session (closeSession)

The session token automatically expires after 30 minutes however a session can be explicitly closed with the use of the **closeSession(sessionToken)** method.

## 2.4 System-Related Commands

### 2.4.1 Show the API version (getApiServerVersion)

Every API version is related to a specific URL. The same API version always offers the same interface, regardless of the OpenScape Voice Assistant version number. Changes to the interface result in a new API version. At least two different API versions are supported at the same time. Therefore, your applications does have to be adapted immediately after interface changes occur.

Use **getApiServerVersion()** method to read the version number of the attached web services. The server version method returns "3.10" for this version.

The method does not require a session token.

### 2.4.2 Show the Software Versions (getSwVersions)

The user can retrieve information regarding the software status of the operating system via **getSwVersions (sessionToken, switchName)** method. The available information concerning the software status are: HiPath8000 version, Assistant version, Patchset level and Build number. The *AssistantSwVersionBean* includes those versions.

## 2.5 Office Code-Related Commands

Office codes must be created before subscribers can be allocated to the switch.

The office code is comprised of country code, area code, location code (i.e. local office code) consecutively.

The user can create, delete and list office codes using the API. The office code bean, named *OfficeCodeBean*, is used for create and list operations.

**Office Code Bean (*OfficeCodeBean*)**

The fields of the office code bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| officeCode | The office code is composed of country code, area code, location code (i.e. local office code). | 1..9 | Numeric string | |
| countryCode | The country code. | 0..4 | Numeric string | |
| areaCode | The area code. | 0..6 | Numeric string | |
| locationCode | The location code (i.e. local office code). | 1..8 | Numeric string | Required |

*Table 1 Fields of the office code bean*

### 2.5.1 List Office Codes (listOfficeCodes)

For listing all available office codes, the **listOfficeCodes(sessionToken, switchName)** method is used. For each office code you will receive its code in a plain and a structured way, also separated in country code, area code and location code (i.e. local office code).

### 2.5.2 Create an Office Code (createOfficeCode)

Creating an office code is the first thing you have to do if you plan to create a business group. To create an office code, **createOfficeCode(sessionToken, switchName, inputBean)** method is used. The *inputBean* points to an *OfficeCodeBean*. This method returns **CreateOfficeCodeResult** which includes the result status and created office code. This is the only create method returning the result bean instead of **ResultStatus**.

The office code is created by concatenation of country code, area code and location code (i.e. local office code), which are specified in the office code bean.

To create an office code, at least a 1-digit non-empty location code (i.e. local office code) shall be provided, and office code should not exceed 9 digits.

### 2.5.3 Delete an Office Code (deleteOfficeCode)

Deleting an office code is done via **deleteOfficeCode(sessionToken, switchName, officeCode)** method. An existing office code shall be provided for successful deletion.

This operations will not be successful in case the office code is still used.

## 2.6 Directory Number-Related Commands

The directory numbers for an office code can be created, deleted and listed by API. Moreover, vacant directory numbers of an office code can be listed. The directory number bean, *DirectoryNumberBean*, is used for list operations.

**Directory Number Bean (*DirectoryNumberBean*)**

The fields of the directory number bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| officeCode | The office for this directory number. | 1..9 | Numeric string | Required |
| extension | The directory number. | 3..6 | Numeric string | Required |

*Table 2 Fields of directory number bean*

### 2.6.1 List Directory Numbers of an Office Code (listDirectoryNumbers)

To get all available directory numbers of an office code, **listDirectoryNumbers (sessionToken, switchName, officeCode)** method is used. The list of directory number beans, named *DirectoryNumberBean*, is retrieved.

### 2.6.2 Add Directory Numbers to an Office Code (createDirectoryNumbers)

With **createDirectoryNumbers(sessionToken, switchName, officeCode, startExtension, endExtension, businessGroup)**, you can add a range of directory numbers in the specified interval of start extension and end extension to an office code.

All parameters of the method are mandatory except the businessGroup. The attribute businessGroup is optional and if provided it would reserve the created directory numbers. The valid values for the parameters shall be provided. Start extension and end extension are usually 4-digit integers and they should have the same digit length.

To create a range of directory numbers, endExtension must be greater than the startExtension number. To create a single directory number, last extension number must equal the start extension number.

### 2.6.3 Delete a Directory Number (deleteDirectoryNumbers)

With **deleteDirectoryNumber(sessionToken, switchName, officeCode, extension)** method, you can delete any directory number. Existing office code and extension of the directory number shall be provided for successful deletion.

### 2.6.4 List Vacant Directory Numbers (listVacantDirectoryNumbers)

For creating subscribers you need a directory number not yet used. With **listVacantDirectoryNumbers (sessionToken, switchName, officeCode)** method, a list of all vacant extensions for a given office code can be retrieved.

## 2.7 Numbering Plan-Related Commands

The numbering plans can be listed by API user. Moreover, subscribers of the numbering plans can be listed.

**Numbering Plan Bean (*NPBean*)**

The fields of the numbering plan bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| name | The name of the numbering plan. | | Character String | |
| type | The numbering plan type. | | Character String | |

*Table 3 Fields of the numbering plan bean*

### 2.7.1 List Numbering Plans (listPrivateNumberingPlans)

The numbering plans can be retrieved using **listPrivateNumberingPlans(sessionToken, switchName, businessGroupName)** method. This method retrieves the private numbers of given business group name in the list of *NPBean*.

### 2.7.2 List Numbering Plan Subscribers (listNumberingPlanSubscribers)

The subscribers of the numbering plans can be listed using **listNumberingPlanSubscribers (sessionToken, switchName, businessGroupName, numberingPlanName)** method. The business group name of the numbering plan and the name of the numbering plan shall be provided.

The list of *subscriberBean* with summary info is retrieved as a result. Resulting subscriber beans include service id, feature profile name, display name and bg line name (i.e. subscriber name) properties of the subscriber.

## 2.8 Global Numbering Plan Resources

### 2.8.1 List Rate Areas (listRateAreas)

You can list all rate areas (i.e. Routing Areas) via **listRateAreas(sessionToken, switchName)** method. The list of rate areas (i.e. routing areas), named *RateAreaBean*, will be retrieved as a result.

### 2.8.2 List Classes of Service (listClassesOfService)

You can list classes of service using **listClassesOfService(sessionToken, switchName)** method. The list of classes of service, named *ClassOfServiceBean*, will be retrieved as a result.

### 2.8.3 List Calling Locations (listCallingLocations)

The calling locations can be retrived using **listCallingLocations(sessionToken, switchName)** method. The list of calling locations, named *CallingLocationBean,* will be retrieved as a result.

## 2.9 Business Group-Related Commands

A Business Group (BG) is an entity of related subscribers.

The user can create, delete, modify, get and list business groups by using API. The generic business group bean, named *BGBean*, is used for create, modify and get operations. The list method returns list of business group list bean, named *BGListBean*.

### Business Group Bean (*BGBean*)

The fields of the business group bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| businessGroupName | The business group name. It should be unique. | 1..28 | Character String | Required |
| defaultOfficeCode | Existing office code shall be provided. | 1..9 | Numeric String<br>Any existing office code[1] | Required |
| displayNumber | The display number, shown when a subscriber makes an external call. It should be unique. | 1..15 | Numeric String | Required |
| numberingPlan | Default numbering plan for the business group. | | | |
| cdrActive | Activate/deactivate value for call data recording. | | True, False | False |
| cdrId | Identification of customer during call data recording. | 10 | Numeric String | |
| emergencyAnnouncement | Emergency announcement of Emergency LIN Administration service. | | Character String<br>Any existing intercept announcement[2] | |
| emergencyNumber | Emergency number of Emergency LIN Administration service. | | Numeric String<br>Any existing subscriber ID[3] | |
| hotDeskHH | Hour value of the auto-logoff time of hot desking service.<br>Auto-logoff time is the time at which all subscribers' hot desking session will be automatically terminated. | | 1 to 24 hour | |
| hotDeskmm | Minute value of the auto-logoff time of hot desking service. | | 0 to 59 min | |
| nightBellCPUActive | Activate/deactivate Night Bell Call Pickup Group service, which is a group of phones that ring simultaneously.<br>Incoming calls can be picked up by any Subscriber in the Business Group that the Night Bell CPU Group is associated with. | | True, False | False |
| nightBellCPUId | The night bell call pickup group. | | Numeric String<br>Any existing pickup group[4] | |

*Table 4 Fields of the business group bean*

1 Office codes can be retrieved using **listOfficeCodes** method (Section 2.5.1, “List Office Codes (listOfficeCodes)”)
2 Emergency announcements can be retrieved by **listInterceptAnnouncement** method (Section 2.10.7, “List Intercept Announcements (listInterceptAnnouncements)”)
3 Subscribers of the business group can be retrieved by **listBusinessGroupSubscribers** method (List the Subscribers of a Business Group (listBusinessGroupSubscribers))
4 Call pickup groups can be retrieved by **listCallPickupGroups** method (Section 2.9.11, “List the Call Pickup Groups of a Business Group (listCallPickupGroups)”)

**Business Group List Bean (*BGListBean*)**

This is the summary bean of business group. It is only used for listing purposes.

The fields of the business group list bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| businessGroupName | The business group name. | | Character String | |
| areaCode | The area code of the office code | | Numeric String | |
| countryCode | The country code of the office code | | Numeric String | |
| localOfficeCode | The location code (i.e. local office code) of the office code | | Numeric String | |
| displayNumber | The display number, shown when a subscriber makes an external call. | | Numeric String | |
| defaultFeatureProfile | The default feature profile | | Character String | |
| numberSubscribers | The number of subscribers | | | |

*Table 5 Fields of the business group list bean*

## 2.9.1 List Business Groups (listBusinessGroups)

You can list all available business groups via **listBusinessGroups (sessionToken, switchName)** method and you will receive a list of all business groups, *BGListBean*’s, including their name, their display number and their amount of associated subscribers.

The list business group method retrieves only some properties of the business groups. If you need the details of the business group, you should use the *getBusinessGroup* method.

## 2.9.2 Create a Business Group (createBusinessGroup)

With **createBusinessGroup(sessionToken, switchName, inputBean)** method, you can create a new business group. The *inputBean* parameter is of type *BGBean*.

It expects at least **businessGroupName**, **defaultOfficeCode** and **displayNumber** being set. Business group name should be unique in the OpenScape Voice node. Display number should be unique in the OpenScape Voice node, too.

Existing office code should be provided. The **listOfficeCodes** method can be used to obtain available office codes.

Business Group specific services are

- Hot Desking,
- Night Bell Call Pickup,
- Emergency LIN Administration.

These services can be activated or deactivated.

Existing values for call pickup group, emergency number and emergency announcement should be set. Proper list methods can be used for retrieveing existing values.

## 2.9.3 Delete a Business Group (deleteBusinessGroup)

For deleting a business group, **deleteBusinessGroup(sessionToken, switchName, businessGroupName)** method is used. Existing business group name shall be provided for successful deletion.

## 2.9.4 Modify a Business Group (modifyBusinessGroup)

You can modify a business group using **modifyBusinessGroup(sessionToken, switchName, inputBean)** method. As an *inputBean*, you have to provide a valid business group bean. Either you can create this bean from scratch or you can first retrieve a business group using *getBusinessGroup* command. If you're creating from scratch, you have to set at least *businessGroupName*, *defaultOfficeCode* and *displayNumber* in the bean.

## 2.9.5 List all properties of a Business Group (getBusinessGroup)

In order to retrieve all properties of a business group, you can use **getBusinessGroup (sessionToken, switchName, businessGroupName)** method. Existing business group name shall be provided. You will get a business group bean as return value which contains all the properties of the business group.

## 2.9.6 List the Subscribers of a Business Group (listBusinessGroupSubscribers)

To list the subscribers, **listBusinessGroupSubscribers(sessionToken, switchName, businessGroupName)** method is used. If the *businessGroupName* parameter is empty, all subscribers of the system (for all Business Groups) are listed. If the *businessGroupName* parameter is not empty, all subscribers of the mentioned Business Group are listed. In the result, you find all valid subscriber IDs which can be used for further operations.

## 2.9.7 List the SIP Subscribers of a Business Group (listBusinessGroupSubscribersSip)

To list all SIP details of a business group, **listBusinessGroupSubscribersSip(sessionToken, switchName, businessGroupName)** method is used.

## 2.9.8 List the SIP Subscriber ID passed to DLS (listBusinessGroupSubscribersSipV4)

As of OpenScape Voice V3.1 R3, in order to list the Internal Display Name (display ID) of all SIP subscribers of a Business Group the following method is used:

**listBusinessGroupSubscribersSipV4(sessionToken, switchName, businessGroupName)**

### 2.9.9 List the Feature Profiles of a Business Group (listFeatureProfiles)

You can list all profiles of a business group or the system wide feature profiles via the **listFeatureProfiles(sessionToken, switchName, businessGroupName, featureProfileType)** method. The featureProfileType is used as a filter criterion with the following possible values:

- FP_LIST_TYPE_ALL: Returns all Feature Profiles from all Bussiness Groups and the System wide Feature Profiles
- FP_LIST_TYPE_SYSTEM_ONLY: Returns all System wide Feature Profiles
- FP_LIST_TYPE_BG_UNION_SYSTEM: Returns the Feature Profiles of the Business Group specified in the `businessGroupName` attribute and the System wide Feature Profiles
- FP_LIST_TYPE_BG_ONLY: Returns the Feature Profiles of the Business Group specified in the `businessGroupName` attribute.

### 2.9.10 List the Departments of a Business Group (listDepartmentsOfBusinessGroup)

The departments of a business group can be listed via **listDepartmentsOfBusinessGroup (sessionToken, switchName, businessGroupName)** method. You will get a list of departments of the business group as a result.

### 2.9.11 List the Call Pickup Groups of a Business Group (listCallPickupGroups)

To list all call pickup groups of a business group, **listCallPickupGroups (sessionToken, switchName, businessGroupName)** method is used. You will get a list of pickup groups of the business group as a result.

## 2.10 Subscriber-Related Commands

You can create, delete, modify, get and list subscribers by using API. The subscriber bean, named *SubscriberBean*, is used for create, modify and get operations. Moreover, the list method returns list of subscriber beans.

### Subscriber Bean (*SubscriberBean*)

The fields of the subscriber bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceID | The service ID of the subscriber. It is composed of office code and extension. | | Numeric String | |
| businessGroupName | The business group name. | | Character String<br>Any existing business group name[1]. | Required |
| officeCode | The office code. | 1..9 | Numeric String<br>Any existing office[2]. | Required |
| extension | The directory number of the subscriber. | 3..6 | Numeric String<br>Any vacant directory number[3]. | Required |
| displayName | The external display name of the subscriber. | 0..30 | Character String | System user name |
| bgLineName | The internal display name of the subscriber. | 0..30 | Character String | System user name |
| pickupGroup | The call pickup group. | | Numeric String<br>Any existing call pickup group[4]. | |
| COS | The class of services. | | Character String<br>Any existing class of service[5]. | |
| rateArea | The rate area. | | Character String<br>Any existing rate area[6]. | |
| callingLoc | The calling location. | | Character String<br>Any existing calling location[7]. | |
| SIP PHONE Fields (They make sense, if Connection Info is SIP Phone) | | | | |
| phoneName | The name for the SIP phone. It is used during the registration of this phone. | 0..15 | Character String. Should be unique. | Required, if Connection Information is SIP Phone |
| regType | The type of the SIP registration. | Enum | Static<br>Dynamic | Dynamic |
| serviceType | The SIP transport type. | Enum | UDP<br>TCP<br>TLS | UDP |
| ipaddress | The IP adress for SIP phone. It is meaningful when the registration type is static. | | Standard IP address convention (XXX.XXX.XXX.XXX) | Required, if registration type is static |

*Table 6 Fields of subscriber bean*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| port | The port for SIP phone.It is meaningful when the registration type is static. | 0..5 | Numeric String<br>Standard port number convention 1..9999 | Required, if registration type is static |
| realm | The domain of SIP phone. | 0..128 | Character String | |
| userName | The user name for the SIP phone user. | 6...64 | Character String | |
| passWord | The password for the SIP phone user. | 6...20 | Character String | |
| Advanced Fields | | | | |
| bgDeptName | The department name | | Character String<br>Any existing department name[8] | |
| timezone | The timezone | Enum | GMT+0, GMT+1...,<br>GMT+1 | |
| PIN | The personal identification number.<br>This code is used by various service features where a PIN is needed, unless that feature has its own PIN. | 0..10 | Numeric String | |
| SIP(connectionInfo) | The connection information. Profile only means that you create a subscriber who does not have a connection. The subscriber cannot use Remote Call Forwarding (RCF). | Enum | SIP Phone, Profile Only | if profile info value is selected, above SIP Phone Fields does not make sense. |
| externalDNFlag | The flag that specifies if the subscriber is an external directory number. External directory number means that a caller can dial this number from outside to reach an internal subscriber. | | True, False | True |
| bgAttendantNumber | The flag specifying if the subscriber is an attendant number. | | True, False | False |
| keyset | The keyset. | Enum | None,<br>Primary Line,<br>Phantom Line | None |
| featureProfileName | The feature profile name. | | Character String.<br>Any existing feature profile name[9] | |
| featureProfileBGName | The business group name of the feature profile. | | Character String<br>The business group name of the selected feature profile name | |

*Table 6 Fields of subscriber bean*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| macAddress | The mac address of the subscriber | | Number String sperated by colons | |
| deviceProfile | The device profile of the subscriber | | Chracter String, Proper device profile from DLS | |
| plugAndPlayStandard | The plug and play standard of the subscriber | | | |
| dlsId | id of the dls server | | Numeric String, Proper dls server id[10] | |
| dlsName | the name of the dls server | | Character String, Proper dls server name[11] | |

*Table 6 Fields of subscriber bean*

1 Defined business groups can be retrieved using the **listBusinessGroups** method (Section 2.9.1, “List Business Groups (listBusinessGroups)”)
2 The available list of office codes can be retrieved using the **listOfficeCodes** method (Section 2.5.1, “List Office Codes (listOfficeCodes)”)
3 The available vacant directory numbers of the office code can be retrieved using the **listVAcantDirectoryNumbers** method (Section 2.6.4, “List Vacant Directory Numbers (listVacantDirectoryNumbers)”)
4 The available list of call pickup groups can be retrieved using the **listCallPickupGroups** method (Section 2.9.11, “List the Call Pickup Groups of a Business Group (listCallPickupGroups)”)
5 The available list of class of services can be retrieved using the **listClassOfServices** method (Section 2.8.2, “List Classes of Service (listClassesOfService)”)
6 The available list of rate areas can be retrieved using the **listRateAreas** method (Section 2.8.1, “List Rate Areas (listRateAreas)”)
7 The available list of calling locations can be retrieved using the **listCallingLocations** method (Section 2.8.3, “List Calling Locations (listCallingLocations)”)
8 The available list of department names can be retrieved using the **listDepartmentsOfBusinessGroups** method (Section 2.9.10, “List the Departments of a Business Group (listDepartmentsOfBusinessGroup)”
9 Defined feature profiles can be retrieved using the **listFeatureProfiles** method (Section 2.11.2, “List all Feature Profiles(listFeatureProfiles)”)
10 Available dls server id’s can be retrieved using the **listDLS** method (Section 2.12.1, “List DLS Servers (listDLS)”)
11 Available dls server names can be retrieved using the **listDLS** method (Section 2.12.1, “List DLS Servers (listDLS)”)

## 2.10.1 List Subscribers (listBusinessGroupSubscribers)

Please refer to Section 2.9.6, “List the Subscribers of a Business Group (listBusinessGroupSubscribers)”.

## 2.10.2 List all Properties of a Subscriber (getSubscriber)

With **getSubscriber(sessionToken, switchName, subscriber)**, the all properties of a subscriber can be retrieved. ServiceID of the subscriber shall be provided as the third parameter. A subscriber bean containing all the properties is retrieved as return value.

### 2.10.3 List SIP details of the Subscribers (listBusinessGroupSubscribersSip)

Please refer to Section 2.9.7, “List the SIP Subscribers of a Business Group (listBusinessGroupSubscribersSip)”).

### 2.10.4 Create a Subscriber (CreateSubscriber)

For creating a subscriber, **createSubscriber(sessionToken, switchName, inputBean)** method is used. The parameter *inputBean* point s to a valid *SubscriberBean*.

Input bean expects at least values for the bean elements *businessGroupName*, *officeCode* and *extension*. Existing office code and vacant directory number as an extension should be provided. As service ID is composed of office code and extension, it should not be assigned to any value during creation. It will be ignored even if it is assigned to a value.

Available office codes can be retrieved by **listOfficeCodes** method. Vacant directory numbers of the office code can be retrieved by **listVacantDirectoryNumbers** method.

The display name and Business Group line name (i.e. subscriber name) fields are automatically set to the user name, if they are left empty. The maximum length is 30 characters.

Existing call pickup group, class of service, routing area, calling location and department should be provided. Available call pickup groups can be retrieved by **listCallPickupGroups** method. Available classes of service can be retrieved by **listClassOfServices** method. Available routing areas can be retrieved by **listRateAreas** method. Available calling locations can be retrieved by **listCallingLocations** method. Available departments can be retrieved by **listDepartments** method.

Connection information (SIP) field can be SIP phone or profile only. *ConstantsConnectionInfo* shall be used to set the value. Profile only means that you create a subscriber who does not have a connection and the subscriber cannot use Remote Call Forwarding (RCF). If connection information is SIP phone, phone name is a mandatory field. It shall be unique and can have a maximum length of 15 characters.

The SIP registration type can be Dynamic or Static. *ConstantsRegType* shall be used to set the value. Default value is dynamic registration. In static registration, IP address and port shall be given.

The time zone is of the type *ConstantsTimezone*.

The keyset is of the type *ConstantsKeysetUse*.

### 2.10.5 Modify a Subscriber (modifySubscriber)

You can modify a subscriber using **modifySubscriber(sessionToken, switchName, inputBean)**. *inputBean* points to a valid *SubscriberBean*. You can either create this bean from scratch or you can first retrieve a subscriber using *getSubscriber.* For modifying a subscriber bean, you have to set at least service ID, office code, business group name and extension in the bean.

### 2.10.6 Delete Subscriber (deleteSubscriber)

To delete one subscriber, **deleteSubscriber(sessionToken, switchName, subscriber, businessGroupName, dlsName, dlsId)** method is used. The subscriber parameter expects valid subscriberID and the business group name parameter expects correct business group name. The DLS Name and DLS Id parameters are required in order that the corresponding subscriber entry in the DLS is deleted as well.

### 2.10.7 List Intercept Announcements (listInterceptAnnouncements)

The intercept announcements can be listed using **listInterceptAnnouncements (sessionToken, switchName)** method. This method retrieves a list of intercept announcements.

## 2.11 Feature Profile-Related Commands

You can create, delete, modify, get and list feature profiles by using API. The feature profile bean, named *FeatureProfileBean*, is used for create, modify and get operations. Moreover, the list method returns list of feature profile beans.

**Feature Profile Bean (*FeatureProfileBean*)**

The fields of the subscriber bean are the following:

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| businessGroupName | The business group name. | | Character String Any existing business group name[1]. | Required |

*Table 7 Fields of the subsrciber bean*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| featureProfileName | The name of the feature profile. | 1..40 | Character String Should be unique for business group | Required |
| fpDefault | The flag for the feature profile being default or not | | True, False | False |
| listOfFeaturesBean | The list of feature beans. It keeps all services of the feature profile. Details of the services are given in the Features section. | | List of valid features | |

*Table 7 Fields of the subsrciber bean*

1 Defined business groups can be retrieved using the **listBusinessGroups** method (Section 2.9.1, “List Business Groups (listBusinessGroups)”)

## 2.11.1 Features

The features of the feature profile are described in this section.

### Base Feature Bean - FeatureBean

Feature bean is the base class for the features of the feature profile. All features extends this bean.

The following table lists the parameters of this base bean.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| featureName | The name of the feature. | | Character String | Required |
| featureSubscribed | Subscribes/ unsubscribes the service feature.<br>Features shall be subscribed before configuring it. | | True, False | False |
| featureActive | Activate/ deactivates the feature. | | True, False | False |

*Table 8 Parameters of base feature bean*

### Automatic Callback Feature - *CfACBean*

When a subscriber calls a number that is busy, this service enables the subscriber to have the switch call the number back when the number becomes available.

This feature shall be subscribed before setting other properties.

The following table lists the input and output parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| ARSwitchControl | The callback switch control. Ability to call back subscribers across switches or restrict call back to subscribers in that switch alone. | Enum | InterSwitch, IntraSwitch, UseSwitchSetting | |

*Table 9 Input and output parameters of automatic callback feature*

### Account Code Feature - *CfAcctCodeBean*

This feature allows the customer to add a special number sequence to the Call Detail Record (CDR) for billing purposes.

### Anonymous Caller Reject Feature - *CfACRBean*

This feature enables subscribers to forward callers who have restricted their calling number presentation status to an anonymous denial announcement.

### Automatic Recall Feature - *CfARBean*

This feature enables subscribers to enter an access code to redial the number of the last incoming call.

The following table lists the input and output parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| ARSwitchControl | The recall switch control. Ability to recall subscribers across switches or restrict call back to subscribers in that switch alone. | Enum | InterSwitch, IntraSwitch, UseSwitchSetting | |

*Table 10 Input and output paramters of the automatic recall feature*

### Authorization Code Feature - *CfAuthCodeBean*

This feature enables a subscriber to make certain external calls. It keeps authorization code for CDR Report and authorization codes for off-net traffic types.

The following table lists the input and output parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| authCodeCDRType | The authorization code CDR type. Specifies how Authorization Code is to be saved in CDR records. It can be saved as an authorization code, as an account code, as both authorization code and account code or can not be saved. | Enum | None, AccountCode, AuthorizationCode, Both | None |
| Authorization Codes for off-net Traffic Types<br>(The parameters in this section specifies which traffic types require Off-Net Authorization) | | | | |
| emergency | Determines if emergency require off-net authorization. | | True, False | False |
| homeDirAssist | Determines if home directory assistance require off-net authorization. | | True, False | False |
| interLATA | Determines if inter lata traffic type require off-net authorization. | | True, False | False |
| international | Determines if international traffic type require off-net authorization. | | True, False | False |
| internationalZone1 | Determines if international zone traffic type require off-net authorization. | | True, False | False |
| intraLATA | Determines if intra lata traffic type require off-net authorization. | | True, False | False |
| localDirAssist | Determines if local directory assistance require off-net authorization | | True, False | False |
| homeDirAssist | Determines if home directory assistance require off-net authorization. | | True, False | False |
| tollFree | Determines if toll free traffic type require off-net authorization. | | True, False | False |

*Table 11 Input and output parameter of authorization code feature*

### Name Delivery Feature - *CfBgCNAMBean*

This feature enables to display the incoming caller's name to the subscriber.

### Call Pickup Group Feature - *CfBgCPUBean*

This feature enables the subscriber to be assigned to a call pickup group and then to retrieve calls within that group.

### Outgoing CID Presentation Status Feature - *CfBgDAPPSBean*

This feature indicates the presentation status for caller ID services.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| subPpsIntraBgCalls | The presentation status | Enum | Public,<br>Private | Public |
| Display order of callers within Business Group | | | | |
| callingIntraBGDisplayFirst | The first display order of callers within business group | Enum | Business Group Name,<br>Subscriber Name,<br>Department Name | Business Group Name |
| callingIntraBGDisplaySecond | The second display order of callers within business group | Enum | Business Group Name,<br>Subscriber Name,<br>Department Name | Business Group Name |
| callingIntraBGDisplayThird | The third display order of callers within business group | Enum | Business Group Name,<br>Subscriber Name,<br>Department Name | Business Group Name |
| Display order of callers outside Business Group | | | | |
| callingNonIntraBGDisplayFirst | The first display order of callers outside business group | Enum | Business Group Name,<br>Subscriber Name | Business Group Name |
| callingNonIntraBGDisplaySecond | The second display order of callers outside business group | Enum | Business Group Name,<br>Subscriber Name | Business Group Name |
| Display order of connected parties within Business Group | | | | |
| connectedIntraBGDisplayFirst | The first display order of connected parties within business group | Enum | Business Group Name,<br>Subscriber Name,<br>Department Name | Business Group Name |
| connectedIntraBGDisplaySecond | The second display order of connected parties within business group | Enum | Business Group Name,<br>Subscriber Name,<br>Department Name | Business Group Name |
| connectedIntraBGDisplayThird | The third display order of connected parties within business group | Enum | Business Group Name,<br>Subscriber Name,<br>Department Name | Business Group Name |

*Table 12 Parameters of outgoing CID presentation status feature*

### Outgoing CID Presentation Status Plus Feature - CfBgDNPPSBean

This feature indicates the default setting of the subscriber's number (private or public) when making calls.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| subPpsIntraBgCalls | The presentation status | Enum | Public,<br>Private | Public |
| state | The default presentation status | Enum | Public, Private | Public |
| externalCallerID | The external caller ID to be used for number caller ID purposes for non-intra BG calls. | Enum | Public, Private | Public |

*Table 13 Parameters of outgoing CID presentation status plus feature*

### Call Forward Busy Feature - *CfCCWBean*

This service feature enables users to forward calls to another destination if the original destination is busy. If a subscriber is subscribed to Call Forward Busy, it cannot be subscribed to Voice Mail.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceId | The redirect number. | 30 | Numeric String,<br>Any related subscriber's service ID | Required when this feature is activated |
| subActivatable | The method by which the subscriber can activate/ deactivate this feature. | Enum | None ( subscriber cannot activate or<br>deactivate this<br>feature)<br>Phone ( subscriber can activate or deactivate this feature via the phone)<br>Web( subscriber can activate or deactivate this feature via the Web).<br>All (subscriber can activate or deactivate this feature via either Web or Phone) | None |

*Table 14 Parameters of call forward busy feature*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| destServiceIdSubControllable | The method of specifying redirect number by which the subscriber can change the destination of Call Forwarding.<br>This field may only be set to Phone only if SubActivatable is set to Phone or All as the subscriber can only change the destination when the feature is activated. | Enum | None(Subscriber can not change the destination for this feature).<br>Phone(subscriber can change the destination for this feature via the Phone).<br>Web(subscriber can change the destination for this feature via the Web). All(subscriber can change the destination for this feature via either the Web or phone) | None |
| CFCourtesyCallBehavior | The courtesy call behavior that occurs when a subscriber updates the Destination ServiceId when this feature is activated.<br>This field can be specified only if DestServiceIdSubControllable is set to Phone or All, since there are no courtesy calls placed when the number is set via the web. | Enum | No Courtesy Call,<br>Required,<br>Not Required | |
| notifySubCallFwdActive | Enables timed reminder. If set to True, the user is reminded that call forwarding is currently active. | | True, False | False |
| CFNotifyCallingParty | The notify calling party. | Enum | Notification (no announcement is played to the calling party).<br>Include calling number(announcement with the forwarding number is played to the calling party).<br>Do not include calling number( announcement that is played to the calling party does not include the forwarding number) | Notification |

*Table 14 Parameters of call forward busy feature*

### Call Forward No Reply Feature - *CfCFDABean*

This feature indicates the disposition of an incoming call to the subscriber when the subscriber does not answer the call. The caller is redirected to the number specified after the number of ring cycles specified (about 6 seconds per ring).

The following table lists parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceId | The redirect number. | 30 | Numeric String,<br>Any related subscriber's service ID | Required when this feature is activated |
| subActivatable | The method by which the subscriber can activate/ deactivate this feature. | Enum | None (subscriber cannot activate or deactivate this feature)<br>Phone (subscriber can activate or deactivate this feature via the phone)<br>Web (subscriber can activate or deactivate this feature via the Web).<br>All (subscriber can activate or deactivate this feature via either Web or Phone) | None |
| destServiceIdSubControllable | The method of specifying redirect number by which the subscriber can change the destination of Call Forwarding.<br>This field may only be set to Phone only if SubActivatable is set to Phone or All as the subscriber can only change the destination when the feature is activated. | Enum | None (Subscriber can not change the destination for this feature).<br>Phone (subscriber can change the destination for this feature via the Phone).<br>Web (subscriber can change the destination for this feature via the Web).<br>All (subscriber can change the destination for this feature via either the Web or phone) | None |
| ringDuration | The duration for which the phone will ring before (if not answered) the call will be redirected. | | 0 to 60 sec | |
| CFCourtesyCallBehavior | The courtesy call behavior that occurs when a subscriber updates the Destination ServiceId when this feature is activated.<br>This field can be specified only if DestServiceIdSubControllable is set to Phone or All, since there are no courtesy calls placed when the number is set via the web. | Enum | No Courtesy Call,<br>Required,<br>Not Required | |
| notifySubCallFwdActive | Enables timed reminder.<br>If set to True, the user is reminded that call forwarding is currently active. | | True, False | False |

*Table 15 Parameters of call forward no reply feature*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| CFNotifyCallingParty | The notify calling party. | Enum | Notification(no announcement is played to the calling party).<br>Include calling number (announcement with the forwarding number is played to the calling party).<br>Do not include calling number (announcement that is played to the calling party does not include the forwarding number). | Notification |

*Table 15 Parameters of call forward no reply feature*

### Call Forward Unconditional Feature - *CfCFVBean*

This service feature indicates the subscriber-specified disposition of an incoming call to the subscriber. The caller is redirected to the number specified. If this feature is enabled, the subscriber's phone receives a courtesy ring when a call is forwarded.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceId | The redirect number. | 30 | Numeric String,<br>Any related subscriber's service ID | Required when this feature is activated |
| subActivatable | The method by which the subscriber can activate/ deactivate this feature. | Enum | None (subscriber cannot activate or deactivate this feature)<br>Phone (subscriber can activate or deactivate this feature via the phone)<br>Web (subscriber can activate or deactivate this feature via the Web).<br>All (subscriber can activate or deactivate this feature via either Web or Phone) | None |

*Table 16 Parameters of call forward unconditional feature*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| destServiceIdSubControllable | The method of specifying redirect number by which the subscriber can change the destination of Call Forwarding.<br>This field may only be set to Phone only if SubActivatable is set to Phone or All as the subscriber can only change the destination when the feature is activated. | Enum | None (Subscriber can not change the destination for this feature).<br>Phone (subscriber can change the destination for this feature via the Phone).<br>Web (subscriber can change the destination for this feature via the Web).<br>All (subscriber can change the destination for this feature via either the Web or phone). | None |
| CFCourtesyCallBehavior | The courtesy call behavior that occurs when a subscriber updates the Destination ServiceId when this feature is activated.<br>This field can be specified only if DestServiceIdSubControllable is set to Phone or All, since there are no courtesy calls placed when the number is set via the web. | Enum | No Courtesy Call,<br>Required,<br>Not Required | |
| notifySubCallFwdActive | Enables timed reminder.<br>If set to True, the user is reminded that call forwarding is currently active. | | True, False | False |
| CFNotifyCallingParty | The notify calling party. | Enum | Notification (no announcement is played to the calling party).<br>Include calling number (announcement with the forwarding number is played to the calling party).<br>Do not include calling number (announcement that is played to the calling party does not include the forwarding number). | Notification |

*Table 16 Parameters of call forward unconditional feature*

### Call Hold Feature - *CfCHLDBean*

This feature enables the user to put a call in progress on hard hold by hook flashing and dialing an access code. Only one call per station line can be held at a time. The held call cannot be added to the other call. Dialing the call hold access code a second time can retrieve the original connection. Hanging up results in the held call re-calling the subscriber.

This feature is not valid for SIP endpoints.

### Outgoing CID Suppression Feature - *CfCIDSBean*

This feature indicates whether the subscriber can control the presentation of their name and ID when making a call. Activation codes indicate whether to deliver or suppress this information. Blocks or delivers both name and number.

### Called Name Delivery Feature - *CfCISNAMEBean*

This feature indicates that when the subscriber originates a call, the name of the called party is delivered to the originating phone for display.

This feature is set as subscribed  by default and cannot be changed.

### Called Name Delivery Feature - *CfCISNAMEBean*

This feature indicates that when the subscriber originates a call, the number of the called party is delivered to the originating phone for display.

This feature is set as subscribed  by default and cannot be changed.

### Outgoing Name Delivery Blocking Feature - *CfCNABBean*

This feature enables the subscriber to use access codes on a per call basis to toggle the calling number presentation status to the called party.

### Outgoing Number Delivery Blocking Feature - *CfCNDBBean*

This feature enables the subscriber to use access codes on a per call basis to toggle the calling name presentation status to the called party.

### Caller ID Feature - *CfCNDBean*

This feature enables the subscriber to see the phone number of a caller.

This feature is set as subscribed  by default and cannot be changed.

### CSTA Feature - *CfCSTABean*

This feature enables a subscriber to use third-party call control that allows PC applications to control calls to and from a co-located SIP Phone.

The following table lists the parameters associated with this feature

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| cstaType | The type of CSTA specifying the capabilities of the endpoint. | Enum | Normal,<br>Siemens Type 1,<br>GCP Residential, MGCP Business, Special 1,<br>Special 2,<br>Special 3,<br>Special 4,<br>Special 5 | Normal |

*Table 17 Parameters of CSTA feature*

### Call Transfer Feature - *CfCTBean*

This feature enables the subscriber to transfer a call to a different phone.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| transferCall | Indicates whether the subscriber can transfer all calls or only internal calls (those from within a business group). | Enum | All,<br>Internal Only | All |
| recallTimer | The amount of time to wait in seconds before calling the person who originated the call. | | 5 to 3600 sec | 50 |
| applyRecallTo | The type of calls for which intercept timer will be applied. | Enum | All,<br>Internal DN,<br>External DN,<br>None | All |
| announcementTimer | The amount of time for the intercept to start. | | 6 to 3600 sec | 70 |
| announcementDestination | The destination for the intercept.<br>The destination DN must be the same BG as the transferring DN. | | Any related subscriber's service ID | |

*Table 18 Parameters of call transfer feature*

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| applyAnnouncementTo | The type of calls for which intercept timer will be applied. | Enum | All,<br>Internal DN,<br>External DN,<br>None | None |

*Table 18 Parameters of call transfer feature*

### Distinctive Ringing for Waiting Calls Feature - *CfDRCWBean*

This feature provides a distinctive ringing tone when a caller calls the subscriber and the calling number is listed in the Public or Private list.

### Enhanced Anonymous Caller Reject Feature - *CfEACRBean*

This feature enables you to block calls from callers who have Caller ID blocking. The subscriber does not have to have Caller ID to subscribe to this service. The subscriber can choose whether the caller receives a denial announcement, is routed to voice mail, or is routed to a Privacy Director. The Privacy Director asks for the caller's name, calls the subscriber, speaks the name of the calling party, and connects the callers based upon the subscriber decision.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| subControlled | Indicates whether the subscriber can activate/ deactivate the feature. | | True, False | False |
| billing | The type of billing. | Enum | Flat Rate,<br>Usage-sensitive | Flat Rate |
| EACRDest | The call treatment a rejected caller receives. | Enum | Play Denial Announcement, Route to Voice Mail, Route to Privacy Director | Play Denial Announcement |
| voiceMailDN | The voice mail directory number.<br>If call treatment is Route to Voice Mail,then voice mail dn can be set. Also, If call treatment is Route to Privacy Director, Anonymous call screener can be set to this field. | | Character string | |

*Table 19 Parameters of enhanced anonymous caller reject feature*

### Call Forward Enhanced Feature - *CfECFBean*

This feature enables the subscriber to forward calls to another number on a subscriber-controlled schedule.

When enabled, this service overrides Call Forward Variable, Call Forward Busy, and Call Forward No Answer, depending on whether the service is active and the type of forwarding selected for that period.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| listOfScreenEntryBeans | The list of numbers to forward. | | 32 entries; up to15 digits each List of numeric strings | |
| ringDuration | Duration that the phone will ring before (if not answered) the call will be redirected to the Voice Mail server. | | 0 to 60 sec | 24 |
| listOfCFECFEntryBeans | The list of schedules. | | Each entry represents a time period for a specific day of the week. | |
| DayOfWeek | The day of the week to which the forwarding data defined in this entry applies. | Enum | Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday | Sunday |
| Schedule | The time schedule. It contains start time and stop time. At start time the forwarding data defined in this entry applies and at stop time the forwarding data defined in this entry no longer applies, in subscriber's local timezone. | | Start and stop time should be HH:MMAM/PM format and are concetaned by "-". (for example 06:00AM-07:00AM) | |
| CallFwdType | The type of call forwarding performed for this entry. | Enum | Always forward, Line is Busy, No Answer, Busy No Answer | Always forward |
| ScreeningList Option | The value indicating which calls will be forwarded. | Enum | All Numbers, Numbers on caller list, Numbers not on caller list. | All Numbers |
| ServiceId | The destination number indicates the destination to which calls are forwarded for the time specified in this entry. | | Number String | |

*Table 20 Parameters of call forward enhanced feature*

### Hot Desking Feature - *CfHotDeskBean*

This feature enables a subscriber to log on and to use a telephone in a remote office. The remote phone has all of the same features and capabilities at the subscriber's home office telephone, except those provided by the telephone itself and not the OpenScape Voice system.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| hotDeskSide | The type of hot desking side. | Enum | Remote, Home Base | Remote |

*Table 21 Parameters of hot desking feature*

### Malicious Call Trace Feature - *CfMCTBean*

This service enables the subscriber to issue a Call Trace Report to the authorities to report harassing calls.

### Music On Hold Feature - *CfMOHBean*

This feature allows the subscriber to specify a music source to be played for any held parties when they are placed on hold by the subscriber with the MOH feature. The source of the music is specified by an intercept name.

### One Number Feature - *CfONSBean*

This feature feature allows a subscriber to originate call using a device other than the subscriber's primary business telephone, while keeping the functions and features of their business telephone.

### Remote Activation Feature - *CfRACFBean*

This feature enables the subscriber to change the CFV data from a remote phone.

### Selective Caller Acceptance Feature - *CfSCABean*

This feature enables the subscriber to restrict incoming callers to an unshared list. Callers who are not on the list are forwarded to another number or hear a denial announcement.The caller can then enter a PIN and reach the subscriber. The PIN is available only when the denial is announced.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| billing | The billing type. | Enum | Flat Rate,<br>Usage-Sensitive | Flat Rate |
| pin | The personal identification number, a caller can use to be placed through to the subscriber even though they are calling from a phone that is not on the screening list. This service cannot be used unless PIN is configured. | 10 | Character String | |
| scaTerminationTreatment | The type of rejected calls, which is indicating whether a denial announcement is to be played or the call is to be redirected to another number. If the call is to be redirected, enter the phone number to which the call is to be redirected in the field. | Enum | Redirect,<br>Play Denial Announcement | Redirect |
| serviceId | The service id of the subscriber to which the call is to be redirected. | | Numeric String | |
| listOfScreenEntryBeans | The list of numbers to forward. | 32 entries; up to15 digits each | List of numeric strings | |

*Table 22 Parameters of selective caller acceptance feature*

### Selective Call Forwarding Feature - *CfSCFBean*

This service enables the subscriber to forward calls from callers who are on a subscriber-controlled list.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceId | The redirect number, which is the destination number to which calls are forward when callers are found to be in the screening list. | | Any related subscriber's service ID | |
| screenListBean | The list of numbers to forward. | 32 entries; up to15 digits each | List of numeric strings | |

*Table 23 Parameters of selective call forwardinf feature*

### Selective Caller Reject Feature - *CfSCRBean*

This feature enables the subscriber to restrict incoming callers. Callers on a subscriber-controlled list hear a denial announcement.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| billing | The billing type. | Enum | Flat Rate, Usage-Sensitive | Flat Rate |
| subControlled | Indicates whether the subscriber can activate/ deactivate this feature. | | True, False | False |
| listOfScreenEntryBeans | The list of selective caller acceptance member. | 32 entries; up to15 digits each | List of numeric strings | |

*Table 24 Parameters of selective caller reject feature*

### Serial Ringing Feature - *CfSERRNGBean*

This feature enables the subscriber to configure a list of DNs to be called if the primary DN does not respond within the specified "duration". The numbers are then dialed sequentially from the list if their status reflects "enabled". Furthermore, the duration for which these lines are dialed is also configured through a Duration parameter.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| ringDuration | The period of time the primary phone rings before the DNs in the list are dialed. | 0 to 60 seconds | Flat Rate, Usage-Sensitive | Flat Rate |
| listOfSerialDNEntryBeans | A list of up to six entries, each with the attributes listed below. | | | |
| serviceID | A subscriber's service ID to be rung serially. | | Numeric String | |
| duration | Period of time this entry's phone rings before the next DN in the list is dialed. | integer | 1 to 120 | |
| ring | Indicates that the DN is a ring. | | True, False | False |

*Table 25 Parameters of serial ringing feature*

### Speed Calling Feature - *CfSPCALLBean*

This feature enables subscribers to store frequently called numbers in the telephone and then program one or two digits to call that number when pressed.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| billing | The billing type. | Enum | Flat Rate,<br>Usage-Sensitive | Flat Rate |
| subControlled | Indicates whether the subscriber can activate/ deactivate this feature. | | True, False | False |
| oneDigit | Indicates whether the subscriber has its own list or shares the list from another entity. This field may also contain the literal subscriber to indicate that the list is shared from the Business Group. | | 0 for No List,<br>1 for Custom List,<br>subscriber's service ID for Shared List | 0 |
| twoDigit | Indicates whether the subscriber has its own list or shares the list from another entity. This field may also contain the literal subscriber to indicate that the list is shared from the Business Group. | | 0 for No List,<br>1 for Custom List,<br>subscriber's service ID for Shared List | 0 |
| oneSPCALLList | The list of one-digit calling numbers with below attributes. | | | |
| SPCALLNum | The one-digit control numbers. | Enum | 2 to 9 | |
| serviceId | The subscriber for which speed call assigned. | | Number String | |
| twoSPCALLList | The list of two-digit calling numbers with below attributes. | | | |
| SPCALLNum | The two-digit control numbers. | Enum | 20 to 49 | |
| serviceId | The subscriber for which speed call assigned. | | Number String | |

*Table 26 Parameters of speed calling feature*

### Line Restriction Feature - *CfSRBean*

This feature enables a BG Administrator to impose restriction levels for calls that originate from or terminate at a subscriber.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| originatingSR | The restriction level for calls the subscriber originates. | Enum | No Restriction, Semi-Restricted Line, Fully Restricted Line, Fully Restricted Line with Attendant Access. | No Restriction |
| terminatingSR | The restriction level for calls terminating to the subscriber. | Enum | No Restriction, Semi-Restricted Line, Fully Restricted Line, Fully Restricted Line with Attendant Access. | No Restriction |

*Table 27 Parameters of line restriction feature*

### Simultaneous Ringing Feature - *CfSRSBean*

This feature enables a subscriber to create a list of six subscribers that will ring simultaneously when the main DN receives an incoming call and is not busy. The call can be answered at the BG main number or any of the other subscribers in the list.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| listOfDNRingEntryBeans | The simultaneous ringing list that can contain up to six entries. | | | |
| serviceID | A DN to be simultaneously rung. | | Number String | |
| extension | Indicates that the DN is an extension, or intercom dialing code (True). If False, the number is not an extension, but a fully qualified national numbering plan formatted number. | | True, False | False |

*Table 28 Parameters of simultaneous ringing feature*

### Toll and Call restrictions Feature - *CfTRSBean*

This feature indicates the type of calling restrictions that are active.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| blkDomLongDist | The restriction for domestic long distance calls of the subscriber | | True, False | False |
| blkIntLongDist | The restriction for international long distance calls of the subscriber | | True, False | False |
| blkOperAssist | The restriction for operator assisted domestic long distance calls of the subscriber. | | True, False | False |
| blkOperRequest | The restriction for operator calls of the subscriber. | | True, False | False |
| blkIntOperAssist | The restriction for operator assisted international long distance calls of the subscriber. | | True, False | False |
| blkLocalDirAssist | The restriction for local directory assistance calls of the subscriber. | | True, False | False |
| restrictedLineList | The toll and call restrictions list. The subscriber cannot make calls to the numbers in this list. The list may contain partial numbers, such as 212, that prevent the caller from calling any number that starts with 212. Individual entries in this list cannot be edited. The list must be completely specified. | | List of Numeric String | |

*Table 29 Parameters toll and call restriction feature*

### Usage Sensitive Call Forward Variable Feature - *CfUSCFVBean*

This feature indicates the subscriber controllable disposition of an incoming call to the subscriber. The caller is redirected to the number specified. If this feature is enabled, the subscriber’s phone receives a "courtesy" ring when a call is forwarded. This feature is similar to the Call Forwarding Unconditional (CFV) feature except that the billing of the feature depends on usage, rather than a flat rate charge.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceId | The redirect number. | 30 | Numeric String,<br>Any related subscriber's service ID | Required when this feature is activated |
| subActivatable | The method by which the subscriber can activate/ deactivate this feature. | Enum | None (subscriber cannot activate or deactivate this feature)<br>Phone (subscriber can activate or deactivate this feature via the phone)<br>Web (subscriber can activate or deactivate this feature via the Web).<br>All (subscriber can activate or deactivate this feature via either Web or Phone) | None |
| destServiceIdSubControllable | The method of specifying redirect number by which the subscriber can change the destination of Call Forwarding.<br>This field may only be set to Phone only if SubActivatable is set to Phone or All as the subscriber can only change the destination when the feature is activated. | Enum | None (Subscriber can not change the destination for this feature).<br>Phone (subscriber can change the destination for this feature via the Phone).<br>Web (subscriber can change the destination for this feature via the Web).<br>All (subscriber can change the destination for this feature via either the Web or phone) | None |
| CFCourtesyCallBehavior | The courtesy call behavior that occurs when a subscriber updates the Destination ServiceId when this feature is activated.<br>This field can be specified only if DestServiceIdSubControllable is set to Phone or All, since there are no courtesy calls placed when the number is set via the web. | Enum | No Courtesy Call,<br>Required,<br>Not Required | No Courtesy Call |

*Table 30 Parameters of usage sensitive call forward variable feature*

### Voice Mail Feature - *CfCFVBean*

This feature forwards calls to a voice mail box if the phone is busy or there is no answer. This service cannot be subscribed if either Call Forward Busy or Call Forward No Answer features have been already subscribed.

The following table lists the parameters associated with this feature.

| Field Name | Description | Length | Range | Default Value |
|---|---|---|---|---|
| serviceId | The voice mail destination number. | 30 | Numeric String,<br>Any related subscriber's service ID | Required when this feature is activated |
| ringDuration | Indicates the duration that the phone will ring before (if not answered) the call will be redirected to the Voice Mail server. | Integer | 0 to 60 sec | |

*Table 31 Parameters of voice mail feature*

## 2.11.2 List all Feature Profiles(*listFeatureProfiles*)

You can list all Feature Profiles via **listFeatureProfiles(sessionToken, switchName, businessGroupName, fpListType)**.

The *fpListType* field is enumerated field. It describes the type of list, depending on the switch-wide or business group specific feature profiles. It is enumerated value and can be set as FP_LIST_TYPE_ALL, FP_LIST_TYPE_SYSTEM_ONLY, FP_LIST_TYPE_BG_UNION_SYSTEM or FP_LIST_TYPE_BG_ONLY values of the *ConstantFpType* class;

- To list all feature profiles, use FP_LIST_TYPE_ALL value.
- To list only switch-wide feature profiles, use FP_LIST_TYPE_SYSTEM_ONLY.
- To list feature profiles of the specific business group with switch-wide feature profiles together, use FP_LIST_TYPE_ BG_UNION_SYSTEM and set business group name.
- To list feature profiles of the specific business group, use FP_LIST_TYPE_BG_ONLY and set business group name.

Do not pass a business group as an argument to list all the feature profiles and only switch-wide feature profiles.

## 2.11.3 List all properties of a Feature Profile (*getFeatureProfile*)

With **getFeatureProfile(sessionToken, switchName, featureProfileName, businessGroupName)**, you can list all properties of a Feature Profile. You will get a bean as return value which contains all the properties.

### 2.11.4 Create Feature Profile *(createFeatureProfile)*

With **createFeatureProfile(sessionToken, switchName, inputBean)** method, you can create both switch-wide feature profile, i.e. global feature profile and feature profile for a specific business group. The input bean points to a valid feature profile bean.

When creating feature profile for a specific business group, *inputBean* expects at least values for the bean elements *businessGroupName* and *featureProfileName*. In other case, when creating global feature profile, at least feature profile name shall be set to the input bean. The services can be specified during creation or they can be set later with modify operation.

Multiple Feature Profiles for the specific business group and switch-wide can be created.

### 2.11.5 Modify a Feature Profile (*modifyFeatureProfile*)

You can modify a Feature Profile using **modifyFeatureProfile(sessionToken, switchName, inputBean)**. *inputBean* points a valid Feature Profile bean. You can either create this bean from scratch or you can first retrieve a Feature Profile using *getFeatureProfile*. For creating a Feature Profile (FP) bean, you have to set at least *businessGroupName* and *featureProfileName* in the bean.

### 2.11.6 Delete a Feature Profile (*deleteFeatureProfile*)

You can delete a feature profile with **deleteFeatureProfile(sessionToken, switchName, featureProfileName, businessGroupName)** method. The method expects a valid feature profile name and business group name.

## 2.12 DLS-Related Commands

### 2.12.1 List DLS Servers (*listDLS*)

With **listDLS(sessionToken)** method, a list of all available DLS servers can be displayed to the user. This list contains several pieces of information, including DLS Server Name, DLS ID, DLS Status and Version, DLS URL, API URL and DLS Client Session ID.

## 2.13 Switch-Related Commands

### 2.13.1 List Switches (*listSwitch*)

**listSwitch(java.lang.String sessionToken)** which returns a ListSwitchResult containing an array of SwitchDTO. The method helps as an entry point to get a switch name to pass to subsequent operations, and for standalone usage, to get switch-related information.

### 2.13.2 List Used and Total Licenses (*listLicenses*)

**listLicenses(String sessionToken, String switchName)** returns the used and total number of licenses available for a given switch name.

### 2.13.3 Retrieve the Operation Mode of a Switch (*getOperationMode*)

Returns the operation mode of 'own' and 'partner' nodes as seen by the two nodes of the switch.

The operation mode can have the following values:

| Operation Mode Values | Description |
|---|---|
| Normal | The switch is simplex or duplex with both nodes active in a cluster, or duplex with one node out of service. |
| StandAlone, StandAlonePrimary, StandAloneSecondary | The two nodes of the switch cannot communicate with each other, but both stay active on their own in switch-over mode. |
| ShuttingDown, StandAoneSynch | Cluster communication is possible again, one of the nodes is sending data to the partner and is going to reboot afterwards. |
| Unknown | This is the status of the partner node, in case there is no x-channel. So each node shows one of the modes defined above for itself and 'unknown' for the partner. |

*Table 32 Operation Modes of a Switch*

## 2.14 OpenScape Branch-Related Commands

### 2.14.1 List Available OpenScape Branch Offices (*listOpenBranches*)

To get all available OpenScape Branch Offices, **listOpenBranches(String sessionToken)** method is used. A list of Branch Offices of type BranchOfficeDTO is retrieved. Each BranchOfficeDTO may contain:

- branchOfficeName,
- branchOfficeRepEndpointName,
- branchOfficeRepEndpointIp,
- businessGroupName,
- listOfUsers,
- switchId,
- soapUrl,
- hiPath8000Name,
- hiPath8000IpAddress,
- userId,
- userPassword
- description.

# 3 API Usage Scenarios

## 3.1 Create a Business Group

For creating a Business Group you must have at least an Office Code and Directory Numbers to be associated to the new group. Therefore the sequence of API calls for creating a new Business Group is:

| API Operation | Reference |
|---|---|
| • *openSession(username, password)* | Section 2.3.1 |
| • *createOfficeCode(sessionToken, switchName, inputBean)* | Section 2.5.2 |
| • *createDirectoryNumbers(sessionToken, switchName, officeCode,startExtension, endExtension)* | Section 2.6.2 |
| • *createBusinessGroup*(*sessionToken, switchName, inputBean)* | Section 2.9.2 |
| • *closeSession(sessionToken, switchName)* | Section 2.3.2 |

## 3.2 Create a Subscriber

For creating a Subscriber you must have at least his/her name, a valid and free subscriber ID and the business group the new subscriber is associated to. Therefore the sequence for creating a new Subscriber is the following.

in case of vacant extensions available:

| API Operation | Reference |
|---|---|
| • *openSession(username, password)* | Section 2.3.1 |
| • *listVacantDirectoryNumbers(sessionToken, switchName, officeCode)* | Section 2.6.4 |
| • *createSubscriber(sessionToken, switchName, inputBean)* | Section 2.10.4 |
| • *closeSession(sessionToken, switchName)* | Section 2.3.2 |

in case of all extensions used:

| API Operation | Reference |
|---|---|
| • *openSession(username, password)* | Section 2.3.1 |
| • *createDirectoryNumbers(sessionToken, switchName, officeCode, startExtension, endExtension)* | Section 2.6.2 |
| • *createSubscriber(sessionToken, switchName, inputBean)* | Section 2.10.4 |
| • *closeSession(sessionToken, switchName)* | Section 2.3.2 |

## 3.3 Modify the Display Name of a Subscriber

For modifying a Subscriber you must have a valid subscriber ID of the Subscriber. Therefore the sequence for modifying a Subscriber is:

| API Operation | Reference |
|---|---|
| • *openSession(username, password)* | Section 2.3.1 |
| • *modifySubscriber(sessionToken, switchName, inputBean)* | Section 2.10.5 |
| • *closeSession(sessionToken, switchName)* | Section 2.3.2 |

in case Subscriber is retrieved before modification

| API Operation | Reference |
|---|---|
| • *openSession(username, password)* | Section 2.3.1 |
| • *getSubscriber(sessionToken, switchName, inputBean)* | Section 2.10.2 |
| • *modifySubscriber(sessionToken, switchName, inputBean)* | Section 2.10.5 |
| • *closeSession(sessionToken, switchName)* | Section 2.3.2 |

# 4 Using the OpenScape Voice with Java

## 4.1 Requirements

For accessing the OpenScape Voice with a Java program there are some requirements to comply with.

First of all you need a Java compiler and a runtime environment where your Java implementation can be used. For implementing web services you also need a web server. For our examples we used Sun's Java JDK and **RTE** version 5.0 and Apache's **Tomcat** web server version 5.5.

For using the SOAP protocol and accessing WSDL files we used Apache **Axis** and **JAX** APIs. To understand the example programs you should be familiar with web services and the mentioned components. All necessary libraries are included in the Axis package and have to be added to every web services project. The files are:

| | |
|---|---|
| • axis.jar | the Axis core library |
| • axis-ant.jar | definition of axis ant tasks |
| • commons-discovery.jar | Jakarta Discovery Framework |
| • commons-logging.jar | Jakarta Logging Framework |
| • jaxrpc.jar | JAX-RPC core API |
| • log4j-1.28.jar | Log4J components used by Axis |
| • saaj.jar | API for SOAP handling |
| • wsdl4j.jar | API for WSDL handling |

## 4.2 Analyzing the OpenScape Voice WSDL file with Java

### 4.2.1 Preparing the WSDL Analysis

Java has an API for representing WSDL documents in Java, called as JWSDL. JWSDL is designed to allow users to read, modify, create and re-organize WSDL documents in memory.

You can use *WSDLFactory*, *WSDLReader*, *WSDLWriter* classes to manipulate WSDL files.

First, obtain a *WSDLFactory* instance via the static *newInstance* method of *WSDLFactory.*

```
WSDLFactory factory = WSDLFactory.newInstance();
```

Once a WSDLFactory instance is obtained, *newDefinition, newWSDLReader,newWSDLWriter* methods can be invoked to create the desired object.

The following is an example of how to create a *WSDLReader* to construct a *Definition* that represents the WSDL file of DSA API.

```
WSDLReader reader           = factory.newWSDLReader();
Definition definition       = reader.readWSDL(null, "https://<yourHiPath8000>/
                              HiPath8000AssistantAPIv310/services/
                              HiPath8000AssistantAPI");
```

## 4.2.2 Showing available operations

To see what the operations defined for a particular service, we have to navigate *Definition.* Before navigating, you need an instance of *Definition* as explained in Section 4.2, “Analyzing the OpenScape Voice WSDL file with Java”. Then you can perform navigation.

```
String tns              = "urn:hipath8000-assistant-api-v310";
Service service         = def.getService(new QName(tns,
                          "HiPath8000AssistantAPIService"));
Port port               = service.getPort("HiPath8000AssistantAPI")
Binding binding         = port.getBinding();
PortType portType       = binding.getPortType();
List operations         = portType.getOperations()
Iterator opIterator     = operations.iterator()
while (opIterator.hasNext()){
   Operation operation =(Operation)opIterator.next();
    if (!operation.isUndefined()){
    System.out.println(operation.getName());
   }
}
```

When you run the above code segment appropriately, you will see the available operations as the following;

```
openSession
closeSession
createOfficeCode
deleteOfficeCode
createFeatureProfile
getFeatureProfile
getBusinessGroup
...
```

### 4.2.3 Showing operational parameters

To display the parameters of the operations, you should analyze the *Operation* class instance obtained in the previous Section 4.2.2, “Showing available operations”. *getInput()* , *getOutput()* and *getParameterOrdering()* methods of the a particular operation will help you to display these parameters as the following;

```
if (!operation.isUndefined()){
   System.out.print(operation.getName());
   System.out.println(operation.getParameterOrdering());
   System.out.println(operation.getInput());
   System.out.println(operation.getOutput());
}
```

When you run this code for the operations, you will see the results like below;

```
openSession[username, password]
Input: name=openSessionRequest
Message: name={urn:hipath8000-assistant-api-
v310}openSessionRequest
Part: name=password
typeName={http://www.w3.org/2001/XMLSchema}string
Part: name=username
typeName={http://www.w3.org/2001/XMLSchema}string
Output: name=openSessionResponse
Message: name={urn:hipath8000-assistant-api-
v310}openSessionResponse
Part: name=openSessionReturn
typeName={http://www.w3.org/2001/XMLSchema}string

createOfficeCode[sessionToken, switchName, inputBean]
Input: name=createOfficeCodeRequest
Message: name={urn:hipath8000-assistant-api-
v310}createOfficeCodeRequest
Part: name=sessionToken
typeName={http://www.w3.org/2001/XMLSchema}string
Part: name=switchName
typeName={http://www.w3.org/2001/XMLSchema}string
Part: name=inputBean
typeName={urn:hipath8000-assistant-api-v310}OfficeCodeBean
Output: name=createOfficeCodeResponse
Message: name={urn:hipath8000-assistant-api-
v310}createOfficeCodeResponse
Part: name=createOfficeCodeReturn
typeName={urn:hipath8000-assistant-api-v310}ResultStatus
```

## 4.3 Sending commands to the OpenScape Voice with Java

### 4.3.1 A simple client

To use the client functionality of Axis you have to expand your class path with the appropriate jar files provided by Axis. For accessing a web service a client needs

- an instance of the Axis classes *Service* and *Call*

```
Service service    = new Service();
Call call          = (Call) service.createCall();
```

- the URL of the web service, called endpoint

  ```
  String endpoint = "https://<yourHiPath8000>/
  HiPath8000AssistantAPIv310/services/HiPath8000AssistantAPI";
  ```

- and optional information about data types of the used parameters and returned values.

Because our first example does not use any parameter we can start to set the endpoint for the *Call* object, define which web service operation has to be called and execute the call via invoke.

```
call.setTargetEndpointAddress( new java.net.URL(endpoint));
call.setOperationName("getApiServerVersion");
String ret = (String) call.invoke( new Object[] {});
```

The invoke method expects an array of objects but as the called operation does not have any parameters the array is empty. The return value of *getApiServerVersion* is a string and can be displayed immediately. This is the whole program:

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class simpleClient {

        public static void main(String[] args) {
              try {
                    String endpoint = "https://<yourHiPath8000>/
                    HiPath8000AssistantAPIv310/services/HiPath8000AssistantAPI";
                    Service service = new Service();
                    Call call = (Call) service.createCall();
                    call.setTargetEndpointAddress( new   java.net.URL(endpoint));
                    call.setOperationName("getApiServerVersion");
```

```
                String ret = (String) call.invoke( new Object[] {});
                System.out.println("API version: " + ret);
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
        }
    }
```

## 4.3.2 A complex client

For accessing different operations of the web service, implementing complex workflow scenarios or using operations with a huge amount of parameters or return values it is very useful to work with classes representing the available operations and their in and out parameters. For our examples we chose the static way in using a set of stub classes.

These classes are generated by the Axis tool WSDL2Java with the command

```
java org.apache.axis.wsdl.WSDL2Java https://
<yourServer>:<yourPort>/HiPath8000AssistantAPIv310/services/
HiPath8000AssistantAPI?wsdl
```

With the factory class *HiPath8000AssistantAPIServiceLocator* you can get a port which implements the *HiPath8000AssistantAPI* interface.

```
port = (HiPath8000AssistantAPI)
  new HiPath8000AssistantAPIServiceLocator().
  getHiPath8000AssistantAPI(https://<yourServer>:<yourPort>/
  /HiPath8000AssistantAPIv310/services/
HiPath8000AssistantAPI);
```

This port can be used to call all the operations provided by the OpenScape Voice as e.g. retrieving the API version

```
String serverVersion = port.getApiServerVersion();
```

The whole program looks like:

```
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import hipath8000_assistant_api_v310.*;

public class complexClient_V1 {
```

*example 0-1 Example 2: complexClient_V1.java*

```
    public static void main(String[] args) {


        HiPath8000AssistantAPI port = null;

        try {

            URL url=new URL("https://<yourServer>:<yourPort>/
              HiPath8000AssistantAPIv310/services/
              HiPath8000AssistantAPI");

            port = (HiPath8000AssistantAPI)
              new HiPath8000AssistantAPIServiceLocator().
              getHiPath8000AssistantAPI(url);

            String serverVersion = port.getApiServerVersion();

            System.out.println("Api server version: " + serverVersion);

        }

        catch (ServiceException e) {

            e.printStackTrace();

        }

        catch (MalformedURLException e) {

            e.printStackTrace();

        }

        catch (RemoteException e) {

            e.printStackTrace();

        }

    }

}
```

*example 0-1 Example 2: complexClient_V1.java*

Instead of only showing the used API version we now want to change a subscriber's display name.

Because the subscriber-related operations can be used by authorized users only, you have to request a session token via *openSession*() before sending the request and after completing all tasks you should finish the session by closing it. So you have to wrap all operations with a session handling. Session token methods are called same as other methods;

```
String sessionToken = port.openSession("myUser",
"myUsersPassword"); port.closeSession(sessionToken);
```

As stated in Section 2.9.6, "List the Subscribers of a Business Group (listBusinessGroupSubscribers)", you can either create the input bean from scratch or retrieve related subscriber using *getSubscriber* method.

```
GetSubscriberResult res =
        port.getSubscriber(sessionToken,switchName,serviceId)
;
SubscriberBean subscriber= result.getSubscriberBean();
subscriber.setDisplayName("newDisplayName");
port.modifySubscriber(sessionToken,switchName,subscriber);
```

Here you can see the complete code;

```
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import hipath8000_assistant_api_v310.*;


public class complexClient_V2 {


     public static void main(String[] args) {


          HiPath8000AssistantAPI port = null;
          String sessionToken="";
          String serviceId="";
          GetSubscriberResult result=null;
          try {
               URL url=new URL("https://<yourServer>:<yourPort>/
                 HiPath8000AssistantAPIv310/services/
                 HiPath8000AssistantAPI");
               port = (HiPath8000AssistantAPI)
                 new HiPath8000AssistantAPIServiceLocator().
                 getHiPath8000AssistantAPI(url);
               sessionToken = port.openSession("myUser",   "myUsersPassword")
               result = port.getSubscriber(sessionToken, switchName, serviceId)
               SubscriberBean subscriber=result.getSubscriberBean();
               subscriber.setDisplayName("newDisplayName");
               port.modifySubscriber(sessionToken,switchName,subscriber);
               port.closeSession(sessionToken);
          };
          catch (ServiceException e) {
               e.printStackTrace();
          }
          catch (MalformedURLException e) {
```

*Table 1 Example 3: complexClient_V2.java*

```
                e.printStackTrace();
            }
            catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }
```

*Table 1 Example 3: complexClient_V2.java*

## 4.4 Secure Access to OpenScape Voice API

The OpenScape Voice can be accessed by https protocol as well. The OpenScape Voice WSDL file can be found and viewed at the URL.

```
https://<yourHiPath8000>:<yourPort>/
HiPath8000AssistantAPIv310/services/
HiPath8000AssistantAPI?wsdl
```

Also, all available operations can be displayed at the URL

```
https://<yourHiPath8000>:<yourPort>/
HiPath8000AssistantAPIv310/services
```

To be able to connect API, you should set some system properties after you created valid keystore and truststore files for your server and client. Additionally, you should put the server keystore files under tomcat conf directory and client files under a specific directory. For detailed information about keystores, please see http://java.sun.com/docs/books/tutorial/security1.2/index.html.

```
System.setProperty("javax.net.ssl.trustStore",
<yourTrustStorePath>);

System.setProperty("javax.net.ssl.trustStorePassword",
<yourTrustStorePassword>);

System.setProperty("javax.net.ssl.keyStore",
<yourKeyStorePath>);

System.setProperty("javax.net.ssl.keyStorePassword",
<yourKeyStorePassword>);
```

Afterwards, you can call openSession method to open a session.

The whole program looks like:

```
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
```

*Table 2 Example 2: secureConnectClient_V1.java*

```
import hipath8000_assistant_api_v310.*;


public class secureConnectClient_V1 {


     public static void main(String[] args) {


          HiPath8000AssistantAPI port = null;

          try {

               System.setProperty("javax.net.ssl.trustStore", <yourTrustStorePath>);

               System.setProperty("javax.net.ssl.trustStorePassword",
               <yourTrustStorePassword>);

               System.setProperty("javax.net.ssl.keyStore", <yourKeyStorePath>);

               System.setProperty("javax.net.ssl.keyStorePassword",
               <yourKeyStorePassword>);

               URL url=new URL("https://<yourServer>:<yourPort>/
                 HiPath8000AssistantAPIv310/services/
                 HiPath8000AssistantAPI");

               port = (HiPath8000AssistantAPI)
                 new HiPath8000AssistantAPIServiceLocator().
                 getHiPath8000AssistantAPI(url);

               sessionToken = port.openSession("myUser",   "myUsersPassword")

          catch (ServiceException e) {

               e.printStackTrace();

          }

          catch (MalformedURLException e) {

               e.printStackTrace();

          }

          catch (RemoteException e) {

               e.printStackTrace();

          }

     }

}
```

*Table 2 Example 2: secureConnectClient_V1.java*

# 5 Using the OpenScape Voice with PHP

## 5.1 Requirements

For accessing the OpenScape Voice with a PHP (Hypertext Preprocessor) script there are some requirements to comply with.

First of all you need a PHP implementation and an environment where this PHP implementation can be used. The most common package is the **XAMPP** distribution, which we used to develop our PHP examples.

As PHP does not support the SOAP protocol and WSDL files directly, you need an appropriate add-on. We have tested the **NuSOAP** open source package (also written in PHP) and it works fine for this purpose. There may be other packages available which are not evaluated yet.

To become familiar with this package read the tutorial 'Introduction to **NuSOAP**'.

## 5.2 Analyzing the OpenScape Voice WSDL file with PHP

### 5.2.1 Preparing the WSDL analysis

In order to use the NuSOAP package we copied the nusoap.php file into the same directory where our test scripts are. If you use another directory you have to adapt the require statement or use an appropriate include path for your PHP environment.

```
require_once('nusoap.php');
```

Create a new instance of the wsdl class, the sole parameter indicates the URL of your WSDL file:

```
$wsdl = new wsdl('https://<yourServer>:<yourPort>/'.
          'HiPath8000AssistantAPIv<version>/services/'.
          'HiPath8000AssistantAPI?wsdl');
          // change <yourServer> to your server's name or IP address
          // change <version> to the current used API version (e.g. 310)
```

If that action fails you will receive an error message:

```
$err = $wsdl->getError();
if ($err) {
   // Display the error
```

```
    echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
    // At this point, you know the call that follows will fail
    // and you can exit your script
    exit();
}
```

In creating that WSDL instance the WSDL file is parsed and analyzed. The WSDL class now knows all about the offered web services. To show this information you need to use the *getOperations* and *getTypeDef* methods of that class. Both methods return an associative array which can be parsed with nested foreach loops, in a recursive form or you can access each element directly if you know the data structure.

## 5.2.2 Showing available operations

For analyzing purpose we choose the former possibility *getOperations*:

```
$operations = array();
$operations = $wsdl->getOperations();
// perhaps you want to show all available methods in a sorted
way
asort($operations);
foreach ($operations as $message => $msgvalue) {
    echo("Operation <b>$message</b><br />");
}
```

Now you can start the script to have a look at the available operations. You will see something like this:

```
Operation closeSession
Operation createBusinessGroup
Operation createDirectoryNumbers
Operation createFeatureProfile
Operation createOfficeCode
Operation createSubscriber
Operation deleteBusinessGroup
Operation deleteDirectoryNumber
Operation deleteFeatureProfile
Operation deleteOfficeCode
Operation deleteSubscriber
...
```

Following the complete script, it is really short:

*Table 3 Example 1: analyze_wsdl_V1.php*

```
<?php
    require_once('nusoap.php');
    $wsdl = new wsdl('https://<yourServer>:<yourPort>/'.
        'HiPath8000AssistantAPIv<version>/services/'.
```

```
        'HiPath8000AssistantAPI?wsdl');
    $err = $wsdl->getError()
    if ($err) {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        exit();
    }
    $operations = array();
    $operations = $wsdl->getOperations();
    asort($operations);
    foreach ($operations as $message => $msgvalue) {
    echo("Operation <b>$message</b><br />");
    }
?>
```

### 5.2.3 Showing operational parameters

Perhaps you now want to know which parameters are necessary to use these operations. Therefore we have to analyze the $operations array. Because this array represents a dynamical structure we write a recursive function showing the content of a dynamical array and change the foreach loop into a call of this new function.
This is the recursive function, showing dynamical data:

```
function showDynData($level, $var) {
    $ofs = '';
    for ($i=0; $i<$level; $i++) $ofs .= '-';
    if (gettype($var) == 'array') {
          foreach ($var as $key => $value) {
                if (gettype($value) == 'array') {
                      if ($level == 0) echo("<h3>$key</h3>");
                      else echo("$ofs <b>$key</b><br />");
                      showDynData($level+1, $value);
                }
                else
                      echo("$ofs $key: $value<br />");
          }
    }
    else
          echo("$ofs $var<br />");
```

```
}
```

and calling this function results in the new script version:

```
<?php
   require_once('nusoap.php');

   function showDynData($level, $var) {
      $ofs = '';
      for ($i=0; $i<$level; $i++) $ofs .= '-';
      if (gettype($var) == 'array') {
            foreach ($var as $key => $value) {
                  if (gettype($value) == 'array') {
                        if ($level == 0) echo("<h3>$key</h3>");
                        else echo("$ofs <b>$key</b><br />");
                        showDynData($level+1, $value);
                  }
                  else
                        echo("$ofs $key: $value<br />");
            }
      }
      else
            echo("$ofs $var<br />");
   }
   $wsdl = new wsdl('https://<yourServer>:<yourPort>/'.
      'HiPath8000AssistantAPIv<version>/services/'.
      'HiPath8000AssistantAPI?wsdl');
   $err = $wsdl->getError();
   if ($err) {
      echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
      exit();
   }
   $operations = array();
   $operations = $wsdl->getOperations();
   asort($operations);
```

*Table 4 Example 2: analyze_wsdl_V2.php*

```
    showDynData(0, $operations);

?>
```

*Table 4 Example 2: analyze_wsdl_V2.php*

Running this new script version shows you the complete content of the dynamical $operations array and you are able to examine its structure and the name of all array indices to address its elements directly.

**closeSession**

```
- name: closeSession
- binding: HiPath8000AssistantAPISoapBinding
- endpoint: https://139.21.215.77:8080/
HiPath8000AssistantAPIv310/services/HiPath8000AssistantAPI
- soapAction:
- input
-- encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
-- namespace: urn:hipath8000-assistant-api-v310
-- use: encoded
-- message: closeSessionRequest
-- parts
--- sessionToken: http://www.w3.org/2001/XMLSchema:string
- output
-- encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
-- namespace: urn:hipath8000-assistant-api-v310
-- use: encoded
-- message: closeSessionResponse
-- parts
- style: rpc
- transport: http://schemas.xmlsoap.org/soap/http
- documentation:
```

```
createFeatureProfile
```

```
- name: createFeatureProfile
- binding: HiPath8000AssistantAPISoapBinding
- endpoint: https://139.21.215.77:8080/
HiPath8000AssistantAPIv310/services/HiPath8000AssistantAPI
- soapAction:
- input
-- encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
-- namespace: urn:hipath8000-assistant-api-v310
-- use: encoded
-- message: createFeatureProfileRequest
-- parts
--- sessionToken: http://www.w3.org/2001/XMLSchema:string
--- switchName: http://www.w3.org/2001/XMLSchema:string
--- inputBean: urn:hipath8000-assistant-api-
v310:FeatureProfileBean
- output
-- encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
-- namespace: urn:hipath8000-assistant-api-v310
-- use: encoded
-- message: createFeatureProfileResponse
-- parts
--- createFeatureProfileReturn: urn:hipath8000-assistant-api-
v310:ResultStatus
- style: rpc
```

```
    - transport: http://schemas.xmlsoap.org/soap/http
    - documentation:
```

In analyzing this shortened output example you find all necessary input parameters and their types. The operation ’closeSession’ has only one string parameter called sessionToken but the ’getFeatureProfile’ operation has an additional one with an API specific type. How to analyze this new type? Its as easy as all things before.

## 5.2.4 Showing WSDL specific types

Using the wsdl class’ method getTypeDef also returns a dynamical array representing the necessary data structure. So we can use the showDynData function the next time within itself.

```
function showDynData($level, $var) {
   global $wsdl;
   $ofs = '';
   for ($i=0; $i<$level; $i++) $ofs .= '-';
   if (gettype($var) == 'array') {
         foreach ($var as $key => $value) {
               if (gettype($value) == 'array') {
                     if ($level == 0) echo("<h3>$key</h3>");
                     else echo("$ofs <b>$key</b><br />");
                     showDynData($level+1, $value);
               }
               else {
                     if ($key != 'namespace'
                           and
                          substr($value,0,28)
                           == 'urn:hipath8000-assistant-api') {
                           $colonpos = strrpos($value,':');
                           $apiType = $wsdl->getTypeDef(substr($value,
                                 $colonpos+1),substr($value,0,$colonpos));
                           echo("<table border=\"1\">
                                 <tr><td style=\"vertical-align:top\">
                                       $ofs $key: </td><td>$value<br />");
                           showDynData(1,$apiType);
                           echo("</td></tr></table>");
                     }
                     else
                           echo("$ofs $key: $value<br />");
```

```
                }
            }
    }
    else
            echo("$ofs $var<br />");
}
```

Replacing this function leads to the 3rd example analyze_wsdl_V3.php which now shows the API types as well, see the appropriate extract:

```
createOfficeCode
- name: createOfficeCode
- binding: HiPath8000AssistantAPISoapBinding
- endpoint: https://139.21.215.77:8080/
HiPath8000AssistantAPIv310/services/HiPath8000AssistantAPI
- soapAction:
- input
-- encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
-- namespace: urn:hipath8000-assistant-api-v310
-- use: encoded
-- message: createOfficeCodeRequest
-- parts
--- sessionToken: http://www.w3.org/2001/XMLSchema:string
--- switchName: http://www.w3.org/2001/XMLSchema:string
--- inputBean:  urn:hipath8000-assistant-api-
v310:OfficeCodeBean
- name: OfficeCodeBean
- typeClass: complexType
- phpType: struct
- compositor: sequence
- elements
-- areaCode
--- name: areaCode
--- nillable: true
--- type: http://www.w3.org/2001/XMLSchema:string
--- form: unqualified
-- countryCode
--- name: countryCode
--- nillable: true
--- type: http://www.w3.org/2001/XMLSchema:string
--- form: unqualified
-- locationCode
--- name: locationCode
--- nillable: true
--- type: http://www.w3.org/2001/XMLSchema:string
--- form: unqualified
- output
-- encodingStyle: http://schemas.xmlsoap.org/soap/encoding/
-- namespace: urn:hipath8000-assistant-api-v310
-- use: encoded
-- message: createOfficeCodeResponse
-- parts
...
```

## 5.3 Sending commands to the OpenScape Voice with PHP

### 5.3.1 Preparing the communication

For using the operations defined in the WSDL file you need an instance of a soap client. If you know the operations you want to use and the data structure of their parameters you don’t need an instance of the wsdl class of your own.

As in the first example you need to include the NuSOAP classes.

```
require_once('nusoap.php');
```

From there you have to instantiate the soap client.

```
$wsdlUrl ='https://localhost:<yourPort>/
HiPath8000AssistantAPIv310/services/’.
   ’HiPath8000AssistantAPI?wsdl';
$client = new soapclient( $wsdlUrl, true);
// the first parameter assigns the URL to the web service or
WSDL file
// the second one tells the system that we use a WSDL file
```

As already seen in the first example we should check if the constructor succeeded.

```
$err = $client->getError();
if ($err) {
   // Display the error
   echo("<h2>Constructor error</h2><pre>" . $err . "</pre>");
   // At this point, you know the call that follows will fail
   exit();
}
```

As seen in the tutorial, it is easier to work with a proxy, so we create one.

```
$proxy = $client->getProxy();
```

### 5.3.2 Sending a simple command

Now all is prepared to send commands to the OpenScape Voice or via this API to the OpenScape Voice. Let’s look what API version the server runs.

```
$response = $proxy->getApiServerVersionString();
```

First we now have to check if this command succeeded.

```
if ($proxy->fault) {
        echo '<h2>Fault</h2><pre>';
```

```
        print_r($response);
        echo '</pre>';
        exit();
    } else {
        // Check for errors
        $err = $proxy->getError();
        if ($err) {
                echo '<h2>Error</h2><pre>' . $err . '</pre>';
                exit();
        }
    }
```

If there was no fault we have received the running version and we can show it.

```
echo("you are working with API version $resproxy<br />");
```

That's it. Here is the code again in a complete form.

```
<?php
    require_once('nusoap.php');
    $wsdlUrl ='https://localhost:<yourPort>/HiPath8000AssistantAPIv310/services/'
           .'HiPath8000AssistantAPI?wsdl';
    $client = new soapclient( $wsdlUrl, true);
    $err = $client->getError();
    if ($err) {
           echo("<h2>Constructor error</h2><pre>" . $err . "</pre>");
           exit();
    }
    $proxy = $client->getProxy();
    $response = $proxy->getApiServerVersion();
    if ($proxy->fault) {
           echo '<h2>Fault</h2><pre>';
           print_r($resproxy);
           echo '</pre>';
           exit();
    } else {
           $err = $proxy->getError();
           if ($err) {
                  echo '<h2>Error</h2><pre>' . $err . '</pre>';
```

*Table 5 Example 3: send_operation_V1.php*

```
                exit();
        }
    }
    echo("you are working with API version $response<br />");
?>
```

*Table 5 Example 3: send_operation_V1.php*

The output of this script is:

```
you are working with API version 3.1
```

This is a simple example because there is no parameter needed and the returning result is of a scalar type. The next example will show you a more complex operation.

### 5.3.3 Sending a complex command

The preparation for sending complex commands is the same as already described in Section 5.3.1, “Preparing the communication”. Adding parameters to an operation is as easy as to add them to functions or methods. You have to use $proxy->operationName(param1, param2, param3);

You can call listBusinessGroups method as the following;

```
$response = $proxy-
>listBusinessGroups('mySessionToken','mySwitchName');
```

Of course you should check the success of this call (see Section 5.3.2, “Sending a simple command”). If the operation succeeded you can evaluate its response. As already seen the response can be displayed directly. Last time the result was a simple string but now the result is a structure of an API specific type, handled by NuSOAP and PHP with an associative array. For the display of such arrays we have defined the function showDynData (see Section 5.2.3, “Showing operational parameters”), which we now use for presenting the result.

Because the API operation of our example can be used by authorized users only, you have to request a session token via *openSession()* before sending the request and after completing all tasks you should finish the session by closing it. So you have to wrap all operations with a session handling.

```
$username = 'myUser';
$password = 'myUsersPassword';
$sessionToken = $proxy->openSession($username, $password);
// do all necessary operations
$proxy->closeSession($sessionToken);
```

Here you can see the complete script:

```
<?php
   require_once('nusoap.php');

   function showDynData($level, $var) {
      $ofs = '';
      for ($i=0; $i<$level; $i++) $ofs .= '-';
      if (gettype($var) == 'array') {
            foreach ($var as $key => $value) {
                  if (gettype($value) == 'array') {
                        if ($level == 0) echo("<h3>$key</h3>");
                        else echo("$ofs <b>$key</b><br />");
                        showDynData($level+1, $value);
                  }
                  else
                        echo("$ofs $key: $value<br />");
            }
      }
      else
            echo("$ofs $var<br />");
   }

   function checkResponse ($response) {
      global $proxy;
         if ($proxy->fault) {
            echo '<h2>Fault</h2><pre>';
            print_r($response);
            echo '</pre>';
            exit();
         } else {
            $err = $proxy->getError();
            if ($err) {
                  echo '<h2>Error</h2><pre>' . $err . '</pre>';
                  exit();
            }
         }
```

*Table 6 Example 4: send_operation_V2.php*

```
}

$wsdlUrl ='https://localhost:<yourPort>/HiPath8000AssistantAPIv310/services/'
   'HiPath8000AssistantAPI?wsdl'; // change to your WSDL's URL
$client = new soapclient( $wsdlUrl, true);
// the first parameter assigns the URL to the web service or WSDL file
// the second one tells the system that we use a WSDL file

$err = $client->getError();
if ($err) {
   echo("<h2>Constructor error</h2><pre>" . $err . "</pre>");
   exit();
}

$proxy = $client->getProxy();

$username = 'myUser'; // has to be changed to your user
$password = 'myUsersPassword'; // change to your user's password
$switchName = 'mySwitch'; // change to your user's password
$response = $proxy->openSession($username, $password);
checkResponse($response);

$sessionToken = $response;

$response = $proxy->listBusinessGroups($sessionToken,$switchName);
checkResponse($response);

showDynData(0, $response);

$proxy->closeSession($sessionToken);
?>
```

*Table 6 Example 4: send_operation_V2.php*

If there are data available this operation results in an output like that:

```
listdisplay
- 0
-- areaCode:
-- countryCode:
-- defaultFeatureProfile:
```

```
-- displayNumber: 4351701
-- localOfficeCode:
-- name: BGstier01
-- numberSubscribers: 2
- 1
-- areaCode:
-- countryCode:
-- defaultFeatureProfile:
-- displayNumber: 4351702
-- localOfficeCode:
-- name: BGstier02
-- numberSubscribers: 0
- 2
-- areaCode:
-- countryCode:
-- defaultFeatureProfile:
-- displayNumber: 4351706
-- localOfficeCode:
-- name: BGstier06
-- numberSubscribers: 0
- 3
-- areaCode:
-- countryCode:
-- defaultFeatureProfile:
-- displayNumber: 4351707
-- localOfficeCode:
-- name: BGstier07
-- numberSubscribers: 2

...
resultStatus
- commandReplies:
- currentState:
- lastJobResultHashCode: 0
```

# Index

## M



## N



## O



## P



## R



## S



## T



## U



## V



## W