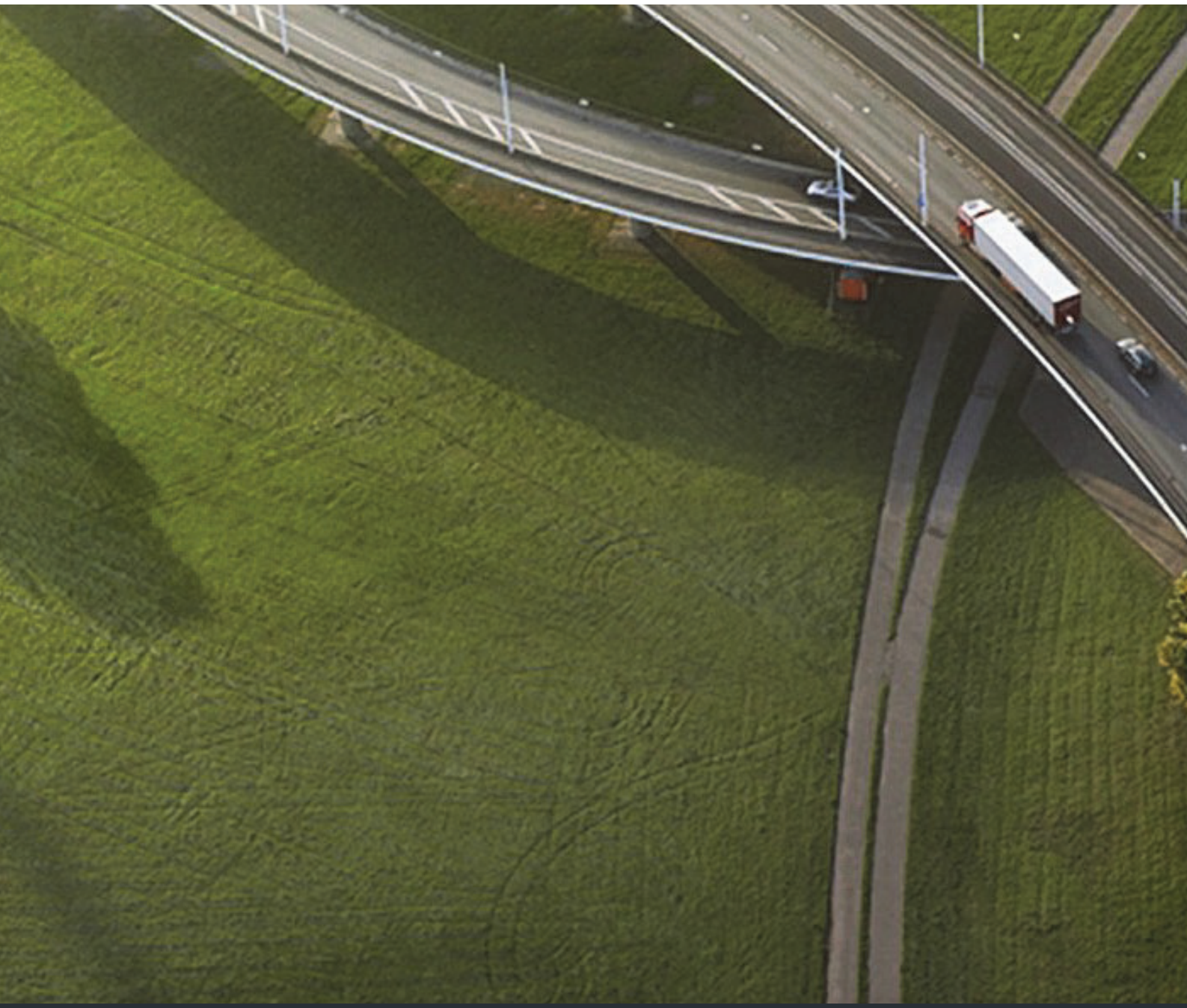
UNIFY Harmonize your enterprise



OpenScape Voice V8
Application Developers Manual

Programming Guide

A31003-H8080-R100-2-7620

Our Quality and Environmental Management Systems are implemented according to the requirements of the ISO9001 and ISO14001 standards and are certified by an external certification company.

Reference No.: A31003-H8080-R100-2-7620

unify Harmonize your enterprise

unify.com

Contents

1 Important Notices

1.1 About This Book

This manual provides overview and programming information for application development working with the OpenScape Voice V4 and later released product.

1.1.1 Prerequisite Knowledge

Users should have completed the appropriate technical and product training for developing applications working with the OpenScape Voice V4 and later releases.

These manual is intended for use by application developers working with the OpenScape Voice V4 and later release product.

1.1.2 How to Use This Book

Use this manual to complete application development programming tasks desired for the OpenScape Voice V4 and later releases.

1.1.3 Special Notices

If applicable, potentially dangerous situations are noted throughout this guide. The three alert methods are defined below:

DANGER	A danger notice calls attention to conditions that, if not avoided, will result in death or serious injury.
WARNING	A warning notice calls attention to conditions that, if not avoided, could result in death or serious injury.
Caution	A caution notice calls attention to conditions that, if not avoided, may damage or destroy hardware or software.

1.2 Documentation Feedback

When you call or write, be sure to include the following information. This will help identify which document you are having problems with.

- **Title:** OpenScape Voice V8, Application Developers Manual, Programming Guide

- **Order Number:** A31003-H8080-R100-2-7620

1.2.1 For U.S. only

To report a problem with this document, call your next level of support:

- Customers should call the Unify Customer Support Center.
- Unify Software and Solutions GmbH & Co KG employees should call the Interactive Custor Engagement Team (i-CET) or use the Document Feedback form that you can access from the front page of the HTML version of this document.

1.2.2 Countries other than U.S.

Please provide feedback on this document as follows:

- Submit a trouble ticket to ICTS, or
- Use the Document Feedback form that you can access from the front page of the HTML version of this document.

2 The Application Developers Manual

The OpenScape Voice V8 Application Developers Manual is subdivided into the following sections.

- Web Services SDK Programming Overview
- Link Failure Management
- Business Group Management
- Subscriber Self-Care

3 About the OpenScape Voice Web Services SDKs

The OpenScape Voice Web Services SDK Programming Overview provides application development programming information that is common to all SDKs, as well as an overview of each SDK.

3.1 Architecture

Bild 1 shows the generic architecture of the Unify Web Services SDKs.

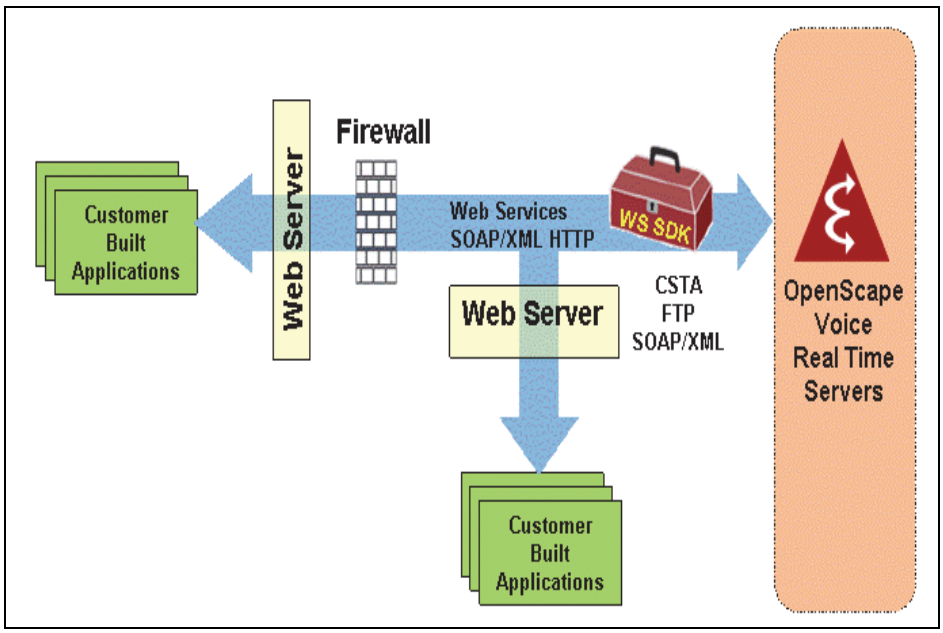


Figure 1 Web Services SDK General Architecture

Each SDK exposes a Web Service Interface. This interface comes from the OpenScape Voice system Real Time Servers.

Customers then build applications that access these Unify provided Web Services. Using the functionality of the Web Services, customers build custom applications to accomplish various tasks.

Because these SDKs are provided as Web Services, they are accessible by Web Servers and custom applications that reside outside the firewall. While this deployment is the most likely scenario, there is no reason the applications and Web Server could not reside within the firewall as well.

There may be numerous customer applications and numerous Web Servers.

For the various SDKs, the Unify SDK Servers host both the Web Service Interface and the Web Service Execution layer.

Attention: For most customers, the SDK Server referenced in this guide will be the OpenScape Voice system. Talk to your Unify representative for details.

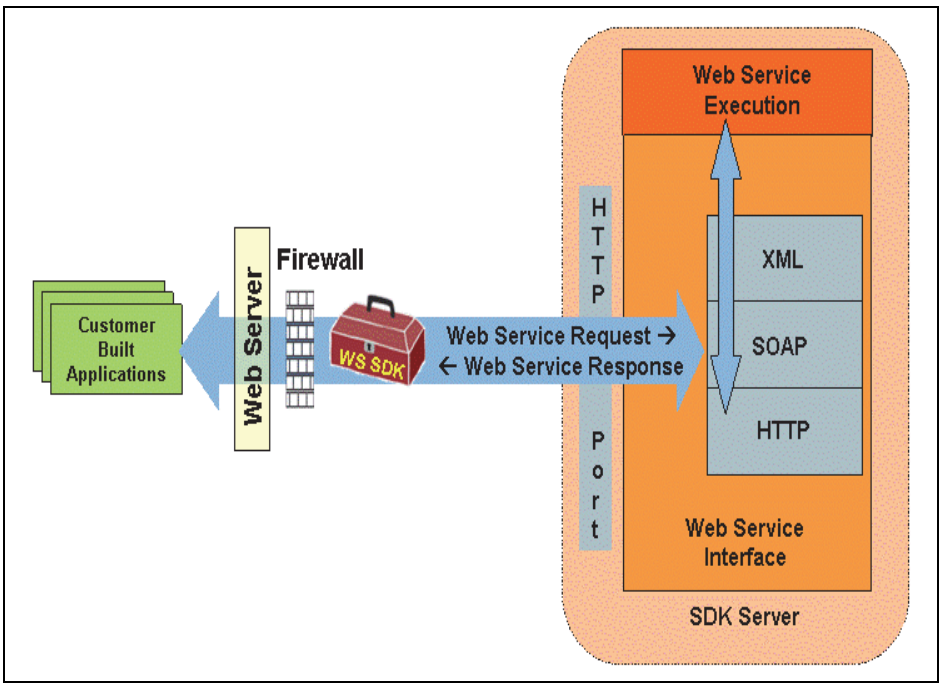


Figure 2 *Communication Between Applications and SDK Servers via Web Services*

Communications between the applications and the SDK Servers are via Web Services. These web services communicate via SOAP/XML over HTTP using the HTTP port of the SDK Server that is hosting the Web Service Interface implementation.

When an HTTP POST message is received, the SOAP envelope is extracted from the HTTP message. Next the XML data is extracted form the SOAP envelope. At this point the XML message is interpreted by the Web Service Execution layer and the appropriate actions are taken.

The result of executing the request (the response) follows the reverse path back to the application. It is given to the Web Service Interface layer to be packaged into XML, stuffed into a SOAP envelope, and then transported back in an HTTP response message.

Once a customer is ready to develop an application, the SDK WSDL files can be run through any number of industry-standard tools to create proxy files in the desired target language; for example, C++ or Java. Unify recommended environments include the SourceForge gSOAP and Apache Axis environments, but any tool that uses WSDL files can be tried.

Once a set of files has been created in the target language, the programmer uses knowledge obtained from this SDK document and sample applications to create his own custom applications. IMS resources such as forums and newsgroups can be used to get answers to specific questions and/or see answers to common problems.

The final result is a working, custom-built application that uses the specified SDK.

3.2 SDK Package Contents

A Web Services SDK is made up of a set of components and tools that the customer uses to create custom applications. Typically, an SDK will consist of:

- The WSDL file
- An ADG, which includes sample code
- The interface specification, provided in Interface Manual: Volume 3, SOAP/ XML Subscriber Interface Provisioning.

3.3 Applications Development

The target applications development environments for the Link Failure Management SDK are any environments that support C++ or Java applications. Two good tools for generating header and source files from WSDL are SourceForge gSOAP and Apache Axis. It should be noted, however, that environmental and tool differences do arise, and each different environment may require some unique solutions. There are no requirements as to which Web Server is used.

3.3.1 SourceForge gSOAP

gSOAP is an open source SOAP toolkit that facilitates Web Services development in a C/C++ environment. Within this environment is a tool called wsdl2h which will take a WSDL file as input and create the C/C++ header and code files needed to call the Web Services defined by that WSDL file. Detailed information on SourceForge gSOAP can be found at: http://gsoap2.sourceforge.net/.

3.3.2 Apache Axis

Axis is the Apache implementation that supports the SOAP standard as defined by W3C. Within this environment is a tool called wsdl2java which will take a WSDL file as input and create the java code needed to call the Web Services defined by that WSDL file.

Detailed information on Apache Axis can be found at: http://ws.apache.org/axis/.

3.4 Technical Implementation Notes

The following sections describe the technical implementation common to all SDKs.

3.4.1 Discovery of Web Services

There is no explicit discovery mechanism for Web Services SDKs. Web Services SDKs are accessed by using the IP address of the SDK Server that hosts the SOAP server and the HTTP port.

3.4.2 Web Service Request/Response

A Web Service request/response consists of an SOAP/XML-encoded request and response that is transported by HTTP, within the HTTP POST and HTTP 200 OK messages. Each protocol identified as follows: HTTP is in plain text, SOAP is underlined, and *XML is in italics*.

3.4.3 Request-Response Operation

All Unify SDKs currently support only a request-response paradigm. In this paradigm, a request is made and a response specific to that request is returned. The basic flow of the request-response model is as follows, with key components highlighted in **bold**:

1. The **Customer-Built Application** makes a Web Service request.
2. The Web Service request is processed by the **Web Server** and the proper SOAP/XML request is sent through the **Firewall** to the **Web Service Interface** on the correct **Unify SDK Server** via an HTTP POST message.
3. The **Web Service Interface** receives the request, parses the SOAP/XML content, and passes it to the **Web Service Execution** layer for execution.

4. The **Web Service Execution** layer executes the request, formulates the response, and gives it to the **Web Service Interface** layer for encoding and packaging.
5. The **Web Service Interface** layer encodes a SOAP/XML message and sends it back through the **Firewall** to the **Web Server** via an HTTP 200 OK message.
6. The **Web Server** forwards the response to the **Customer-Built Application**, which can then continue its processing.

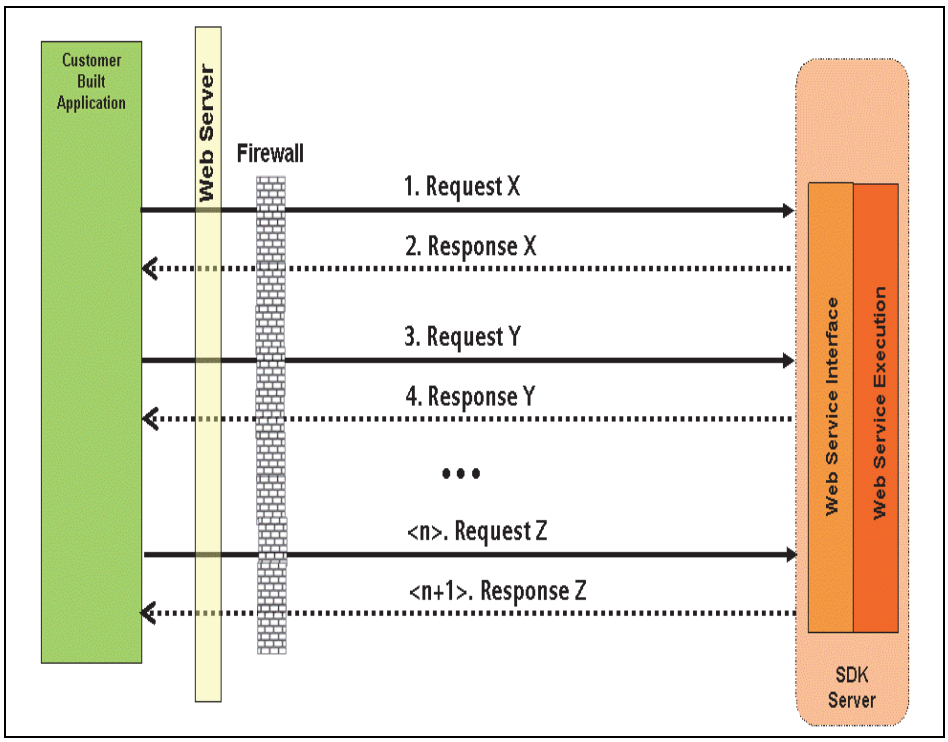


Figure 3 Request-Response Operation

Responses are not correlated to requests with any type of request ID, and as such an application must wait for a response before sending the next request. In other words, on a single thread of the application, no more than one single request can be outstanding at one time.

The Unify Web Services SDKs do not support any kind of monitoring or subscribe-notify paradigm where asynchronous event flow is required.

If a response is not returned within a certain timeframe, the HTTP request will time out and an error will be returned to the application.

3.4.4 Security

The Unify Web Services SDKs do not provide any general mechanisms for security common to all SDKs. Authentication, authorization, and other security mechanisms will be provided on a per-SDK basis as deemed necessary. Refer to the specific SDK documentation for details.

3.4.5 Sessions

With respect to the application context, the Unify Web Services SDKs are sessionless.

3.4.6 Server Restart

There will be no specific indication to the application of a server restart. The only indication an application might see is the timeout of a current request.

3.4.7 Application Restart

The Unify SDK Servers will have no indication of an application restart.

3.4.8 Heartbeat Mechanisms

The Unify SDKs do not support any common heartbeat messages.

3.4.9 Error Handling

Responses may return specific errors relating to a specific request. There is no overall general error message scheme amongst all SDKs. For specific errors, refer to the supporting documentation of each SDK.

Under certain conditions, a Web Service request may time out. This general condition may indicate failure in the network, failure on the server, etc. It does not point to one specific failure.

SOAP and XML processing errors may also be returned. They indicate badly formed SOAP or XML messages, which typically indicate a customer application problem.

4 Description of Web Services SDKs

This chapter describes the set of SDKs that are currently provided and supported by Unify for the OpenScape Voice environment.

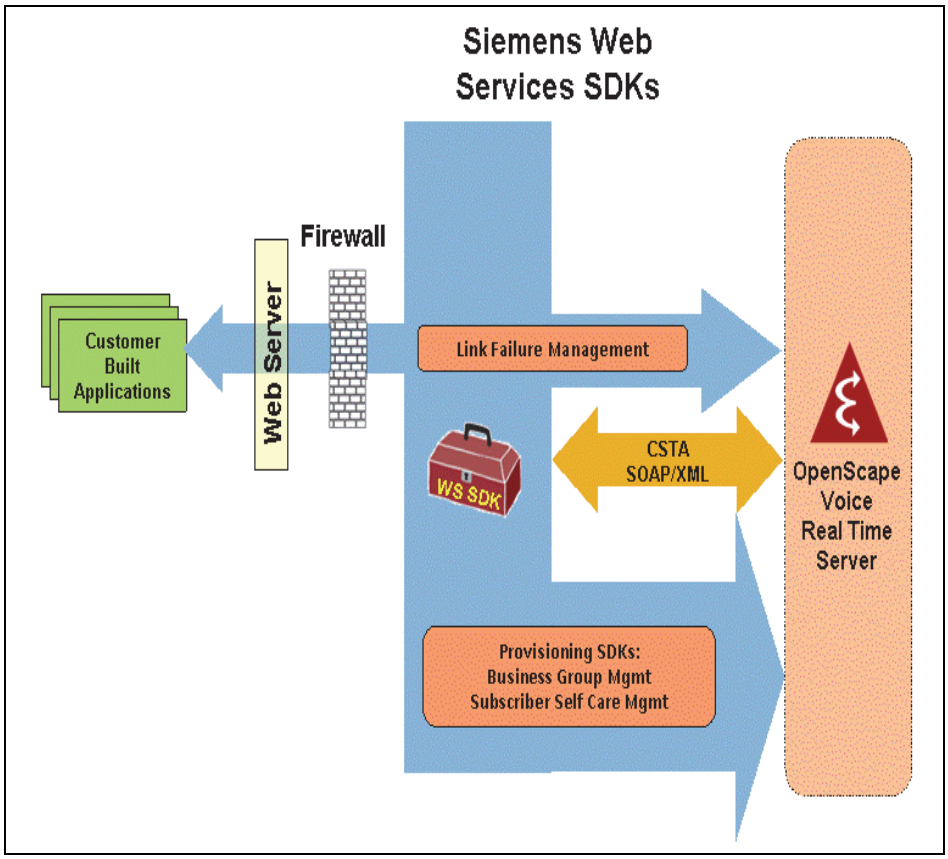


Figure 4 Current Set of Unify WS SDKs for OpenScape Voice

There are several SDKs provided by the OpenScape Voice system Real Time Server. Bild 4 shows which SDKs are exposed by which components.

Taken together, these SDKs make up the Web Services SDKs supported by Unify for the OpenScape Voice system.

4.1 Link Failure Management SDK on the OpenScape Voice

The following SDK is provided by the OpenScape Voice system real time server to facilitate the management of link failures.

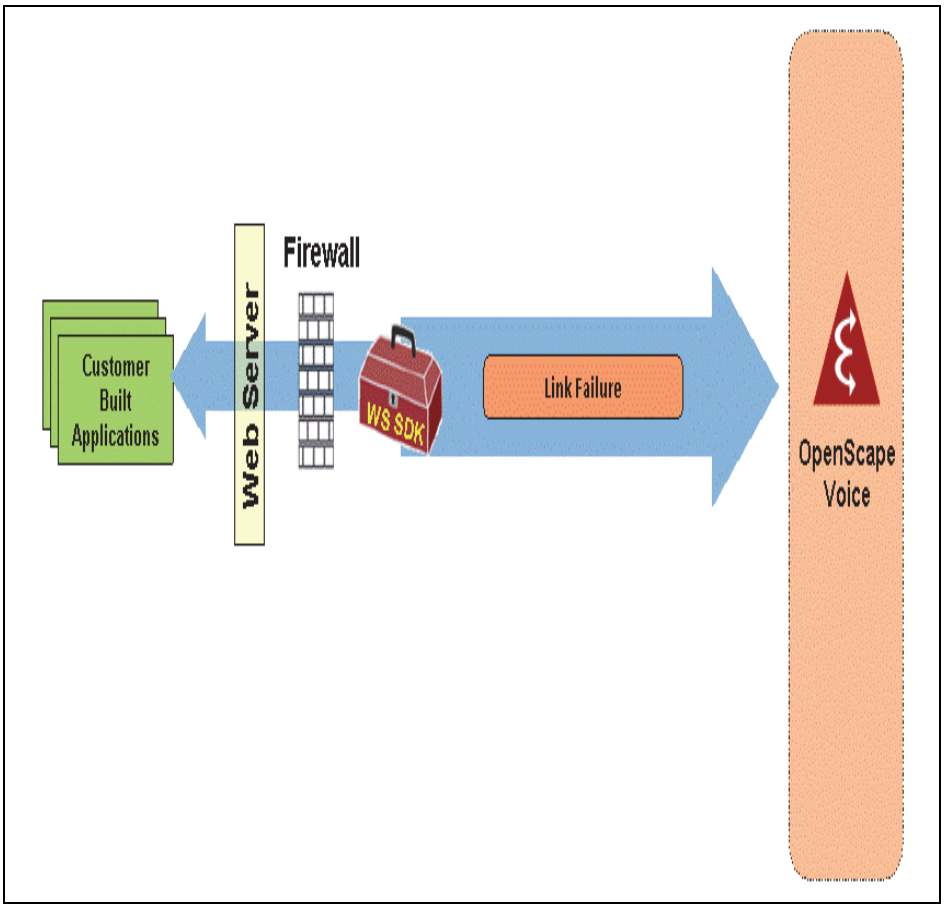


Figure 5 OpenScape Voice Link Failure SDK

With the Web Service for Link Failure Management, a Network Management Service (NMS) application can inform the OpenScape Voice system that an access link is down and instead a backup link will be used, or vice versa. With this information, an NMS application can always use the appropriate link capacity for its call admission calculations.

4.2 SDKs for Provisioning

The following set of SDKs will be provided by the OpenScape Voice system Real Time Server to facilitate Subscriber and Business Group management.

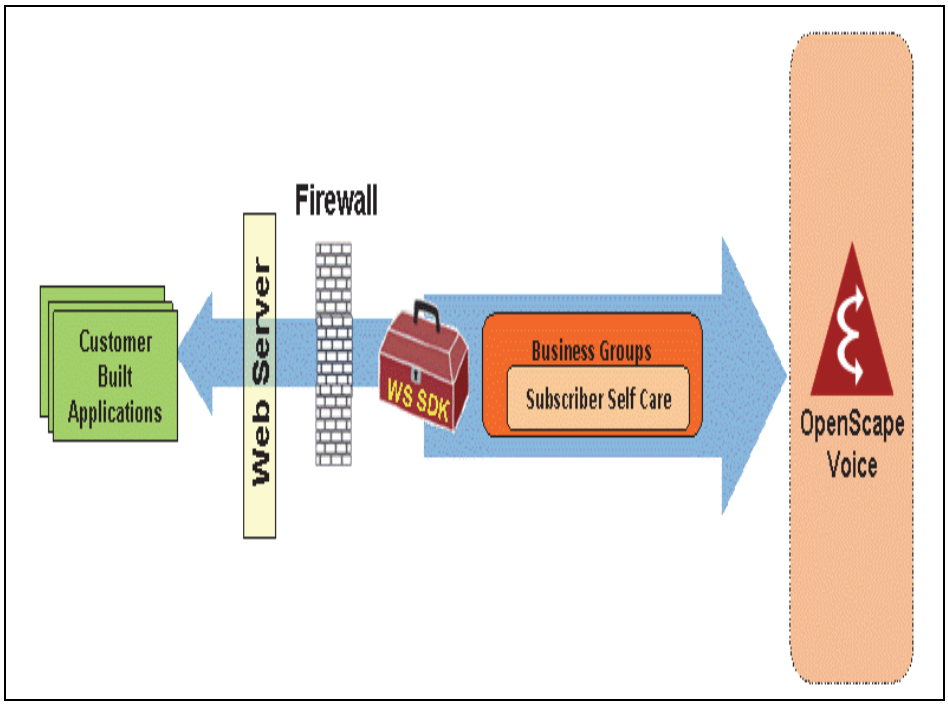


Figure 6 OpenScape Voice Provisioning SDKs

This is a set of layered SDKs. The outside layer is the Business Groups SDK, which adds its own functionality and contains the Subscriber Self Care SDK.

This set of SDKs supports the following functionality:

- ***Business Group Management***
  This SDK provides functions associated with managing business groups, BGLs, and numbering plans.
- ***Subscriber Self Care***
  This SDK provides functions associated with subscriber self care (SSC).

5 About the Link Failure Management SDK

The Link Failure Management (LFM) Management Software Development Kit (SDK) permits an application to directly manage the primary access links associated with a call admission control (CAC) group.

This Application Developer Guide (ADG) is one component of the Link Failure Management Provisioning SDK. The LFM SDK consists of a set of components as defined below:

- Link Failure Management WSDL File
- SDK Specific Documentation (Link Failure Management ADG - this document)
- Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning document

---

Attention: General information that applies to all Unify Software Development Kits (SDKs) is described in the *OpenScape Voice, Application Developers Manual: Volume 1, Web Services SDK Programming Overview*. Refer to that guide for information about the SDK package content, distribution, and other general topics.

5.1 SOAP/XML Concept

The Simple Object Access Protocol (SOAP) is a protocol for exchanging XML-based messages over computer networks, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework upon which more abstract layers can be built.

The SOAP Server is an application that hosts the server software, defines the WSDL and methods to process the requests. The SOAP Server is an integrated component of the OpenScape Voice software.

The SOAP Client is an application that hosts the client software and sends in the data via methods to provision the objects such as subscriber feature data on the switch.

The ADG for Link Failure Management Provisioning SDK is based on the Client/Server architecture. The Network Management System (NMS) application hosts the SOAP Client and the OpenScape Voice system hosts the SOAP Server. This ADG describes the web based interface for managing the primary access links associated with a call admission control group.

SOAP/XML interfaces provided by the OpenScape Voice system are backward compatible for N-2 software releases, N being the current release.

All of the SOAP/XML methods are modified to confirm the standards starting in V3.0 to allow one input parameter and one output parameter. Hence, all of the method names end with 2. This ADG does not explicitly state exact syntactical method names and instead they must be perceived as functional descriptions. To get correct syntactical method names, refer to the WSDL file. In addition, the code

5.2 Link Failure Management Concept

With the Link Failure Management SDK, an application can directly manage the primary access links associated with a call admission control (CAC) group. A typical application would be a Network Management System (NMS) application that is monitoring the status of the primary access link. Via the Link Failure Management SDK, the application can obtain a list of all the primary access links, obtain the status within the OpenScape Voice system of all primary access links, reset all primary access links, and notify the OpenScape Voice system of the failure or re-establishment of any primary access links.

For example, consider the following network topology in Figure 7:

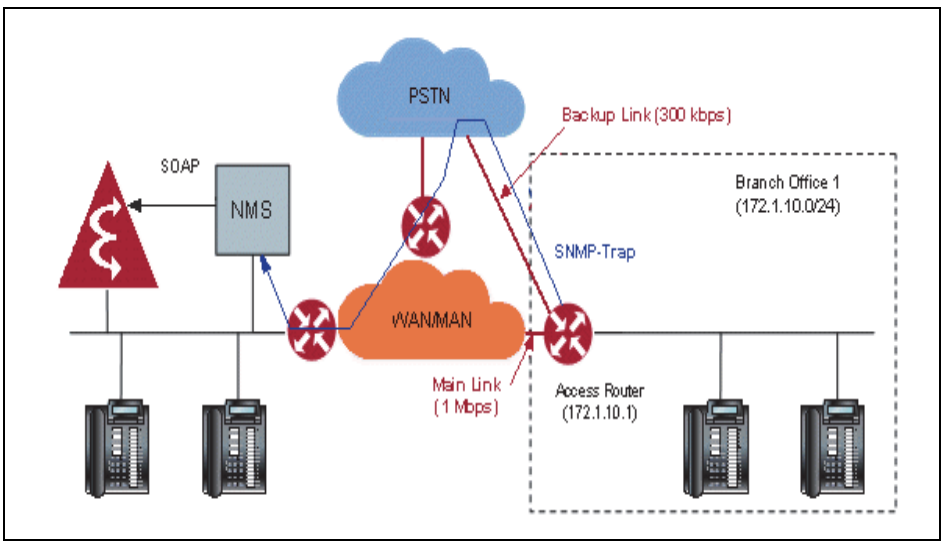


Figure 7 Example Topology

An NMS application monitors the primary access link and receives an SNMP trap when this link changes status. The NMS application could then notify the OpenScape Voice system of the status change, allowing the OpenScape Voice system to switch between the primary access link and the backup access link as necessary.

Call admission control is supported within the OpenScape Voice system by CAC policies and CAC groups.

CAC policies allow the provisioning of the capacity of the primary access links and backup access links.

CAC groups allow provisioning of a primary access link and a secondary access link, and associating them with an access router IP address and name (e.g., eth0). The provisioning of a secondary access link is optional and should be done only if the access router is configured to switch to a backup access link when the primary access link is down. The CAC group also provides a field that indicates the status of the primary access link, which can be UP or DOWN. If a link is marked as DOWN, a timestamp is applied indicating the time at which it was marked down.

For Figure 8, the following CAC group and policy could be created for the branch office as follows:

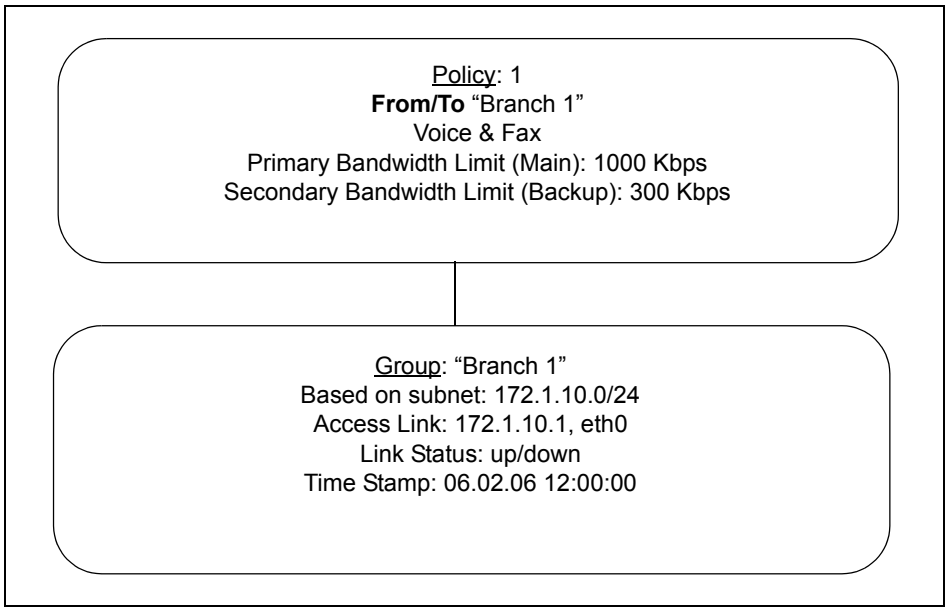


Figure 8 Example CAC Policy and Group

Using the Link Failure Management SDK, the NMS application can obtain the list of all primary access links that are associated to CAC groups in the OpenScape Voice system.

If the external NMS application becomes aware of a primary access link failure, it can notify the OpenScape Voice system via the Link Failure Management SDK. The OpenScape Voice system will then switch to use the backup access link capacity for the policy of the CAC group associated with the failed primary access link. If the primary access link comes up again, the NMS application will again notify the OpenScape Voice system via the Link Failure Management SDK. The OpenScape Voice system will then switch to use the primary access link for the associated CAC group.

The OpenScape Voice system enforces that the IP address of the primary access link assigned to a CAC group must be unique (i.e., the same IP address cannot be assigned to more than one CAC group).

5.3 Link Failure Management SOAP/XML Methods

The Link Failure Network Management Service (NMS) application can manage the CAC access links in conjunction with the OpenScape Voice system. The following subsections provide brief descriptions of the SOAP/XML methods needed to ensure the NMS application and the OpenScape Voice system uses the appropriate link capacity for its CAC calculations.

5.3.1 GetVersion2

This request allows the application to obtain version information from the OpenScape Voice system.

5.3.2 GetProvisionedCACPrimaryLinks

This request allows an application to query for the IP address and interface name of all primary access links that are associated with a CAC group in the OpenScape Voice system.

5.3.3 GetCACPrimaryLinkStatus

This request allows the application to query the link status (UP/DOWN) for one or for all CAC primary access links. This request gets the status that is set for the primary access links within the OpenScape Voice system; it does not get the real link status.

5.3.4 NotifyCACPrimaryLinkStatus

This request allows the application to inform the OpenScape Voice system about a change in status for a CAC primary access link. The Internal Resource Manager then uses the appropriate capacity according to the currently used link (primary or backup) for call admission control calculations.

This request is also used to refresh/confirm a link DOWN status for a primary access link. If the OpenScape Voice system does not receive the confirmation that the primary access link remains down, it automatically resets the link status to UP after a timeout. This timeout value can be configured in the OpenScape Voice system.

5.3.5 ResetAllCACPrimaryLinks

This request allows the application to attempt to reset the status for all primary access links provisioned in the OpenScape Voice system to UP.

5.4 Technical Implementation Notes

The following sections describe the technical implementation specific to this SDK.

5.4.1 Client Architecture

The ADG for Link Failure Management SDK is based on a Client/Server architecture. The Network Management System (NMS) application implements the SOAP client and the OpenScape Voice system implements the SOAP server. This ADG describes the web based interface for managing primary access links associated with CAC Groups on the OpenScape Voice system.

Section 5.3, “Link Failure Management SOAP/XML Methods”, on page 26 provides brief descriptions of the SOAP/XML methods that the NMS application can use to manage the subscribers on the OpenScape Voice system. Since this is based on request and response architecture every method has a response.

5.4.2 Licensing

Any one who purchases and acquires the Link Failure Management SDK is allowed to use all of the Web Services offered by this SDK.

5.4.3 Discovery of Web Services

There is no discovery mechanism defined. This web service can be accessed by using the OpenScape Voice system IP address and the port for the SOAP/XML operations. However, the Web Services Client's IP address must be set up as a trusted entity on the OpenScape Voice system to ensure the security of the OpenScape Voice system.

5.4.4 Connectivity

The OpenScape Voice system supports SOAP/XML using HTTP. By default the SOAP Server on the OpenScape Voice system is bound to a starting TCP port 8767. The default number of ports on which the SOAP Server accepts requests is incremented sequentially starting with 8767; by default, the valid ports are

8767, 8768, 8769, and 8770. The starting port and number of ports can be modified according to the needs of the client but must match the ports on the SOAP Client (NMS) side as well.

5.4.5 Request-Response Operation

Refer to the *OpenScape Voice, Application Developers Manual: Volume 1, Web Services SDK Programming Overview*.

5.4.6 Security

Network security shall be ensured by configuring the Web Services Client host IP Address as trusted port on the OpenScape Voice system.

5.4.7 Sessions

The SOAP Server is sessionless. Each SOAP request received is independent of all others.

5.4.8 SOAP Server Restart

The SOAP client is not informed when the SOAP/XML Server restarts. If the SOAP Server restarts, no response will be sent for any outstanding SOAP requests.

5.4.9 SOAP Client Restart

SOAP Server has no knowledge of the Web service client application restart. It is up to the Client to re-establish the connection if it needs to communicate with the SOAP Server.

5.4.10 Heartbeat Mechanisms

The SOAP Server cannot monitor the SOAP clients. If the SOAP Client requires a heartbeat mechanism with the SOAP Server, it is recommended that the SOAP client use the "GetVersion" SOAP/XML method because this method will respond the quickest while imparting the least impact on the call processing capabilities of the OpenScape Voice system. Periodically sending a "GetVersion" request and receiving a response would validate the connection between the SOAP Client on

the NMS application and the SOAP server on OpenScape Voice system. Care must be taken in deciding on the period used to implement the heartbeat, if the period is too short and the performance of the OpenScape Voice system or other SOAP clients can be impacted.

5.4.11 Error Handling

The errors that can occur during the processing of a SOAP request are described in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

5.4.12 Paging

For the SOAP requests that accomplish the Get List functionality, it is desirable that a mechanism be used to facilitate paging through the result set of the returned data by SOAP Servers, such as OpenScape Voice Assistant, that display the data graphically to the user. Therefore, the following paging mechanism exists to be used by the various Get List SOAP requests, as necessary.

All the SOAP methods that retrieve a list support the Paging mechanism.

For Get List SOAP requests, a paging structure is defined in the WSDL interface that includes the following fields:

- Page Size — the number of rows to be shown on a page. This is the number of rows displayed in the GUI. Must be >= 1 and the maximum of 1000.
- Page Number — The page that will be shown in the GUI. Must be >= 1 or be equal to -10, which indicates the last page.

Depending on the individual Get List request, there may be additional input parameters, including filter criteria, to apply to the existing objects on the system.

For Get List SOAP requests, a PaginatedInfo structure is defined in the WSDL interface that includes the following fields:

- Number of rows in the result set (total number of entries meeting the filter criteria, if any)
- Number of pages in the result set (total number of pages of data, given the number of rows and the Page Size)
- Page Size (same as Page Size in request)

- Page Number (same as Page Number in request)

Note: Paging is optional in order to make GetList SOAP requests backward compatible. If no paging information is sent, the entire list, up to the maximum possible elements, is returned. The following default behavior applies to paging requests:

- If PageSize is <= 0 or >= the total number of existing elements, no paging is assumed and the entire list is returned.
- If PageNumber == -10, or >= the last page, the last page is displayed (with the starting entry as it is specified with the last page number, so the size of this page may be less than the specified PageSize)
- If PageNumber <= 0, the first page is displayed.

The PaginatedInfo structure in the response will reflect the CurrentPage info according to the default behavior, not according to the actual user input for Paging.

Note: Sorting is a separate issue from Paging and is not addressed in this guide.

5.4.13 Unknown Tags

All the SOAP methods have a WSDL version input in the header to ensure only the data defined in that particular WSDL version are returned. It is recommended that the SOAP/XML processing on the SOAP Client side be written in such a manner that it ignores any unknown tags in the responses. This requires turning off the XML validations on the SOAP/Client during the code generation. However, if the SOAP Client does not provide the WSDL version in the SOAP method and SOAP Server in turn returns more data than SOAP Client expects, ignoring the tags will be very appropriate and does not harm the functionality. If the SOAP Client does not ignore then any extra data returned by the SOAP Server may cause an error on the SOAP Client.

5.4.14 SOAP Server Result

For a description of the Result structure returned by the SOAP Server to the SOAP Client, please refer to the chapter named "Subscriber Provisioning Result Information" in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

In case the SOAP Client issues a blank GET to the SOAP Server, it will receive no response.

5.4.15 Versioning Concept

A versioning concept is defined within the SOAP Server in order to control the SOAP/XML interface changes. The SOAP Client application need not align to the interface versioning concept because of the backward compatibility unless the application is requiring the changes. The backward compatibility is only assured for N-2 product releases.

The possible reasons for Web Services interface modifications are:

- The SOAP/XML interface can change within the lifecycle of a Release due to fault reports and/or change requests. (update)
- The SOAP/XML interface can change between two Releases due to the introduction of new features. (upgrade)

In both cases the SOAP Client application must be aware of these changes or must be able to safely ignore them. In this context, one has to distinguish between compatible and incompatible interface changes:

- **Compatible Interface Changes**

  Changes in the SOAP interface are compatible if they do not lead to verification failures on the SOAP Client application side.

  If the SOAP Client application was created using code generation tools like Axis (for Java), the code created probably does an xml-schema verification of the incoming SOAP/XML responses. When the verification fails, the operation will likely not be processed. Changes in the SOAP interface are compatible if they do not lead to verification failures on the SOAP Client application side.

  A typical compatible change is adding a new XML tag to the SOAP/XML methods. Since the SOAP Client application that does not know about this change is not aware of this new tag, it can not request it and therefore does not receive a response it can not handle. If the SOAP Client passes the WSDL version in the requests then responses will always correspond to the requested WSDL.

- **Incompatible Interface Changes**

  Incompatible interface changes are all changes that lead to an xml schema verification failure on the SOAP Client application side. Examples of such changes are:

  – Adding a new tag to a SOAP/XML response

  – Changing an enumeration definition

  – Deleting a SOAP operation

  In general, the OpenScape Voice SOAP Server will not make any incompatible interface changes like these (within the N-2 product releases at least).

In order to track changes of the SOAP/XML interface, a product version is being defined which is returned in the "getInterfaceVersion" SOAP/XML operation. This product version consists of the following components:

<Release>.<Version>.<Build>.<Internal Revision>

- Release: This is the general release version of the OpenScape Voice product. This will not change for SOAP adaptations within a release.
- Version: This is a point release version of the OpenScape Voice product. This will not change for SOAP adaptations within a release.
- Build: This identifies the WSDL version. Any changes to this number indicate the SOAP/XML interfaces have been changed.
- Internal Revision: This identifies the internal changes to the WSDL version. These changes do not have any impact to the existing interfaces.

5.4.16 WSDL Parser and Compiler

Currently, Unify uses gSOAP and Axis to parse the WSDL and compiler to generate the stubs and skeletons to invoke the service or build a new service based on the WSDL. However, there are some customers who use different tools such as Web logic to generate their code using the Unify produced WSDL files. If the customer chooses to use any other tools than the gSOAP and Axis, additional support may be required.

All the WSDL files contain the following lines:

```
<port name="siemens-hiq8000" binding="tns:siemens-hiq8000">
<SOAP:address location="http://www.siemens.com/cgi-bin/soapServer"/>/
port>
```

Note that the location field is not used.

5.5 Link Failure Management Interface Details

A detailed description of the Link Failure Management interface is provided in the *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

5.6 Link Failure Management SDK Usage Examples

This section outlines at the high level some typical usage scenarios. The level of detail is down to the parameter level of the requests and their responses.

5.6.1 NMS Application Startup Scenario

This usage scenario describes how a NMS application could initialize itself using the Link Failure Management SDK in conjunction with the OpenScape Voice system-internal call admission control solution. Pseudo code will be used in the examples.

Preconditions:

- A configured, working OpenScape Voice system that uses CAC groups and CAC policies.
- An NMS application that will use the Link Failure Management SDK to manage the configured access links.
- The NMS application is starting up.

Usage Scenario

- NMS application gathers all provisioned primary access links with a call to GetProvisionedCACPrimaryLinks. The NMS application must then save this information locally.
- To clear any previously reported states in the OpenScape Voice system, the NMS application attempts to reset all primary access links by using ResetAllCACPrimaryLinks.
- The NMS application finds out the current status of all primary access links obtained earlier via its interface to the NMS host. This could be done, for instance, by sending SNMP GET requests to the access routers. This is not part of the Link Failure Management SDK.
- For any link that is down, the NMS application notifies the OpenScape Voice system using NotifyCACPrimaryLinkStatus. In this example, link 172.1.10.1 is down.
- At this point the NMS and the OpenScape Voice system are in sync and the NMS application can perform link monitoring. Refer to Section 5.6.2, “NMS Application Link Monitoring Scenario”.
- NMS application starts "listening" for standard linkUp/linkDown SNMP traps from all primary access links. This is not part of the Link Failure Management SDK.

5.6.2 NMS Application Link Monitoring Scenario

This usage scenario describes how a NMS application could use the Link Failure Management SDK in conjunction with the OpenScape Voice system-internal call admission control solution to manage primary link status. Pseudo code will be used in the examples.

Preconditions:

- A configured, working OpenScape Voice system that uses CAC groups and CAC policies.
- An NMS application that has gone through a startup sequence similar to that described in Section 5.6.1, “NMS Application Startup Scenario”.

Usage Scenario:

- Upon receiving a linkUp/linkDown SNMP trap, the NMS application will notify the OpenScape Voice system of the primary access link status change via a call to NotifyCACPrimaryLinkStatusRequest in order to set the primary access link status to UP or DOWN. See Section 5.6.1, “NMS Application Startup Scenario” for an example.

  Note: The NMS application should include some kind of link settling period and/or mechanisms to prevent it from bombarding the OpenScape Voice system with link up, link down statuses for the same link in a short period of time. A 5- to 10-second settling period is recommended.

- When the NMS application reports a primary access link status as LinkDown, it should continuously monitor the real status of the link via its interface with the NMS host. This is required because SNMP traps are delivered via UDP and thus are not 100% reliable.

  While the link status remains DOWN, the NMS application must periodically confirm the link DOWN status in the OpenScape Voice system via a call to NotifyCACPrimaryLinkStatusRequest. If the OpenScape Voice system does not receive the confirmation that the primary access link remains DOWN, it automatically resets the link status to UP after a timeout. This timeout value can be configured in the OpenScape Voice system. Assume in this example that the NMS application needs to confirm that link 172.1.10.1 is still down.

5.6.3 OpenScape Voice Provisioning Change

This usage scenario describes how an application can process provisioning changes for CAC primary access links.

Preconditions:

- An application that has been using the Link Failure Management SDK to manage configured access links.
- A provision change has been performed on the OpenScape Voice system, adding and/or deleting CAC primary access links.

Usage Scenario:

- The application will receive no indication from the Link Failure Management SDK of a CAC provisioning change on the OpenScape Voice system. The application should query the OpenScape Voice system for all provisioned access links on specified intervals (e.g., every 6 hours) to maintain synchronization. In addition, the application should provide the means to request a new synchronization manually.
- Upon being notified of the change, the application must re-obtain the list of primary access links and do a comparison to what it has, removing any links not there and adding monitoring to any links that are new. See Section 5.6.1, “NMS Application Startup Scenario” for pseudo code.
- The application can obtain the status of the new links using GetCACPrimaryLinkStatus. In this example, it gets the status of all links.

5.6.4 Heartbeat Between Application and OpenScape Voice

This usage scenario describes how an application could use the GetVersion2 request to implement a heartbeat between itself and the OpenScape Voice system.

Preconditions:

- An application that has been using the Link Failure Management SDK to manage configured access links.

Usage Scenario:

- The application does a GetVersion2 request to ensure proper connectivity to the OpenScape Voice system. If the ResultCode is GOOD then connectivity exists. See Section 5.6.1, “NMS Application Startup Scenario” for an example of how to use this request.

5.6.5 Handling Paged Data

This usage scenario describes how an application can use the various forms of paging to get result data. See Section 5.4.12, “Paging” for a description of paging.

Preconditions:

- Application is going to use a request that uses paging.

Usage Scenarios:

- Get data all at once.
- Get the 1st page of data.
- Get the last page of data.
- Get data in chunks from beginning to end.

6 About the Business Group Management SDK

The Business Group (BG) Management Software Development Kit (SDK) can be used for managing business group resources such as main numbers, features, private number plans, authorization codes and other BG relevant data. The Business Group Provisioning SDK consists of a set of components (Bild 9) as defined below:

- Business Group WSDL File
- SDK Specific Documentation (Business Group Provisioning ADG - this document)
- Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning document
- Sample applications: Java sample code illustrating some basic provisioning and the respective executables to test the methods GetSubInfoByRel, GetVersion and UpdateSubscriberFeatures.
- Sample SOAP/XML messages for GetSubInfoByRel, GetVersion and UpdateSubscriberFeatures.

This Application Developer Guide (ADG) is intended for application developers who wish to use the Provisioning SDK for the development of custom Business Group Provisioning Applications. This SDK describes all of the SOAP/XML methods that are needed to provision and maintain the Business Group system.

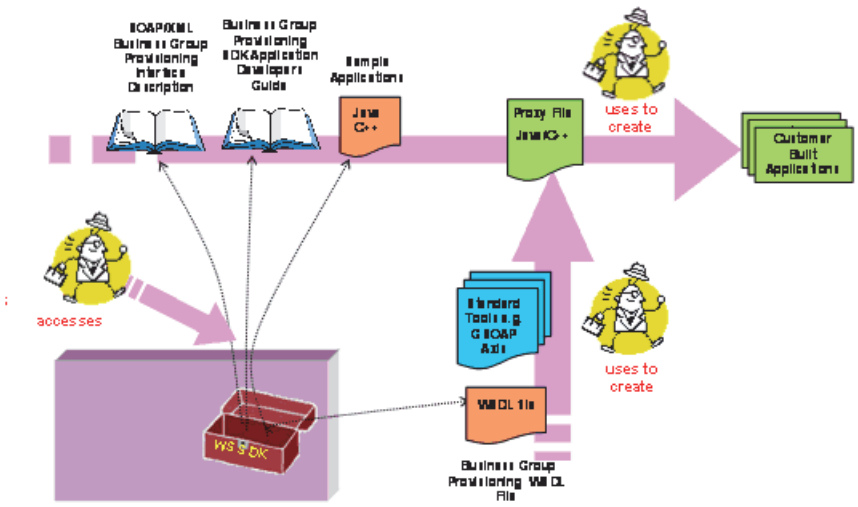


Figure 9 Components of Business Group SDK

Although this ADG enables customers to enhance their applications by accessing BG Provisioning management web service interfaces, it does not provide end-user authentication nor authorization functions. Those are expected to be handled by the application using their own databases.

Attention: General information that applies to all Unify Software Development Kits (SDKs) is described in the *OpenScape Voice, Application Developers Manual: Volume 1, Web Services SDK Programming Overview*. Refer to that guide for information about the SDK package content, distribution, and other general topics.

6.1 SOAP/XML Concept

The Simple Object Access Protocol (SOAP) is a protocol for exchanging XML-based messages over computer networks, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework upon which more abstract layers can be built.

The SOAP Server is an application that hosts the server software, defines the WSDL and methods to process the requests. The SOAP Server is an integrated component of the OpenScape Voice system software.

The SOAP Client is an application that hosts the client software and sends in the data via methods to the provision the objects such as BG and BG Lines on the system.

The ADG for Business Group Provisioning SDK is based on the Client/Server architecture. The Operations Support System (OSS) application hosts the SOAP Client and the OpenScape Voice system hosts the SOAP Server. This ADG describes the web based interface for building and managing the Business Group.

SOAP/XML interfaces provided by the OpenScape Voice system are backward compatible for N-2 software releases, N being the current release.

All of the SOAP/XML methods are modified to confirm the standards starting in V3.0 to allow one input parameter and one output parameter. Hence, all of the method names end with 2. This ADG does not explicitly state exact syntactical method names and instead they must be perceived as functional descriptions. To get correct syntactical method names, refer to the WSDL file. In addition, the code generation tools ensure that correct method names are used.

6.2 Basic Business Group Concept

Typically, service providers take the orders for a new telephone service and feed the data into the backend system that connects to the OpenScape Voice system and manages the BGs. For example, a Service provider takes the order for a new BG service. The BG information will then be fed into the backend office and it will trigger an Operations Support System (OSS) to connect to the OpenScape Voice system to create a BG with the requested features on the system. Refer to Bild 10.

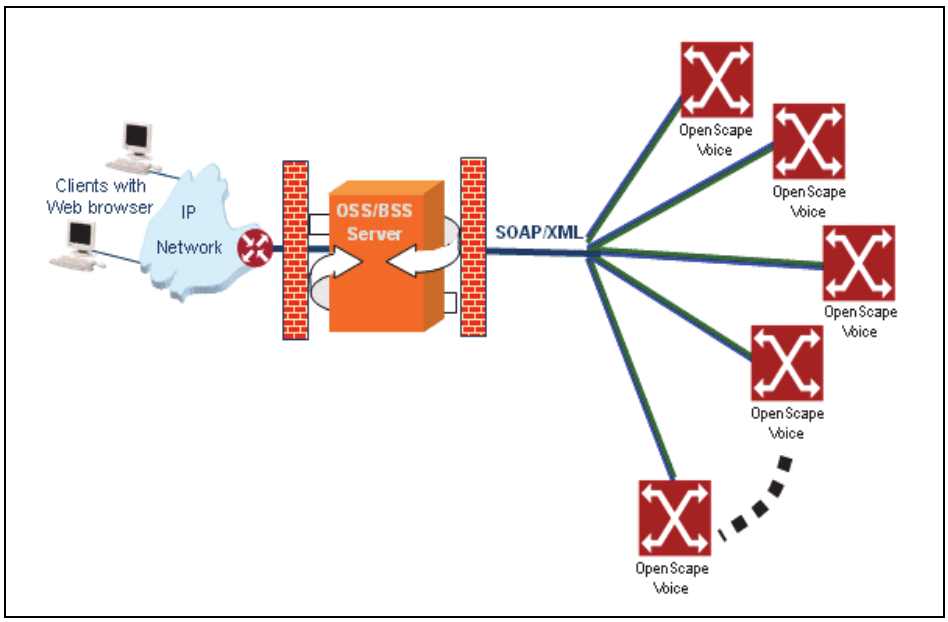


Figure 10 BG Management Topology

6.3 Business Group SOAP/XML Methods

The following subsections provide brief descriptions of the SOAP/XML methods that are needed to provision and maintain the OpenScape Voice system Business Group system. They do not list every possible response because every method can return different error codes that are described in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*. Some responses are explained where warranted.

6.3.1 GetVersion

This SOAP/XML request is used to get the OpenScape Voice system build and release information including the Web Service Description Language (WSDL) version such that the Client/OSS may know what interface methods are available on the system. In the Client/Server architecture, the Client must never operate a higher WSDL version than the WSDL version on the Server because the Server will not be able to recognize the data in the SOAP/XML method such as new tags/data. For example if the application is communicating with more than one OpenScape Voice systems that operate on different releases having different WSDL versions, it is necessary that the SOAP Client is operating at the older version of WSDL. Since the OpenScape Voice system is N-2 releases backwardly compatible, the SOAP Client must be upgraded last in the network of OpenScape Voice system(s) and Client(s)/OSS(s).

6.3.2 GetVersionResponse

A successful response contains the WSDL version, the OpenScape Voice system load build version, Environment of the system (Enterprise) and the timestamp when the SOAP Server process was last built.

6.3.3 CreateBG

A Business Group (BG) is an entity of related subscribers. Subscribers may optionally belong to a Business Group. The name may be up to 16 characters long. Business Groups present new service features to members of the Business Group and a simpler method for managing the subscriber of a group.

Business Groups have a status, Main Numbers, Attendant Number, Message Detail Recording (MDR) information, Numbering Plan Name and Service Access Code. Features may be assigned to Business Groups. Subscribers who are members of a BG inherit the features of the group that they are part of, if Feature Profile is used. Additionally, individual features may be assigned or restricted to individual subscribers of the group.

If a Numbering Plan Name is not provided when creating a Business Group, the default Numbering Plan is provisioned with E164 NANP. Each Business Group must have a unique Private Numbering Plan.

Once a Business Group is created with a Numbering Plan, it can be changed to another Numbering Plan only if there are no subscribers existing in the Business Group.

When creating a Business Group, the system generated Customer Id will be used as the Business Group Index. And the user specified Business Group Name will be used as the Customer Name.

This method is used to create a Business Group. A BG Name is assigned with this method along with an administrator-assigned OperatorId that assists in identification/auditing of the BG. Also assigned here is the Numbering Plan Name for the number plan that the BG follows. There is also a Display Number field for the number that will be presented as the calling number whenever a private number in the BG makes an external call.

6.3.4 AddBGMainNumber

The AddBGMainNumber method is used to add the Main Number for the BG, Attendant Number, auto Attendant Number and Auto Attendant Status. The Main Number can be a virtual number or real number. The Attendant needs to be created before assigning to the BG.

6.3.5 DeleteBGMainNumber

The DeleteBGMainNumber removes a Main Number from a BG.

6.3.6 UpdateBGMainNumber

This SOAP request is used to add, replace, or delete a BG's main number.

6.3.7 UpdateBGFeatures

The method UpdateBGFeatures is used to Add, Replace or Delete a feature to a BG. If the feature is assigned to the BG successfully, the BGL will inherit the feature. When removing the feature from BG, the feature will also be removed from BGL.

6.3.8 GetBGInfo

The method GetBGInfo is used to retrieve BG related information.

6.3.9 GetBGInfoByOptions

This method returns the BG Info based on the options specified. The options include a list of features, CPUs, MainNumber, DialPlan and BGService Access Codes.

6.3.10 GetBGList

The method GetBGList is used to retrieve the list of BGs created on the system.

6.3.11 DeleteBG

This SOAP request is used to delete a BG. The BG cannot have any existing subscribers, endpoint profiles or associated Numbering Plans. Other BG-dependent objects, such as BG Subnets and BG Departments are deleted automatically upon successful completion of this request.

6.3.12 GetBGAttendantNumbers

The method GetBGAttendantNumbers is used to retrieve the list of DNs provisioned as Attendant Numbers for the BG.

6.3.13 GetTotalCPUCountUsed

This method returns the total used number of call pickup groups. Note: the same CpuId in different BGs are counted as different.

6.3.14 DeleteBGCpu

Deletes the list of CPU's (Call PickUp groups) belonging to this BG. All the BGLs belonging to this BG and having the corresponding CPUId will have their CPUId set to 0. All of the BGLs will have this subscriber feature set to inactive.

6.3.15 UpdateBGParms

This request is used to modify the values of BG-related parameters for an existing Business Group. Examples are Message Detail Recording (MDR) Indicator, Number Plan Name, and Display Number.

6.3.16 CreateBGDept

Business Groups can further be divided into departments and BGLs can then be assigned to them. This method is used to create a BG Department. The following functionality for departments is supported.

- Create department (this method) - Every BG can have a maximum of 50 departments, with any number of BGLs assigned to them. The name of a department is restricted to twenty characters.
- Delete department - Departments which do not have any BGLs assigned to them can be deleted. If a department has BGLs assigned to them the request for deletion will be rejected.
- Modify department name - Change the name of an existing department.
- Retrieve a list of departments - This returns the list of departments created for the particular BG.
- Retrieve a list of subscribers assigned to a department.
- Assign/Unassign a BGL to a department.

6.3.17 DeleteBGDept

This method enables the operator to delete a BG department.

6.3.18 ModifyBGDept

This method modifies a BG department.

6.3.19 GetBGDeptList

This method retrieves a list of BG departments.

6.3.20 GetBGSubnetInfo

Business Group Subnet administration provides the BG administrator with the ability to provision the E911 Subnet Table on a per-BG basis. The purpose of a BG Subnet is to associate a LIN, or Location Identification Number, with a particular physical location to be used by an emergency operator, also known as a PSAP (Public Safety Answering Point). When a call is received from a SIP phone with an IP address within a particular BG Subnet, the LIN that is associated with that subnet is used to determine the location to which the needed emergency services should be dispatched. A BG Subnet can consist of a range of IP addresses that represents a particular office or section of a building, or that can be administered to the granularity of a single IP address that represents an individual desk. With the SOAP requests described in the following section, the SOAP Server provides the ability to create, update, delete, and retrieve information about BG Subnets.

This method retrieves information about a BG's subnets.

6.3.21 CreateBGSubnet

This method creates a BG subnet entry.

6.3.22 GetBGSubnetList

This method retrieves a list of subnets.

6.3.23 DeleteBGSubnet

This method deletes BG Subnets from the OpenScape Voice system.

6.3.24 UpdateBGSubnet

This method adds, modifies, or removes a BG subnet.

6.3.25 CreateAuthCode

Depending on the traffic type, an off-network authorization code may be required to allow external calls. The authorization codes are defined at the BG level, not at the BGL level. Each BG can have 50,000 authorization codes defined. The maximum number of authorization codes that can be defined on the system is 100,000. Each time a BG subscriber dials a number that corresponds to an

external call, an authorization code may be provided. If provided, it is validated by call processing to determine whether the subscriber is authorized to make the call. With the SOAP requests described in the following sections, the SOAP Server provides the ability to create, delete, and retrieve BG authorization codes. All of these requests can be done for one authorization code or a list of authorization codes.

This method creates authorization codes.

6.3.26 DeleteAuthCode

This method deletes an authorization code.

6.3.27 GetAuthCodeList

This method retrieves a list of authorization codes.

6.3.28 GetAuthCode

This method retrieves an authorization code.

6.3.29 CreateBgSpeedDialList

The SOAP requests in this section allow a Business Group administrator to define, populate, and manage BG Speed Dial lists. Out of a system pool of 10,000 BG System Speed Dial lists and 1,000,000 Speed Dial entries, a BG can be assigned up to 10 (1..10) BG System Speed Dial lists, where a list may have up to 1,000 entries (0..999). When the available pool of lists or of entries is exhausted, the BG administrator is informed during provisioning.

This request creates a BG Speed Dial List.

6.3.30 UpdateBgSpeedDialList

This request updates the name of a BG Speed Dial List.

6.3.31 UpdateBgSpeedDialListEntries

This request adds entries to a BG Speed Dial List or updates existing entries.

6.3.32 DeleteBgSpeedDialListEntries

This request deletes entries from a BG Speed Dial List.

6.3.33 DeleteBgSpeedDialList

This request deletes an entire BG Speed Dial List including all its entries.

6.3.34 GetBgSpeedDialList

This request retrieves a BG Speed Dial List.

6.3.35 CreateBGSuite

This request creates a BG Suite.

6.3.36 UpdateDNReservation

This request updates a DN reservation to reserve DNs or unreserve previously reserved DNs for a BG.

6.3.37 GetBGDnList

This request retrieves a list of DNs reserved for a BG.

6.3.38 CreateNumberPlan

The Private dialing and Numbering Plan (PNP) feature provides for up to 999 PNPs. A PNP exists in a private network and is used by private network subscribers that may belong to a Business Group (BG). A number in a PNP Numbering Plan is in the form Location_Code+Extension. The PNP defines the dialing patterns and access codes independent of the Public Dialing Plan. The PNP is assigned at the Business Group level and must be unique for each BG, except for the system default number plan.

The Numbering Plan is the first object to be created for a PNP. All other related dialing objects, such as Prefix Access Code, will be assigned to a particular NumberPlanName that is part of the PNP.

This method creates a PNP.

6.3.39 UpdateNumberPlan

This method is used to associate the BG with a Numbering Plan. If the specified Numbering Plan is not reserved for a BG originally, it can be updated to be reserved for an existing BG. If it is already reserved for a BG, but not assigned to a BGL, it can be reserved for another existing BG or un-reserved from the currently reserved for BG.

If the Numbering Plan provided is already reserved for a BG and already assigned to some BGL, it can not be reserved for another BG or un-reserved from the currently reserved-for BG unless removing the associations with all the BGLs.

6.3.40 DeleteNumberPlan

This method deletes the Numbering Plan. Numbering Plan cannot be deleted if there are existing references to it, including if it is configured as the BG Common Number Plan for any BG.

6.3.41 GetNumberPlanList

This method is used to retrieve the list of Numbering Plans available on the system or information about a specified Number Plan.

6.3.42 CreateDestCode

This method is used to create an E.164 Destination Code for a Numbering Plan. This enables the association of an E.164 Destination Code to the appropriate destination. Each E.164 Code is uniquely defined by the E.164 Code, the Numbering Plan Id, the Nature of Address, the Originating Class of Service, and the Originating Rate Area.

Note: The provisioning of the CodeIndex object is overlaid on the Destination Code (E164 DN code). To qualify the object being administered as a CodeIndex, a new field is introduced, CodeIndexName, which will differentiate the object from the E164 code objects.

6.3.43 ModifyDestCodeInfo

This method is used to change the Destination Code information. Only the NPA, the Destination Name, Destination Type, the TrafficType, and/or the HomeDN Office Code can be updated when changing the Destination Code information.

6.3.44 DeleteDestCode

This method is used to delete a Destination Code or a Code Index.

6.3.45 GetDestCodeList

This method is used to retrieve a list of Destination codes. When filters are provided, all the Destination Codes that belong to the specified filter are retrieved. If a Destination Code is specified with a wild card, all the Destination Codes that start with the specified Destination Code digits are retrieved. Entering a filter is available with this option. If neither the Destination Code nor a filter is specified, all the Destination Codes are retrieved.

If max Destination Codes is specified, all the Destination Codes are retrieved up to the maximum number of Destination Codes specified. If the max Destination Code is not specified, the maximum number of Destination Codes retrieved at one time is 100.

6.3.46 GetDestCode

This method is use to retrieve the given Destination Code data.

6.3.47 CreatePrefixAccessCode

This method is used to create a Prefix Access Code. The Digits, DNMinLen and DNMaxLen must identify a unique Prefix Access Code within the specified numbering plan.

6.3.48 ModifyPrefixAccessCode

This method is used to modify the specified Prefix Access Code data.

6.3.49 DeletePrefixAccessCode

This method is used to delete the specified Prefix Access Code.

6.3.50 GetPrefixAccessCodeList

This method is used to retrieve the list of all the Prefix Access Codes on the system.

6.3.51 GetPrefixAccessCodeOne

This method is used to retrieve the data of a given Prefix Access Code.

6.3.52 CreatePnpLocationCode

A number in a PNP Numbering Plan is in the form LOC+Extension, where LOC is the Location Code, which can be broken down into the L2, L1 and L0 level. The levels L2, L1 and L0 are optional, meaning that a PNP Numbering Plan does not necessarily use L2, L1 or L0.

L0 (Level 0), the subscriber location code, is 0 to 4 digits in length, specifically 3 digits, fixed length for the PSTN Business Group uniform Numbering Plan (RNX-XXXX). L0 digits can overlap with the Extension digits, for example, 923-5505, where 923 is the L0 code and 3-5505 is the extension.

L1 (Level 1), the national location code, is 0 to 6 digits in length. L1 cannot be administered without L0.

L2 (Level 2), the international location code, is 0 to 4 digits in length. L2 cannot be administered without L1.

This method is used to create Private Numbering Plan Location Code.

6.3.53 ModifyPnpLocationCode

This method is used to modify the Private Numbering Plan Location Code.

6.3.54 DeletePnpLocationCode

This method is used to delete the Private Numbering Plan Location Code.

6.3.55 GetPnpLocationCodeList

This method is used to retrieve the list of Private Numbering Plan Location Codes on the system.

6.3.56 CreatePnpExtension

Extension Dialing allows a subscriber in a Business Group to dial other subscribers in the same Business Group using an abbreviated number, referred to as an extension (or intercom code). The extension number is determined by the Private Numbering Plan.

The Extension Number is a number that uniquely identifies the subscriber in the location. The most significant digits of extension numbers and the least significant digits of L0 can overlap, for example, 956XXXX, where L0 is 956 and extension is 6XXXX. An extension number is 1 to 7 digits in length.

This method is used to create the extension part for the Private Numbering Plan.

6.3.57 ModifyPnpExtension

This method is used to modify the extension codes and length of the extension needed to dial intercom.

6.3.58 DeletePnpExtension

This method is used to delete the specified Private Numbering Plan Extension.

6.3.59 GetPnpExtensionList

This method is used to retrieve a list of all the Private Numbering Plan Extension on the system of a given Numbering Plan or set of Extensions that match the given prefix.

6.3.60 CreateDNDefinition

From the end user’s perspective, it is useful for the called party to see a displayed number that can be used to call back the same calling party (instead of prefixing the public access code, country code, etc. to the display number before calling back). In order to support the display of a dialable number, there are administrative tables for number modification.

The dialable display number feature is per Numbering Plan based. It is applicable for all markets, all solutions. All these display number related tables are optional.

Any existing features or future features which have the number display functionality shall interact with the dialable display number feature. The dialable display number feature applies only if the display number supporting tables are provisioned. If they are not provisioned, then the current display number function will be used with no modification applied. To disable the dialable display number feature, the entries in these tables need to be removed.

This method creates an entry in the Display Number Definition table.

6.3.61 DeleteDNDefinition

This method deletes the specified entry from the Display Number Definition table.

6.3.62 GetDNDefinition

This method retrieves the entries from the Display Number Definition table corresponding to the specified Numbering Plan.

6.3.63 GetDNDefinitionNPList

This method retrieves a list of Numbering Plans with entries in the Display Number Definition table.

6.3.64 CreateDNPrefix

This method creates an entry in the Display Number Prefix table.

6.3.65 ModifyDNPrefix

This method modifies the specified entry in the Display Number Prefix table.

6.3.66 DeleteDNPrefix

This method deletes the specified entry from the Display Number Prefix table.

6.3.67 GetDNPrefix

This method retrieves the specified entry from the Display Number Prefix table.

6.3.68 GetDNPrefixList

This method retrieves a list of entries from the Display Number Prefix table for the NumberPlanName provided. If NumberPlanName is an empty string, this is interpreted as "ANY" Numbering Plan. If NumberPlanName is not provided (NULL), no filtering is performed based on Numbering Plan and the results for all Numbering Plans are returned.

6.3.69 CreateDNModification

This method creates an entry in the Display Number Modification table.

6.3.70 ModifyDNModification

This method modifies the specified entry in the Display Number Modification table.

Note: The value DNMAny may be passed for the parameter DNMCallTypeOfNumberIn only if the value DNMAny or DNMAnyPre is passed for the parameter DNMCallTypeOfNumber.

6.3.71 DeleteDNModification

This method deletes the specified entry from the Display Number Modification table.

6.3.72 GetDNModification

This method retrieves the specified entry from the Display Number Modification table.

6.3.73 GetDNModificationList

This method retrieves a list of entries from the Display Number Modification table for the FromNumberPlanName and ToNumberPlanName provided. If the number plan name is an empty string, this indicates "ANY" Numbering Plan. If the number plan name is not provided (NULL), no filtering is performed based on that number plan name.

6.3.74 GetSubscriberInfoByRel Request

This method retrieves all the subscriber's data including features, account user info etc. Service ID which is same as end user DN is a required parameter. The data to be retrieved is managed by setting the attribute GetSubscriberDataOption in the method. For example to retrieve only the list of features assigned to the subscriber attribute IncludeListOfFeatures must be set. If no attribute is set then all the data pertaining to the subscriber will be returned. It may be simpler to the

Client application to get all the subscriber data instead of choosing the data that might differ from one end uer to another, and display only the data that must be visible to the end user.

6.3.75 GetSubscriberInfoByRel response

If the Service ID is a valid subscriber DN, then all the data attributes set in the Subscriber Data Options parameter determines the data returned. If none are set all the end user data will be returned back to the client application.

6.3.76 GetSubTranStatus

This method retrieves the Subscriber transient status, for the DN supplied in the Service ID.

6.3.77 UpdateSubscriberFeatures request

This method is used to add/modify/delete the features of a subscriber. The client must control what operations are allowed to be performed by the subscriber and what features are allowed to manage. Typically the subscribers are only allowed to update the status of their features such as activate/deactivate Call Forwarding Variable and update the call forward to number.

6.3.78 Create Subscriber

This method is used to create a Business Group Line. The same method is used to create a BG and BG Line because both of these are treated as subscribers and mostly have common data. To create a SIP subscriber only the IPPhone data must be provided in the ConnectionInfo structure. All the features applicable to a Business Group Line may be assigned during the creation of the Business Group Line. It is advised to use a QoS profile instead of providing specific QoS parameters in the method. QoS profile and Feature Profile must be created before being assigned to the subscriber.

6.3.79 GetSubscriberList

This method is used to retrieve the list of subscribers on the system. The request contains a starting subscriber and a maximum subscriber count to be retrieved. The subscribers are returned in numerical order from lowest DN to highest. A

number of filter criteria are possible so that a particular subset of existing subscribers meeting certain criteria can be retrieved. All conditions specified are AND-ed together.

The highest supported and default maximum number of subscribers to retrieve is 1000. This limit is set in the system to not cause any overload problems having a detrimental effect on the call processing. However, the total number of existing subscribers that meet the given filter criteria is returned in the TotalSubsFound field so that if there are more than 1000 existing subscribers, the client application can send multiple requests, each one with a starting subscriber one greater than the last one returned and thus obtain all existing subscribers. If no filter criterion is set the TotalSubsFound indicates the total number of subscribers provisioned on the system.

Note: The response list will contain disconnected DNs only if an InterceptAnnouncement filter criterion is provided.

6.3.80 GetTombStoneSubscriberList

This method is used to retrieve the list of subscribers deleted after a specified date. It can also be used to retrieve the list of subscribers that has had a specific service unassigned (currently only CSTA is supported) after a specified date.

6.3.81 GetKeysetPrimaryList

This method retrieves a list of all the Keyset Primary Lines that the DN provided in the request as a Line Appearance.

6.3.82 DeleteSubscriber

This request is used to delete asubscriber from the system and this includes the disconnected DNs. Upon successful execution of this request, there is no longer any data associated with this DN on the system.

If there is a failure to delete complete data associated with the DN on the system, it will continue and delete everything that it can and it will set the HomeDN to vacant so that this deleted DN is available to be assigned for a new service.

6.3.83 UpdateSubscriberStatus

This method is used to change the administrative status of the subscriber. If the subscriber is blocked, the subscriber will not be able to receive or originate any calls.

6.3.84 DisconnectSubscriber

This method is used to remove a subscriber and mark the DN as disconnected. The subscriber will no longer be defined on the system and callers to this number will receive a Number Disconnected announcement or the New Number announcement if the new DN is provided that is not on the system. See UpdateSubscriberDN to change a subscriber's Directory Number to a new number on the same system.

The DN will not be usable for re-assigning to other subscribers until the DN is removed from the system using the DeleteSubscriber method.

When service for a subscriber is to be terminated, the subscriber will be disconnected and the DN will not be reused for some sterilization period. The DN should remain in the disconnected state during this sterilization period.

The system does not manage any sterilization period or require it. It does not automatically delete disconnected numbers after some time period. Instead the DN must be deleted explicitly by the system administrator.

6.3.85 UpdateSubscriberPICs

This method is used to change the subscriber's local long distance, domestic long distance and international default Primary inter-LATA Carrier. When modifying any PIC, the data for all the other PICs must be provided. If not, the existing PIC will be removed and no default PIC is available for calls.

6.3.86 UpdateSubscriberAccountMgtInfo

This method is used to update the account management data of a subscriber. Class of Service, Rate Area and Feature Profile must be defined before assigning it to a subscriber.

6.3.87 UpdateSubscriberAccountUserInfo

This method is used to change the user-controlled information of the subscriber.

6.3.88 UpdateSubscriberQOS

This method is used to modify the quality of service data of a subscriber. The QoS profile ID must be defined before associating it with the subscriber. This method allows the individual quality of service attributes but Unify suggests using the QoS profile ID for two reasons. First, in general, a system will have very few profiles defined and it is easier to manage the changes necessary if a profile is assigned to the subscribers. Second, individual QoS parameters such as QoS Primary, QoS Secondary etc. will be removed from the interface in a future release.

6.3.89 UpdateSubscriberCapabilities

This method is used to modify the subscriber's line capabilities such as bearer types.

6.3.90 UpdateKeysetInfo

This method is used to modify the keyset related data.

6.3.91 UpdateSubscriberDN

This method is used to change the DN of a subscriber. The new DN must be available and vacant on the system.

6.3.92 UpdateSubscriberFeatures

This method is used to add, edit, remove or deny features to a subscriber. Each feature would require different data, for more information refer to the *SOAP/XML Subscriber Provisioning Book*. Also some of the features interact with each other, for more information refer to *OpenScape Voice Reference Manual, Volume 3, Feature Description*.

6.3.93 UpdateConnectionInfo

This SOAP request is used to change the subscriber's connection information. Not all fields can be updated. The Connection information can be SIP. The subscriber's current connection cannot be changed from one type of connection to another.

Note: When modifying SIP Security information (Username, Password, or realm); ALL of these parameters must be included in the request. Also note that in order to remove SIP Security information, it is necessary to set the Scheme parameter to "no-security" and not pass any of the other parameters.

6.3.94 UpdateSubscriberFeatureProfile

This method is used to modify the Feature Profile assignment. If the new Feature Profile has the same services that are part of the old Feature Profile and the subscriber has personalized service data associated with these services such as Call Forwarding number, Screening List numbers etc., and the intent of the service provider is to retain the subscriber's personalized data then the parameter PreserveLocPres must be set to True. If not, the default value of the PreserveLocPres is set to False and the entire subscriber's personalized service data will be lost.

6.3.95 AuditFeatureProfileListRequest

This SOAP request is used to execute an audit on the features subscribers have assigned to them, against the features specified in the Feature Profile (FP).

There are times that subscribers does not inherit all the features from the FP they belongs to. If the subscriber contains features conflicting with features in the FP, the subscriber will retain the conflicting features. Since FP assignment to subscribers is handled as a job, any failure during feature propagation to subscribers will be stored in the job-result table (database) for about five days and then will be lost.

1. By choosing audit action "ReportOnly" this API will crosscheck subscriber features with FP-features upon request. It will report either conflicting or missing features in the subscriber with features in the FP. To reinherit missing features go to step 4.
2. In case of conflict the administrator has to manually remove the conflicting feature from the subscriber.
3. The "ReportOnly" audit action can be used again to verify the removal of the feature. This time the administrator will be warned about a missing feature.

4. By choosing audit action "ReInherit" this API will reinherit any features missing from the subscriber which are present in the FP.

6.3.96 AddContactList

This method is used to add contact to a SIP subscriber. The number of contacts that can be associated with a SIP subscriber is determined by an RTP parameter Srx/Xdm/MaxNonKeysetContacts defined at the office level. The default number of contacts allowed is currently set to 5.

6.3.97 DeleteContactList

This method is used to delete the contact of a SIP subscriber.

6.3.98 GetContactList

This method is used to get the list of contacts associated with a SIP subscriber.

6.3.99 CreateQoSProfile

This method is used to create a QoS profile. It is expected that there will be small number of QoS profiles in a system and these are shared by many subscribers. Although there is no hard limit on the number of QoS Profiles that can be created in the system, Unify advises to create no more than 1000.

6.3.100 ModifyQoSProfile

This method is used modify the QoS profile data.

6.3.101 DeleteQoSProfile

This method is used to delete the QoS Profile. The deletion of QoS profile will be rejected with an appropriate error code if any subscriber or BG is using the QoS profile.

6.3.102 GetQoSProfileList

This method is used to retrieve a list of QoS Profiles. The result will be list of QoS profile IDs only. To get details about any specific QoS, GetQoS Profile method must be used. You can limit the scope of the list to a BG or system wide only or BG and system wide.

6.3.103 GetQoSProfile

This method is used to the QoS profile data of a given QoS profile ID.

6.3.104 CreateFeatureProfile

This method is used to create a Feature Profile. Feature Profiles are the feature templates that contain set of features and feature data that can be associated to many subscribers on the system. For example, an operator may have three levels of feature sets defined and marketed such as Platinum, Gold and Silver that contain various combinations of features offered by the system. These feature sets can be directly correlated to the service packages sold to the subscribers and the corresponding feature set will be associated during the creation of the subscriber. Feature Profiles also provide the ability to easily extend the feature packages associated with a large number of associated subscribers instead of modifying each subscriber with the new feature(s).

6.3.105 UpdateFeatureProfile

This method is used to modify the features and feature data that are associated with a Feature Profile. These changes are reflected for all the subscribers that have this Feature Profile associated. The features that are assigned via Feature Profile will not overwrite the discrete assignment of the features at the subscriber level and similarly for feature data. For example, if a Feature Profile having the call forwarding variable feature with a destination DN of 5615551212 is assigned to a subscriber that already has call forwarding variable with the destination DN 561111111, then the subscriber's discretely assigned feature call forwarding variable and call forwarding to number will not be modified and retained as such.

Feature Profile ID, Feature Profile Type and Feature Profile connection type cannot be modified.

6.3.106 DeleteFeatureProfile

This method is used to delete the Feature Profile. Feature Profile deletion will be rejected if the Feature Profile is associated with any subscriber. GetSubscriberList may be used to obtain all the subscribers that have the Feature Profile that is to be deleted. Remove the association of the feature profile from all the subscribers using UpdateSubscriberFeatureProfile and then re-execute this method to successfully delete the Feature Profile.

6.3.107 GetFeatureProfileList

This method is to retrieve the list of all the Feature Profile IDs provisioned on the system.

6.3.108 GetFeatureProfile

This method is used to retrieve the Feature Profile data of a Feature Profile.

6.3.109 CreateMlhg

This method is used to create a Hunt Group. The PilotServiceId is the pilot DN that represents the main DN of the Hunt Group. Any calls coming in to the hunt group will be terminated to this Pilot DN and then a hunt member who is currently idle is chosen to receive the call using the hunt type associated with the Hunt Group. If the Pilot DN is a business group line DN then the Business Group Name must be provided. Services such as Call Forwarding must be assigned using UpdteSubscriberFeatures. For more information on the features allowed refer to *OpenScape Voice Reference Manual, Volume 3, Feature Description*.

6.3.110 ModifyMlhgInfo

This method is used to modify the Hunt Group data. Pilot DN number cannot be modified. For modification of services, use method UpdateSubscriberFeatures. For more information on the features allowed refer to *OpenScape Voice Reference Manual, Volume 3, Feature Description*.

6.3.111 DeleteMlhg

This method is used to delete a given Hunt Group. RemoveMlhgTerms parameter provides an option to remove all the hunt terminals associated with this hunt group. If members still exist in the specified Hunt Group and the RemoveMlhgTerms parameter is set to false (which is the default) the request will be rejected with an appropriate error message.

6.3.112 GetMlhgInfo

This method is used to retrieve the Hunt Group information. Parameter GetMlhgInfoOptionsList retrieves the list of all the hunt members and their attributes also.

6.3.113 AddSubToMlhg

This method is used to add a subscriber or Business Group Line to Hunt Group that is hunt terminals of this hunt group.

6.3.114 ModifyMlhgTermInfo

This method is used to modify the hunt member information. This method also allows moving a hunt member from one hunt group to another hunt group.

6.3.115 DeleteMlhgTerm

This method is used to remove a given hunt member from a given hunt group. This hunt member will still exist as a normal subscriber DN that can make and receive calls. Use method DeleteSubscriber to completely remove this DN from the system.

6.3.116 GetMlhgTMData

This method is used to get the traffic measurement data of a given Hunt Group.

6.4 Technical Implementation Notes

The following sections describe the technical implementation specific to this SDK.

6.4.1 Client Architecture

The ADG for BG and BG Line Provisioning SDK is based on a Client/Server architecture. The Consumer Portal application implements the SOAP Client and the OpenScape Voice system implements the SOAP Server. This ADG describes the interface for building and managing the subscribers on the OpenScape Voice system.

Abschnitt 6.3, “Business Group SOAP/XML Methods”, auf Seite 6-40 provides brief descriptions of the SOAP/XML methods that the Consumer Portal can use to manage the subscribers on the OpenScape Voice system. Since this is based on request and response architecture every method has a response.

6.4.2 Licensing

Any one who purchases and acquires the BG and BG Line Provisioning SDK is allowed to use all of the Web Services offered by this SDK.

6.4.3 Discovery of Web Services

There is no discovery mechanism defined. This web service can be accessed by using the OpenScape Voice system IP address and the port for the SOAP/XML operations. However, Web Services Client's IP address must be set up as a trusted entity on the OpenScape Voice system to ensure the security of the OpenScape Voice system.

6.4.4 Connectivity

The OpenScape Voice system supports SOAP/XML using HTTP. By default the SOAP Server on the OpenScape Voice system is bound to a starting TCP port 8767. The default number of ports on which the SOAP Server accepts requests is incremented sequentially starting with 8767; by default, the valid ports are 8767, 8768, 8769, and 8770. The starting port and number of ports can be modified according to the needs of the client but must match the ports on the SOAP Client (OSS) side as well.

6.4.5 Request-Response Operation

Refer to the *OpenScape Voice, Application Developers Manual: Volume 1, Web Services SDK Programming Overview*.

6.4.6 Security

Network security is ensured by configuring the Web Services Client host IP Address as a trusted port on the OpenScape Voice system.

6.4.7 Sessions

The number of TCP ports configured for SOAP Server use on the OpenScape Voice system determines the number of sessions that can be supported on the OpenScape Voice system. The OpenScape Voice system does not have the knowledge of the client application and, therefore, cannot enforce the TCP port usage for Web Services. Refer to Abschnitt 6.4.4, “Connectivity”, auf Seite 6-62 for the optimum number of SOAP/XML sessions on the OpenScape Voice system.

6.4.8 SOAP Server Restart

The SOAP client is not informed when the SOAP/XML Server restarts. If the SOAP Server restarts, no response will be sent for any outstanding SOAP requests.

6.4.9 SOAP Client Restart

SOAP Server has no knowledge of the Web service client application restart. It is up to the Client to re-establish the connection if it needs to communicate with the SOAP Server.

6.4.10 Heartbeat Mechanisms

The SOAP Server cannot monitor the SOAP Clients. If the SOAP Client requires a heartbeat mechanism with the SOAP Server, it is recommended that the SOAP Client use the "GetVersion" SOAP/XML method because this method will result in a relatively quick response while imparting the least impact on the call processing capabilities of the OpenScape Voice system. Periodically sending a "GetVersion" request and receiving a response would validate the connection between the SOAP Client on the OSS and the SOAP Server on the OpenScape

Voice system. Care must be taken in deciding on the period used to implement the heartbeat. If the period is too short, the performance of the OpenScape Voice system or other SOAP Clients can be impacted.

6.4.11 Error Handling

The errors that can occur during the processing of a SOAP request are described in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

6.4.12 Paging

For the SOAP requests that accomplish the Get List functionality, it is desirable that a mechanism be used to facilitate paging through the result set of the returned data by SOAP Server, such as OpenScape Voice Assistant, that is designed to display a manageable subset of a large list of data to the user. Therefore, the following paging mechanism exists to be used by the various Get List SOAP requests, as necessary.

All the SOAP methods that retrieve a list support the Paging mechanism.

For Get List SOAP requests, a paging structure is defined in the WSDL interface that includes the following fields:

- Page Size — the number of rows shown on a page. This is the number of rows displayed in the GUI. Must be >= 1 and the maximum of 1000.
- Page Number — The page that is shown in the GUI. Must be >= 1 or be equal to -10, which indicates the last page.

Depending on the individual Get List request, there may be additional input parameters, including filter criteria, to apply to the existing objects on the system.

For Get List SOAP requests, a PaginatedInfo structure is defined in the WSDL interface that includes the following fields:

- Number of rows in the result set (total number of entries meeting the filter criteria, if any)
- Number of pages in the result set (total number of pages of data, given the number of rows and the Page Size)
- Page Size (same as Page Size in request)

- Page Number (same as Page Number in request)

Note: Paging is optional in order to make GetList SOAP requests backward compatible. If no paging information is sent, the entire list, up to the maximum possible elements, is returned. The following default behavior applies to paging requests:

- If PageSize is <= 0 or >= the total number of existing elements, no paging is assumed and the entire list is returned.
- If PageNumber == -10, or >= the last page, the last page is displayed (with the starting entry as it is specified with the last page number, so the size of this page may be less than the specified PageSize)
- If PageNumber <= 0, the first page is displayed.

The PaginatedInfo structure in the response will reflect the CurrentPage info according to the default behavior, not according to the actual user input for Paging.

Note: Sorting is a separate issue from Paging and is not addressed in this document.

6.4.13 Unknown Tags

All the SOAP methods have a WSDL version input in the header to ensure only the data defined in that particular WSDL version are returned. It is recommended that the SOAP/XML processing on the SOAP Client side be written in such a manner that it ignores any unknown tags in the responses. This requires turning off the XML validations on the SOAP/Client during the code generation. However, if the SOAP Client does not provide the WSDL version in the SOAP method and SOAP Server in turn returns more data than the SOAP Client expects, ignoring the tags will be very appropriate and does not harm the functionality. If the SOAP Client does not ignore unknown tags then any extra data returned by the SOAP Server may cause an error on the SOAP Client.

6.4.14 SOAP Server Result

For a description of the Result structure returned by the SOAP Server to the SOAP Client, please refer to the chapter named "Subscriber Provisioning Result Information" in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

In case the SOAP Client issues a blank GET to the SOAP Server, it will receive no response.

6.4.15 Versioning Concept

A versioning concept is defined within the SOAP Server in order to control the SOAP/XML interface changes. The SOAP Client application need not align to the interface versioning concept because of the backward compatibility unless the application requires the changes. The backward compatibility is only assured for N-2 product releases.

The possible reasons for Web Services interface modifications are:

- The SOAP/XML interface can change within the lifecycle of a Release due to fault reports and/or change requests. (update)
- The SOAP/XML interface can change between two Releases due to the introduction of new features. (upgrade)

In both cases the SOAP Client application has to be aware of these changes or has to be able to safely ignore them. In this context one has to distinguish between compatible and incompatible interface changes:

- **Compatible Interface Changes**

  Changes in the SOAP interface are compatible if they do not lead to verification failures on the SOAP Client application side.

  If the SOAP Client application was created using code generation tools like Axis (for Java), the code created probably does an xml-schema verification of the incoming SOAP/XML responses. When the verification fails, the operation will likely not be processed.

  A typical compatible change is adding a new XML tag to the SOAP/XML methods. Since the SOAP Client application is not aware of this new tag, it can not request it and therefore does not receive a response it can not handle. If the SOAP Client passes the WSDL version in the requests then responses will always correspond to the requested WSDL.

- **Incompatible Interface Changes**

  Incompatible interface changes are all changes that lead to an xml schema verification failure on the SOAP Client application side. Examples of such changes are:

  – Adding a new tag to a SOAP/XML response

  – Changing an enumeration definition

  – Deleting a SOAP operation

  In general, the SOAP Server will not make any incompatible interface changes like these (within the N-2 product releases at least).

In order to track changes of the SOAP/XML interface, a product version is being defined which is returned in the "getInterfaceVersion" SOAP/XML operation. This product version consists of the following components:

<Release>.<Version>.<Build>.<Internal Revision>

- Release: This is the general release version of the OpenScape Voice product. This will not change for SOAP adaptations within a release.
- Version: This is a point release version of the OpenScape Voice product. This will not change for SOAP adaptations within a release.
- Build: This identifies the WSDL version. Any changes to this number indicate the SOAP/XML interfaces have been changed.
- Internal Revision: This identifies the internal changes to the WSDL version. These changes do not have any impact to the existing interfaces.

6.4.16 WSDL Parser and Compiler

Currently Unify uses gSOAP and Axis to parse the WSDL and compiler to generate the stubs and skeletons to invoke the service or build a new service based on the WSDL. However, there are some customers that use different tools such as Web logic to generate their code using the Unify Produced WSDL files. If the customer chooses to use any other tools than the gSOAP and Axis, additional support may be required.

All the WSDL files contain the following lines:

```
<port name="siemens-hiq8000" binding="tns:siemens-hiq8000">
<SOAP:address location="http://www.siemens.com/cgi-bin/soapServer"/>/
port>
```

Note that the location field is not used.

6.5 BG Management Interface Details

A detailed description of the BG Management interface is provided in the *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

6.6 Business Group Management SDK Usage Examples

This section outlines at the high level some typical usage scenarios. The level of detail is down to the parameter level of the requests and their responses.

6.6.1 SOAP/XML Examples

The attached zip file illustrates SOAP/XML examples for GetSubInfoByRel, GetVersion, and UpdateSubscriberFeatures request and response.

As explained earlier, the SOAP Server provides a backward compatibility for N-2 product releases, so even though the examples were developed using a V3 WSDL file, the examples also run in a V3.1 system.

Refer to the included file "ReadMe.txt" for an explanation of the examples provided.

6.6.2 Application Code Examples

The attached zip file illustrates sample Java code examples. It contains examples for GetSubInfoByRel, GetVersion and UpdateSubscriberFeatures methods.

6.6.3 Validation Application Installation and Configuration

The Business Group Provisioning SDK currently does not include any "Validation Application." Client Applications must be connected to a OpenScape Voice system for proper validation testing.

7 About the Subscriber Self Care SDK

The Subscriber Self Care (SSC) Management Software Development Kit (SDK) can be used to write or enhance portal application(s) in conjunction with the OpenScape Voice system Linux server offered functionalities.

This Application Developer Guide (ADG) is one component of the Subscriber Self Care Provisioning SDK, which is intended for use by the customer to create and test custom applications. The SSC SDK consists of a set of components (Bild 11) as defined below:

- Selfcare WSDL File
- SDK Specific Documentation (Subscriber Self Care ADG - this document)
- Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning document
- Sample applications: Java sample code illustrating some basic provisioning and the respective executables to test the methods GetSubInfoByRel, GetVersion and UpdateSubscriberFeatures.
- Sample SOAP/XML messages for GetSubInfoByRel, GetVersion and UpdateSubscriberFeatures.

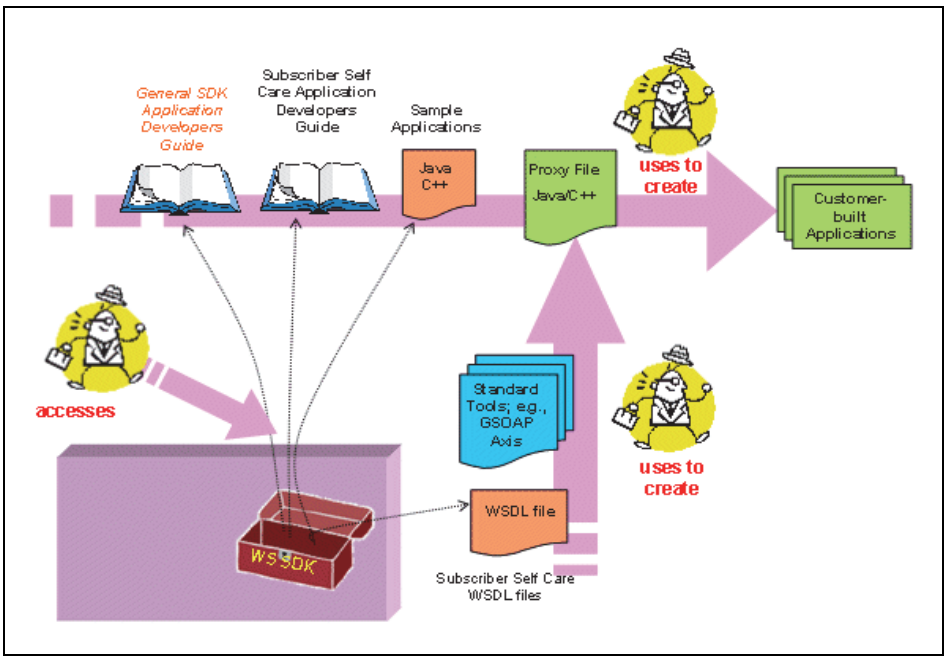


Figure 11 Components of Subscriber Self Care SDK

Attention: General information that applies to all Unify Software Development Kits (SDKs) is described in the *OpenScape Voice, Application Developers Manual: Volume 1, Web Services SDK Programming Overview*. Refer to that guide for information about the SDK package content, distribution, and other general topics.

7.1 SOAP/XML Concept

The Simple Object Access Protocol (SOAP) is a protocol for exchanging XML-based messages over computer networks, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework upon which more abstract layers can be built.

The SOAP Server is an application that hosts the server software, defines the WSDL and methods to process the requests. The SOAP Server is an integrated component of the OpenScape Voice system software.

The SOAP Client is an application that hosts the client software and sends in the data via methods to provision the objects such as subscriber feature data on the switch.

The ADG for Subscriber Self Care Provisioning SDK is based on the Client/Server architecture. The Consumer Portal application hosts the SOAP Client and the OpenScape Voice system hosts the SOAP Server. This ADG describes the web based interface for self-care administration.

SOAP/XML interfaces provided by the OpenScape Voice system are backward compatible for N-2 software releases, N being the current release.

All of the SOAP/XML methods are modified to confirm the standards starting in V3.0 to allow one input parameter and one output parameter. Hence, all of the method names end with 2. This ADG does not explicitly state exact syntactical method names and instead they must be perceived as functional descriptions. To get correct syntactical method names, refer to the WSDL file. In addition, the code generation tools ensure that correct method names are used.

7.2 Subscriber Self Care Concept

The Subscriber Selfcare Provisioning SDK can be used to write or enhance portal application(s) in conjunction with the functionalities offered by the OpenScape Voice system server. The Web services for the Subscriber Self Care interface provides users with functions to allow self-administration (e.g., changing the call forwarding status of their own phone). A typical customer can be a service provider offering a web-portal to end users to facilitate their personal phone usage and administration.

This Application Developer Guide (ADG) enables customers to enhance their portal application by accessing the Subscriber Selfcare Provisioning management web service interfaces. Refer to Bild 12. Although this ADG enables customers to enhance their applications by accessing Subscriber Self Care management web service interfaces, it does not provide end-user authentication nor authorization functions. Those are expected to be handled by the portal application using their own databases.

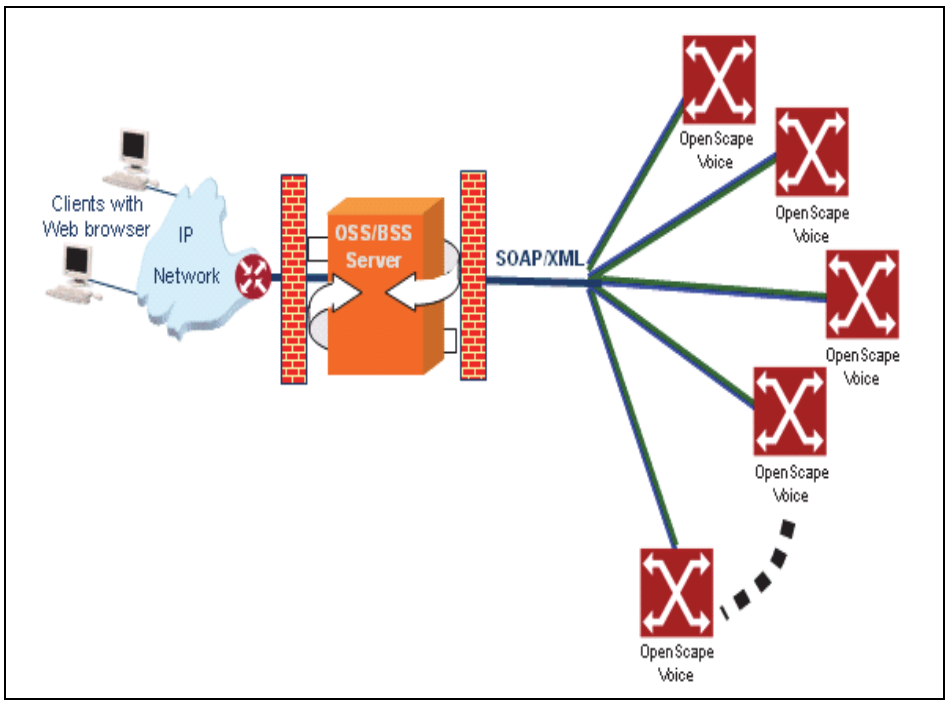


Figure 12 Self Care Provisioning Topology

Typically, Service Providers let the end users view and manage the status/data of their subscribed features. For example, a Service Provider will assign features such as Call Forwarding Variable and let the end user manage the activation and the forward to number. Since the OpenScape Voice system will also allow the assignment of features and many other attributes, it is the portal application provided by the Service provider that will determine what is allowed by the end users to be seen and modifiable. Hence, the portal application will determine the display to an end customer.

7.3 Subscriber Self Care SOAP/XML Methods

The following subsections provide brief descriptions of the SOAP/XML methods that the Service provider portal application will use to get and set the data on the OpenScape Voice system to support subscriber self care functionality.

7.3.1 GetInterfaceVersion

This SOAP/XML request is used to get the OpenScape Voice system build and release information including the WSDL version such that the Portal may know what interface methods are available on the system. In the Client/Server architecture, Client must never operate higher WSDL version than the WSDL version on the Server because the Server will not be able to recognize the data in the SOAP/XML method such as new tags/data. For example, if the Client application is communicating with more than one OpenScape Voice system that operate on different releases having different WSDL versions, it is necessary that the SOAP Client is operating at the older version of WSDL. Since the OpenScape Voice system is N-2 releases backward compatible, SOAP Client must be upgraded last in the network of OpenScape Voice system(s) and client(s).

7.3.2 GetInterfaceVersionResponse

A successful response contains the WSDL version, the OpenScape Voice load build version, the Environment of the system (Enterprise) and the timestamp when the SOAP Server process was last built.

7.3.3 GetSubscriberInfoByRel Request

This method retrieves all the subscriber's data including features, account user info etc. Service ID which is same as end user DN is a required parameter. The data to be retrieved is managed by setting the attribute GetSubscriberDataOption in the method. For example, to retrieve only the list of features assigned to the subscriber attribute IncludeListOfFeatures must be set. If no attribute is set then all the data pertaining to the subscriber will be returned. It may be simpler to the client application to get all the subscriber data instead of choosing the data that might differ from one end uer to another, and display only the data that must be visible to the end user.

7.3.4 GetSubscriberInfoByRel response

If Service ID is valid subscriber DN, then all the data attributes set in the Subscriber Data Options parameter determines the data returned. If none are set all the end user data will be returned back to the client application.

7.3.5 GetSubTranStatus

The GetSubTranStatus SOAP request is used to retrieve the transient status of a subscriber (the DN supplied in the Service ID). The information about the current status of the specified subscriber is obtained from the PDM (Port Data Manager) component.

7.3.6 UpdateSubscriberFeatures request

This method is used to add, edit, remove or deny features to a subscriber. Each feature would require different data, for more information refer to the *SOAP/XML Subscriber Provisioning Book*. Also some of the features interact with each other, for more information refer to *OpenScape Voice Reference Manual, Volume 2, Feature Description*.

7.3.7 UpdateSubscriberFeatures response

Depending on the operation being successful or not, various responses are returned.

7.4 Technical Implementation Notes

The following sections describe the technical implementation specific to this SDK.

7.4.1 Client Architecture

The ADG for Subscriber Selfcare Provisioning SDK is based on a Client/Server architecture. The Consumer Portal application implements the SOAP client and the OpenScape Voice system implements the SOAP server. This ADG describes the web based interface for managing the subscribers on the OpenScape Voice system.

Abschnitt 7.3, “Subscriber Self Care SOAP/XML Methods”, auf Seite 7-72 provides brief descriptions of the SOAP/XML methods that the Consumer Portal can use to manage the subscribers on the OpenScape Voice system. Since this is based on request and response architecture every method has a response.

7.4.2 Licensing

Any one who purchases and acquires the Subscriber Self Care Provisioning SDK is allowed to use all of the Web Services offered by this SDK.

7.4.3 Discovery of Web Services

There is no discovery mechanism defined. This web service can be accessed by using the OpenScape Voice system IP address and the port for the SOAP/XML operations. However, the Web Services Client's IP address must be set up as a trusted entity on the OpenScape Voice system to ensure the security of the OpenScape Voice system.

7.4.4 Connectivity

The OpenScape Voice system supports SOAP/XML using HTTP. By default the SOAP Server on OpenScape Voice system is bound to a starting TCP port 8767. The default number of ports on which the SOAP Server accepts requests is incremented sequentially starting with 8767; by default, the valid ports are 8767, 8768, 8769, and 8770. The starting port and number of ports can be modified according to the needs of the client but must match the ports on the SOAP Client (Consumer Portal) side as well.

7.4.5 Request-Response Operation

Refer to the *OpenScape Voice, Application Developers Manual: Volume 1, Web Services SDK Programming Overview*.

7.4.6 Security

Network security shall be ensured by configuring the Web Services Client host IP Address as trusted port on the OpenScape Voice system.

7.4.7 Sessions

The SOAP Server is sessionless. Each SOAP request received is independent of all others.

7.4.8 SOAP Server Restart

The SOAP client is not informed when the SOAP/XML Server restarts. If the SOAP Server restarts, no response will be sent for any outstanding SOAP requests.

7.4.9 SOAP Client Restart

SOAP Server has no knowledge of the Web service client application restart. It is up to the Client to re-establish the connection if it needs to communicate with the SOAP Server.

7.4.10 Heartbeat Mechanisms

The SOAP Server cannot monitor the SOAP clients. If the SOAP Client requires a heartbeat mechanism with the SOAP Server, it is recommended that the SOAP client use the "GetVersion" SOAP/XML method because this method will respond the quickest while imparting the least impact on the call processing capabilities of the OpenScape Voice system. Periodically sending a "GetVersion" request and receiving a response would validate the connection between the SOAP Client on the Consumer Portal and the SOAP server on OpenScape Voice system. Care must be taken in deciding on the period used to implement the heartbeat, if the period is too short and the performance of the OpenScape Voice system or other SOAP clients can be impacted.

7.4.11 Error Handling

The errors that can occur during the processing of a SOAP request are described in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

7.4.12 Paging

For the SOAP requests that accomplish the Get List functionality, it is desirable that a mechanism be used to facilitate paging through the result set of the returned data by SOAP Servers, such as the OpenScape Voice Assistant, that display the data graphically to the user. Therefore, the following paging mechanism exists to be used by the various Get List SOAP requests, as necessary.

All the SOAP methods that retrieve a list support the Paging mechanism.

For Get List SOAP requests, a paging structure is defined in the WSDL interface that includes the following fields:

- Page Size — the number of rows to be shown on a page. This is the number of rows displayed in the GUI. Must be >= 1 and the maximum of 1000.
- Page Number — The page that will be shown in the GUI. Must be >= 1 or be equal to -10, which indicates the last page.

Depending on the individual Get List request, there may be additional input parameters, including filter criteria, to apply to the existing objects on the system.

For Get List SOAP requests, a PaginatedInfo structure is defined in the WSDL interface that includes the following fields:

- Number of rows in the result set (total number of entries meeting the filter criteria, if any)
- Number of pages in the result set (total number of pages of data, given the number of rows and the Page Size)
- Page Size (same as Page Size in request)
- Page Number (same as Page Number in request)

Note: Paging is optional in order to make GetList SOAP requests backward compatible. If no paging information is sent, the entire list, up to the maximum possible elements, is returned. The following default behavior applies to paging requests:

- If PageSize is <= 0 or >= the total number of existing elements, no paging is assumed and the entire list is returned.

- If PageNumber == -10, or >= the last page, the last page is displayed (with the starting entry as it is specified with the last page number, so the size of this page may be less than the specified PageSize)
- If PageNumber <= 0, the first page is displayed.

The PaginatedInfo structure in the response will reflect the CurrentPage info according to the default behavior, not according to the actual user input for Paging.

Note: Sorting is a separate issue from Paging and is not addressed in this guide.

7.4.13 Unknown Tags

All the SOAP methods have a WSDL version input in the header to ensure only the data defined in that particular WSDL version are returned. It is recommended that the SOAP/XML processing on the SOAP Client side be written in such a manner that it ignores any unknown tags in the responses. This requires turning off the XML validations on the SOAP/Client during the code generation. However, if the SOAP Client does not provide the WSDL version in the SOAP method and SOAP Server in turn returns more data than SOAP Client expects, ignoring the tags will be very appropriate and does not harm the functionality. If the SOAP Client does not ignore then any extra data returned by the SOAP Server may cause an error on the SOAP Client.

7.4.14 SOAP Server Result

For a description of the Result structure returned by the SOAP Server to the SOAP Client, please refer to the chapter named "Subscriber Provisioning Result Information" in *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

In case the SOAP Client issues a blank GET to the SOAP Server, it will receive no response.

7.4.15 Versioning Concept

A versioning concept is defined within the SOAP Server in order to control the SOAP/XML interface changes. The SOAP Client application need not align to the interface versioning concept because of the backward compatibility unless the application is requiring the changes. The backward compatibility is only assured for N-2 product releases.

The possible reasons for Web Services interface modifications are:

- The SOAP/XML interface can change within the lifecycle of a Release due to fault reports and/or change requests. (update)
- The SOAP/XML interface can change between two Releases due to the introduction of new features. (upgrade)

In both cases the SOAP Client application must be aware of these changes or must be able to safely ignore them. In this context, one has to distinguish between compatible and incompatible interface changes:

- **Compatible Interface Changes**

  Changes in the SOAP interface are compatible if they do not lead to verification failures on the SOAP Client application side.

  If the SOAP Client application was created using code generation tools like Axis (for Java), the code created probably does an xml-schema verification of the incoming SOAP/XML responses. When the verification fails, the operation will likely not be processed. Changes in the SOAP interface are compatible if they do not lead to verification failures on the SOAP Client application side.

  A typical compatible change is adding a new XML tag to the SOAP/XML methods. Since the SOAP Client application that does not know about this change is not aware of this new tag, it can not request it and therefore does not receive a response it can not handle. If the SOAP Client passes the WSDL version in the requests then responses will always correspond to the requested WSDL.

- **Incompatible Interface Changes**

  Incompatible interface changes are all changes that lead to an xml schema verification failure on the SOAP Client application side. Examples of such changes are:

  – Adding a new tag to a SOAP/XML response

  – Changing an enumeration definition

  – Deleting a SOAP operation

  In general, the OpenScape Voice SOAP Server will not make any incompatible interface changes like these (within the N-2 product releases at least).

In order to track changes of the SOAP/XML interface, a product version is being defined which is returned in the "getInterfaceVersion" SOAP/XML operation. This product version consists of the following components:

<Release>.<Version>.<Build>.<Internal Revision>

- Release: This is the general release version of the OpenScape Voice product. This will not change for SOAP adaptations within a release.

- Version: This is a point release version of the OpenScape Voice product. This will not change for SOAP adaptations within a release.
- Build: This identifies the WSDL version. Any changes to this number indicate the SOAP/XML interfaces have been changed.
- Internal Revision: This identifies the internal changes to the WSDL version. These changes do not have any impact to the existing interfaces.

7.4.16 WSDL Parser and Compiler

Currently, Unify uses gSOAP and Axis to parse the WSDL and compiler to generate the stubs and skeletons to invoke the service or build a new service based on the WSDL. However, there are some customers who use different tools such as Web logic to generate their code using the Unify produced WSDL files. If the customer chooses to use any other tools than the gSOAP and Axis, additional support may be required.

All the WSDL files contain the following lines:

```
<port name="siemens-hiq8000" binding="tns:siemens-hiq8000">
<SOAP:address location="http://www.siemens.com/cgi-bin/soapServer"/>/
port>
```

Note that the location field is not used.

7.5 Subscriber Self Care Interface Details

A detailed description of the Subscriber Self Care interface is provided in the *Interface Manual: Volume 2, SOAP/XML Subscriber Interface Provisioning*.

7.6 Subscriber Self Care SDK Usage Examples

This section outlines at the high level some typical usage scenarios. The level of detail is down to the parameter level of the requests and their responses.

7.6.1 SOAP/XML Examples

The attached zip file illustrates SOAP/XML examples for GetSubInfoByRel, GetVersion, and UpdateSubscriberFeatures request and response.

As explained earlier, the SOAP Server provides a backward compatibility for N-2 product releases, so even though the examples were developed using a V3 WSDL file, the examples also run in a V3.1 system.

Refer to the included file "ReadMe.txt" for an explanation of the examples provided.

7.6.2 Application Code Examples

The attached zip file illustrates sample Java code examples. It contains examples for GetSubInfoByRel, GetVersion and UpdateSubscriberFeatures methods.

7.6.3 Validation Application Installation and Configuration

The Subscriber Selfcare Provisioning SDK currently does not include any "Validation Application." Client Applications must be connected to a OpenScape Voice system for proper validation testing.

Index

T

U

W

X