



Web Services API – Programmer's Guide

(version 2.4.41.13)

Disclaimer

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR WYDE VOICE REPRESENTATIVE FOR A COPY.

IN NO EVENT SHALL WYDE VOICE OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF WYDE OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright

Except where expressly stated otherwise, the Product is protected by copyright and other laws respecting proprietary rights. Unauthorized reproduction, transfer, and or use can be a criminal, as well as civil, offense under the applicable law.

WYDE Voice and the WYDE Voice logo are registered trademarks of WYDE Voice LLC in the United States of America and other jurisdictions. Unless otherwise provided in this Documentation, marks identified with “R” / ®, “TM” / ™ and “SM” are registered marks; trademarks are the property of their respective owners.

For the most current versions of documentation, go to the WYDE support Web site:
<http://docs.wydevoice.com/>

November 11, 2011

Symbols and Notations in this Manual

The following notations and symbols can be found in this manual.



Denotes any item that requires special attention or care. Damage to the equipment or the operator may result from failure to take note of the noted instructions

Figure	Denotes any illustration
Table	Denotes any table
Text	Denotes any text output
<i>Button</i>	Denotes any button caption

Table of Contents

Symbols and Notations in this Manual	3
Table of Contents	4
Tables List	7
Figures List	8
Chapter 1: Introduction.....	9
Assumed Skills	9
Web Services	10
Definitions	10
Chapter 2: Data Structures.....	14
General Data Structure	14
Data Classes (Entities).....	16
Subscriber	16
Conference Account – Conference User (Confuser).....	16
Conference Info (ConfInfo).....	17
DNIS	18
DNIS Alias (DnisAlias).....	18
Call Flow (CallFlow).....	18
Attribute.....	19
Conference.....	19
Operator Status (OperatorStatus).....	20
ConferenceDR	21
Polling Result (PollingResult).....	21
Operator's Statistic (OperatorStatistic)	21
Session	22
SessionDR	23
DTMF Event (DtmfEvent)	24
Subscriber Conference (SubscriberConference)	24
Dialout Subscriber (DialoutSubscriber)	24
Chapter 3: Samples of Functions.....	25
WYDE Web Services Initialization.....	25
Sample of WYDE Web Services Initialization	25
Web Methods' XML Requests and Responses	25
Sample of XML for Function with Multiple Parameters Sent and List of Objects Received	25
Sample of XML for Function with the Object Parameter Sent and the Object Received	26
Subscribers Management.....	26
Sample of Subscriber and his Conference Accounts Creation	26
Sample of Subscribers Filtering, Modifications, Conference Accounts Modifications	26
Sample of Subscribers Filtering and Deletion.....	27
Sample of Getting Conference Users Information	27
Conferences and Calls Management	27
Sample of Conferences Filtering, Changes Secure Mode, Dropping the Conferences	27

Sample of Placing the Entire Conference on Hold, Starting and Stopping Q&A Sessions and Conference Recording	28
Sample of Conference Polling Sessions	29
Sample of Calls Filtering, Mute the Calls, Dropping the Calls	29
Sample of Setting Custom Name and Placing Calls on Hold	29
CDRs Management	30
Sample of Getting Conferences Historical Information	30
Sample of the Shared Recording Generation	30
Sample of Getting Calls Historical Information	31
Sample of Historical Calls Filtering	31
Active Speaker Notification	31
Storage Library	33
Folders Structure and Showing Folder Content for Conference and Bridge Files	34
File upload	36
Files management	36
Chapter 4: Function Reference	38
Subscribers Management	38
Subscribers' Conference Users Management	41
Conference Info Management	44
Conferences and Calls Management	46
Subscribers' Conferences Management	60
CDRs Management	63
Call Flow and DNIS Management	72
Backend and Frontend Services Management	77
Exceptions	78
Constants	79
Appendix A: Code Samples	81
WYDE Web Services Initialization	81
Sample of WYDE Web Services Initialization	81
app.config	83
Web Methods' XML Requests and Responses	84
Sample of XML Request for Function with Multiple Parameters Sent	84
Sample of XML Response for Function with List of Objects Received	85
Sample of XML Request for Function with the Object Parameter Sent	87
Sample of XML Response for Function with the Object Received	88
Subscribers Management	93
Sample of Subscriber and his Conference Accounts Creation (Sample_ManageSubscriber1)	93
Sample of Subscribers Filtering, Modifications, Conference Accounts Modifications (Sample_ManageSubscriber2)	96
Sample of Subscribers Filtering and Deletion (Sample_ManageSubscriber3)	99
Sample of Getting Conference Users Information (Sample_ManageConfuser1)	100
Conferences and Calls Management	103
Sample of Conferences Filtering, Changes Secure Mode, Dropping the Conferences (Sample_ManageConference1)	103
Sample of Placing the Entire Conference on Hold, Starting and Stopping Q&A Sessions and Conference Recording (Sample_ManageConference2)	106

Sample of Conference Polling Sessions (Sample_ManageConference3)	109
Sample of Calls Filtering, Mute the Calls, Dropping the Calls (Sample_ManageCall1)	111
Sample of Setting Custom Name and Placing Calls on Hold (Sample_ManageCall2)	114
CDRs Management	116
Sample of Getting Conferences Historical Information (Sample_InfoConferenceDR1)	116
Sample of the Shared Recording Generation (Sample_InfoConferenceDR2)	118
Sample of Getting Calls Historical Information (Sample_InfoSessionDR1)	120
Sample of Historical Calls Filtering (Sample_InfoSessionDR2)	123
Appendix B: Support Resources	125
Support Documentation	125
Web Support	125
Telephone Support	125
Email Support	125

Tables List

Table 1: Properties of Subscriber	16
Table 2: Properties of Confuser.....	17
Table 3: Properties of ConfInfo.....	17
Table 4: Properties of DNIS	18
Table 5: Properties of DnisAlias	18
Table 6: Properties of CallFlow	18
Table 7: Properties of Attribute.....	19
Table 8: Properties of Conference.....	20
Table 9: Properties of OperatorStatus	20
Table 10: Properties of ConferenceDR	21
Table 11: Properties of PollingResult.....	21
Table 12: Properties of OperatorStatistic	21
Table 13: Properties of Session	22
Table 14: Properties of SessionDR.....	23
Table 15: Properties of DtmfEvent.....	24
Table 16: Properties of SubscriberConference.....	24
Table 17: Properties of DialoutSubscriber	24

Figures List

Figure 1: The Web Services Architecture	10
Figure 2: The UML Class Diagram.....	15
Figure 3: Folder Content Sample	34

Chapter 1: Introduction

WYDE conferencing bridges (like SB-HD100, SB-HD1000, and SB-HD10000) provide different API that allow manage conferences and calls, configure subscribers and their conference account, maintain DNIS and call flow management. The basic APIs are

- web services API,
- RT (real time) interface,
- different adapters, for instance
 - billing adapter that allow writing calls and conferences information to an external database,
 - authentication adapter that allow user authentication based on external database), etc.

This document is programmer's guide for the web services API only. Other APIs are being described in the separate documentation.

Please note that if call flow is setup to use external authentication server (like RADIUS) user management API should not be used.

WYDE web services API is designed to query and manage calls and conferences happening on the bridge, manage subscribers and their conference accounts. Through the API you also can manage users and access code used for local authentication. API helps to get information not only in real time mode, but also happened in the past.

The URL for the WYDE web services is <http://<Wyde bridge domain>/dnca/jAdmin?wsdl>. In some languages to point to WYDE web services you may need to use URL without “?wsdl” suffix: <http://<Wyde bridge domain>/dnca/jAdmin>. Here <WYDE bridge domain> is either the registered domain name or IP address that gives the destination location for the WYDE web services URL. For instance the possible WYDE web services URLs could be <http://dnca0.freeconferencecall.com/dnca/jAdmin?wsdl> or <http://192.168.1.5/dnca/jAdmin?wsdl>.



This Web Service Interfaces – Programmer's Guide is based on WYDE web services API version **2.4.41.13**. If you use another version of API the same functions may be different and you may need other version of the guide.

You can check the version of your software using the following URL: <http://<Wyde bridge domain>/version.html>. For instance the possible WYDE software version URLs could be <http://dnca0.freeconferencecall.com/version.html> or <http://192.168.1.5/version.html>.

Assumed Skills

This programmer's guide assumes you have a working knowledge of the following technologies and skills:

- PC usage

- System administration
- Programming basics (in some kind of programming languages)
- Understanding of object-oriented classes structure, UML basics
- VOIP basics
- TCP/IP networking
- Web Administration Interface – User Guide

Web Services

Formal Web Service definition is given by World Wide Web Consortium (W3C) – the main international standards organization for the World Wide Web. According to W3C, a web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web services architecture is shown on Figure 1.

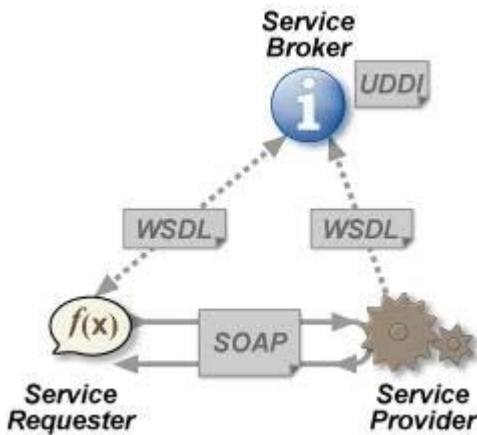


Figure 1: The Web Services Architecture

Web services are platform independent. Web services are based on open standards and protocols. Web services are supported by most major software vendors and industry analysts. You can access WYDE web services from different platforms and from different programming languages.

The detail information about web services can be read in the following articles:

- Web Services Architecture – <http://www.w3.org/TR/ws-arch/>
- Web Services Activity – <http://www.w3.org/2002/ws/>
- Web Services Glossary – <http://www.w3.org/TR/ws-gloss/>

Definitions

In order to discuss the WYDE web services API effectively, we need to have a common set of terminology. For this purpose, we should define the dictionary for the terms you will see throughout this programmer's guide:

- **Class** – A programming language construct that is used as a template to create objects of that class. This template describes the state and behavior that the objects of the class all share. An object of a given class is called an instance of the class. The class that contains that instance can be considered as the type of that object. The classes that are designed in the web services API are Subscriber, Call Flow (CallFlow), DNIS, Conference Account/User (Confuser), Attribute, Conference Info (ConfInfo), Session, SessionDR, Conference, ConferenceDR.
- **Identifier** – A unique key to uniquely identify each instance of the class. In WYDE web services API data structure, the identifier is the single property value, usually it is numeric (long) identifier (ID). Identifier can be used to retrieve information about the single instance of the class; the WYDE web services API contains methods get<Class> (for instance getSubscriber, getDNIS, etc.) that are used to get single instance of the class using the transferred parameter – the identifier of the object instance.
- **Reference Identifier** – A referential constraint between two classes that is used to join the classes. The reference identifier identifies a column or a set of columns in one (referencing) class that refers to a column or set of columns in another (referenced) class. The columns in the referencing class must be the identifier. The values in the referencing columns of one class instance must occur in a single instance in the referenced class; an instance in the referencing class cannot contain values that don't exist in the referenced class. In other words these constructs are being used to join the classes and the instances of these classes. For instance Confuser class has reference identifier subscriberId; the values of this attribute allow join different Conference User objects with Subscribers, who own these Conference Users.
- **Subscriber** – A real person, he has a name, phone number, e-mail address, etc. The subscriber can have conference accounts, he does not have access codes, but access codes are properties of conference accounts that have subscribers. Note that non-admin (non-operator) subscribers can see only “own” information, i.e. his information and information that belongs to subscribers created by him, he can see only their calls, conferences, the reports will show only their data, etc.
To describe subscribers web services API has the class Subscriber; the identifier of this class is subscriberId; the following classes have reference identifiers to the Subscriber class: Confuser, Session, SessionDR, i.e. they are joined with Subscribers; Subscribers can own conference accounts (conference users) information.
- **PIN** – The login ID for the subscriber (must be unique). It can be used either as login in Web Administration Interface (in this case it can be either number or alpha-numeric) or as login for some call flows (in this case must be numeric) for participants authorization.
- **Conference Account** – The element of subscriber conferences configuration. Conference accounts always belong to subscriber. It is being used to define a person in a conference with a particular role (e.g. host, participant, listener, etc.), the DNIS number that should be used to call to the conference, and the access code that should be entered by the user that called to the conference DNIS to determine his role. A subscriber could be a host user in one conference and a listener in another. Conference accounts with the same conference number represent single conference setup.

To describe conference accounts web services API has the class Confuser (Conference User); the identifier of this class is confuserId; this class has reference identifier to the following classes: Subscriber, DNIS, ConfInfo, and set of Attributes.

- **DNIS** – A unique set of numbers that is outputted by a phone carrier that indicates the intended destination for a particular call. It can be any length digits (although usually 10 digits). DNIS is the property of the conference account, but different DNIS numbers can be used to connect to the same conference.

To describe DNIS web services API has the class DNIS; the identifier of this class is dnisId; this class has reference identifier to the CallFlow classes and set of Attributes; the Confuser class has reference identifier to the DNIS class.

- **Access Code** – A numeric code unique for DNIS that allows a host or participant or listener access to a conference call. When users call to DNIS number they being asked to enter their access code. The access code determines the conference and the user role in the conference. Different access codes can determine the same conference, for instance one access code can determine the connected user has host role, another access code can determine that connected user has participant role, and another access code can determine that connected user has listener role.
- **Host** – A user in the conference call that can make changes to the system while the conference call is in progress. Like change the security setting, change who can talk or answer, etc. Sometimes the host user is called moderator. This user role is defined in conference account.
- **Participant** – A person in the conference who can actively participate in a call by both talking and listening. This user role is defined in conference account.
- **Listener** – A person in the conference who can hear the conference call, but cannot speak. Their audio path is one way only (receive). This user role is defined in conference account.
- **Conference Number** – A unique external conference number. Conference number is the property of conference account. If the conference accounts have the same conference number all these accounts determine one single conference. For instance the user can create one conference account record that determine host role, another conference account record that determine participant role, and another conference account record that determine listener role – all these records should have the same conference number to determine one unique conference.

To represent unique conference (conference number) web services API has the class ConfInfo (Conference Info); the identifier of this class is conferenceNumber; the following classes have reference identifiers to the ConfInfo class: Confuser, Session, SessionDR, Conference, ConferenceDR, i.e. they are joined with specific conference information.

- **Conference ID** – A unique conference ID that represents the instance of a conference. When any conference is being started it receives unique conference ID, and all calls to this conference have the same conference ID; if this conference has been completed and another conference is being started that conference will receive another conference ID. Conference ID is normally not exposed to users, unless on the reports.
- **Call Flow** – A unique conference service setup, the logic that is used to process the conference calls. This is the process a call goes through from call setup to, to processing, to call tear down. It includes the logic, DTMF key-presses used, functions,

and the recorded prompts. There are two basic call flow categories: call flows without authentication and call flows with authentication.

To describe call flows web services API has the class `CallFlow`; the identifier of this class is `callflowId`; this class has a set of `Attributes`; the `DNIS` class has reference identifier to the `CallFlow` class.

- **Attribute** – In terms of WYDE web services API, a data structure is used to carry attributes for call flow (`CallFlow`), `DNIS` and conference user (`Confuser`). The attributes skeleton is defined by call flow; other attributes can only override some of them, so for instance when a user called in to the conference `DNIS` it gets attributes exposed by the call flow, but some of these attributes can be already altered by the `DNIS`. Each attribute has name, type, value, and role. The names of the attributes are unique; `CallFlow`, `DNIS`, and `Confuser` classes have a set of `Attribute` objects associated with them.

- **Conference** – A data structure is used to describe ongoing conference on the bridge. Objects of this type are only created by server. User may fetch these objects by calling appropriate function. When conference is over the conference object is deleted by the server.

The conference object is identified by the `conferenceId` property value, this is a globally unique identifier that represents the instance of a conference; this class has reference identifier to a `ConfInfo` class (conference number); `SessionDR` class has reference identifier to the `Conference` class.

- **ConferenceDR** – A data structure is used to describe conference which is already terminated on the on the bridge. User can not directly create this object. The conferenceDR object is identified by the `conferenceId` property value; this class has reference identifier to a `ConfInfo` class (conference number).
- **Session** – A data structure represents a single ongoing call on the server. User can not directly create this object. When the call is over server automatically deletes this object. Normally this data structure is used to get information about call attributes like calling/called number etc., or do something with the call, for instance mute, hang, hold etc.

The identifier of the `Session` class is `sessionId`; this class has reference identifiers to `Subscriber` and `ConfInfo` classes.

- **SessionDR** – A data structure represents a single call on the server which is already terminated on the on the bridge. User can not directly create this object. The identifier of the `SessionDR` class is `sessionId`; this class has reference identifiers to `Subscriber`, `ConfInfo`, and `Conference` classes.

Chapter 2: Data Structures

General Data Structure

The class diagram, data classes (entities) and relations between them are shown on Figure 2. Boxes on this figure are representing data classes (entities), these classes will be described in the next section of this guide; **names of the classes** are shown in bold, **identifiers** are shown in blue color, **reference identifiers** are shown in green color, **encapsulated properties** are shown in brown color, references (relations) between classes are shown with black solid arrows, encapsulations (aggregations) between classes are shown with brown dash lines ended with diamonds, related class data (data that can be retrieved using the related class identifier) are shown with brown dotted lines ended with diamonds. Classes and fields added in the version 2.x are shown **highlighted (turquoise)**.

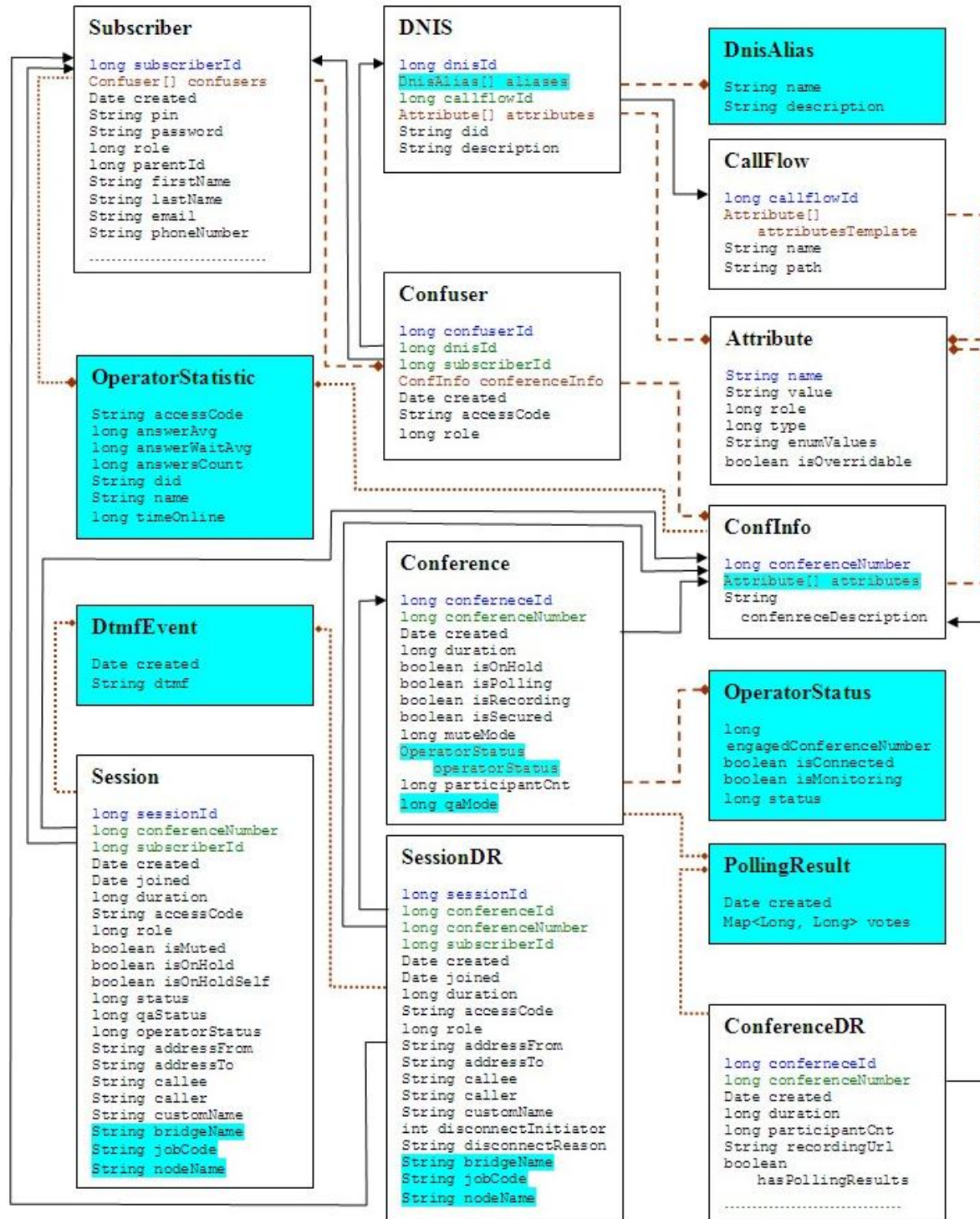


Figure 2: The UML Class Diagram

Data Classes (Entities)

Subscriber

This data structure holds information about subscribers. Subscriber is a real person; he has a name, phone number, e-mail address, etc. The subscriber can have conference accounts, he does not have access codes, but access codes are properties of conference accounts that have subscribers. Subscribers should make a hierarchy – that is why each subscriber has reference to another subscriber who created it. Subscriber which doesn't have a parent - called Administrator. Note that non-admin (non-operator) subscribers can see only “own” information, i.e. his information and information that belongs to subscribers created by him, he can see only their calls, conferences, the reports will show only their data, etc.

Table 1: Properties of Subscriber

String address1	Subscriber's address
String address2	
String city	Subscriber's city
Confuser[] confusers	List of confusers this subscriber associated with. It can be populated by user during subscriber
String country	
Date created	Date when record is created; assigned by the server
String details	Any additional details
String email	Subscriber's e-mail
String firstName	Subscriber real first name (*)
String lastName	Subscriber real last name (*)
long parentId	ID of parent subscriber (*)
String password	password for the logging in to the web interface (*)
String phoneNumber	Subscriber's phone number used if server needs to dial-out to this subscriber
String pin	pin for the logging in to the web interface (*) pin should be unique among all subscribers on the server if pin is used to identify subscriber in a callflow it should consist only digits
long role	Subscriber's role (i.e. admin, operator, regular user, etc.) Possible values: ROLE_ADMIN (1L), ROLE_OPERATOR (2L), ROLE_USER (3L)
String state	Subscriber's state (****)
long subscriberId	Unique ID assigned by the server
String zip	Subscriber's zip code

* – for this and all subsequent classes designates mandatory fields during object creation or modification

** – for this and all subsequent classes designates fields that were added in version 2.1 and did not exist in version 1.x.

*** – for this and all subsequent classes designates fields that were renamed in version 2.1.

**** – for this and all subsequent classes designates fields that were added in version 2.3 and did not exist in previous version versions

***** – for this and all subsequent classes designates fields that were added in version 2.4 and did not exist in previous version versions

[Click here to see subscriber XML and class definition.](#)

Conference Account – Conference User (Confuser)

Conference user (Confuser) class represents conference account, described in web administration interface guide.

Conference account is the element of subscriber conferences configuration. Conference accounts always belong to subscriber. It is being used to define a person in a conference with a particular role (e.g. host, participant, listener, etc.), the DNIS number that should be used to call to the conference, and the access code that should be entered by the user that called to the conference DNIS to determine his role. A subscriber could be a host user in one conference and a listener in another. Conference accounts with the same conference number represent single conference setup.

Additionally, it is possible to override some attributes exposed by default callflow so this Conference user has a customized behavior (For example this user can disable entry tones just for him while all other users on this number still have them on).

Conference user object can exist only if there is the subscriber that own this confuser and if this conference user assigned to DNIS and if this conference user has conference info (conference number) information that is referred by him. Thus subscriber deletion, DNIS deletion, conference info deletion performs cascade delete of all associated conference users.

Table 2: Properties of Confuser

String accessCode	Access code for this user. It is used for authentication in a conference. Access code should be unique across other accessCodes (*)
ConfInfo conferenceInfo	Holds information about the conference this confuser participates in
long confuserId	Unique ID assigned by the server
Date created	Date when record is created; assigned by the server
long dnisId	ID of DNIS object this user is associated with (*)
long role	Role of this confuser Moderator/Host (1L), Participant (2L), Listener (3L) (*)
long subscriberId	ID of subscriber this confuser belongs to

[Click here to see conference user XML and class definition.](#)

Conference Info (ConfInfo)

This data structure is designed to uniquely identify conference. It is a part of "Conference User" definition and consists of the fields described in Table 3.

All Conference Users with any access codes and the same conferenceNumber will be assigned to the same conference. Please note that Conference Users are not obliged to dial the same DNIS to get to the same conference. To create a new conference you need to pass 0 as a conferenceNumber and provide meaningful description of this conference. In this case server automatically assigns a new unique conferenceNumber.

Table 3: Properties of ConfInfo

Attribute[] attributes	List of attributes and their values imposed by the call flow this conference is assigned to. These attributes may be overwritten for this particular user or taken from parent or defaults (**)
long conferenceNumber	Identifier of the conference where this user will be assigned after successful authentication. It should be unique across other conference numbers; 0 means create a new one (***)
String description	Description of the conference; if conferenceNumber=0 holds new conference description

[Click here to see conference info XML and class definition.](#)

DNIS

DNIS is a unique set of numbers that is outpulsed by a phone carrier that indicates the intended destination for a particular call. This data structure holds information about registered DNIS (called phone numbers) on the bridge. Besides the phone number (usually 10 digits length) each DNIS has a reference to a callflow.

Conference accounts have DNIS (dnisId) as its property, but different DNIS numbers can be used to connect to the same conference. In addition different DNISes can be based on the same callflows but just have different attributes (like a welcome prompt for example).

Table 4: Properties of DNIS

DnisAlias[] aliases	Available aliases for this DNIS (**)
Attribute[] attributes	DNIS attributes inherited and may be overwritten from callflow
long callflowId	ID of callflow this DNIS belongs to
String description	Description
String did	Telephone number, or name if connected to VOIP switch (*)
long dnisId	Unique ID assigned by the server

[Click here to see DNIS XML and class definition.](#)

DNIS Alias (DnisAlias)

The DnisAlias data structure represents a DNIS alias.

Table 5: Properties of DnisAlias

String description	Alias description
String mask	Number pattern (*, 712*, etc.)

Note. This data structure was added in version 2.1 and did not exist in version 1.x.

[Click here to see DNIS alias XML and class definition.](#)

Call Flow (CallFlow)

Call flow is a unique conference service setup, the logic that is used to process the conference calls. This is the process a call goes through from call setup to, to processing, to call tear down. It includes the logic, DTMF key-presses used, functions, and the recorded prompts. Each script takes several parameters (like welcome prompt).

Call flows cannot be dynamically created by user as they need to be put into the proper place on the file system and need to be configured by administrator. However end-user should be able to change attributes of already registered call flows in order to customize their behavior.

Table 6: Properties of CallFlow

Attribute[]	Template of attributes for DNIS and confusers (***)
attributesTemplate	
long callflowId	Unique ID assigned by the server
String name	Callflow description (*), for instance CONF, SPECTEL, etc.
String path	Directory where callflow resides on the server (*)

[Click here to see call flow XML and class definition.](#)

Attribute

This data structure is used to carry attributes for call flow (CallFlow), DNIS and conference user (Confuser). The attributes skeleton is defined by call flow. Other entities can only override some of them. So when a user called in to the conference DNIS it gets attributes exposed by the call flow. Some of these attributes can be already altered by the DNIS. After the user provided his access code and authentication was successful some attributes can be overwritten again by the conference user (Confuser).

It is important to remember that list of attributes is always defined by call flow. Values of some attributes may be overwritten by DNIS and Confuser. Each attribute can be allowed or disallowed for modification by the administrator. The call flow offers default values for each attribute.

Each attribute has name, type and value. Depending of the type web application should apply one or another validation rule. Also attribute has a "role" so confusers can only see those attributes which role matches their own role.

Table 7: Properties of Attribute

String enumValues	if type is eEnum this variable holds possible choices like choice1;choice2;choice3 - this is readonly field populated by server
boolean isOverridden	<ul style="list-style-type: none"> • if the attributes are being getting for Call Flow (attributesTemplate property) this property is always false; • if the attributes are being retrieved for DNIS (as aggregated attributes property) true value means that the attribute is defined on DNIS level and false value means that the attribute is defined on call flow level; • if the attributes are being retrieved for ConfInfo (as aggregated attributes property) true value means that the attribute is defined on ConfInfo level, false - otherwise; • if DNIS object is being saved (using createDNIS or updateDNIS) this property true value means that the attribute should be overridden (saved) on DNIS object level; • if ConfInfo object is being saved (using createConferenceInfo or updateConferenceInfo) this property true value means that the attribute should be overridden (saved) on ConfInfo object level;
String name	attribute name like "ALLOW_CONTINUE" (*)
long role	confuser role this attribute belongs to (*): ROLE_CALLFLOW (3L), ROLE_CONFERENCE (1L), ROLE_DNIS (0L)
long type	attribute type like TYPE_STRING (0L), TYPE_BILLINGRULE (1L), TYPE_INT (2L), TYPE_DTMF (3L), TYPE_ROLE (4L), TYPE_CHOICE (5L) (*)
String value	attribute value like TRUE (*)

[Click here to see attribute XML and class definition.](#)

Conference

This data structure is used to describe ongoing conference on the bridge. Objects of this type are only created by server. User may fetch these objects by calling appropriate function. When conference is over object is deleted by the server.

The conference object is identified by `conferenceId`, this is a globally unique identifier that represents the instance of a conference. So if user has two conferences with the same access code or conference number – these conferences will have different `conferenceId`. It is important to not mix it up with Conference Number. In the previous example these two conferences will have the same Conference Number; the conference number is the property of conference account; if the conference accounts have the same conference number all these accounts determine one single conference.

Table 8: Properties of Conference

<code>long conferenceId</code>	Unique ID assigned by the server
<code>long conferenceNumber</code>	This is a conference number (***)
<code>Date created</code>	Time when this conference was created – the first caller arrived
<code>long duration</code>	Number of seconds which have elapsed since the conference was created
<code>long isOnHold</code>	This field determines whether the conference is on hold; if the conference is online (is not on-hold) the field value is 0L, if the conference is on-hold the field value could be: <code>HOLD_MODE_HOST (1L)</code> , <code>HOLD_MODE_PARTICIPANT (2L)</code> , <code>HOLD_MODE_LISTENER (4L)</code>
<code>boolean isPolling</code>	This field determines whether the polling session is started (**)
<code>boolean isRecording</code>	This field determines whether the conference is being recorded
<code>boolean isSecured</code>	This field determines whether the conference is secured, i.e. new calls allowed to join to the conference or not
<code>long muteMode</code>	This field determines mute mode: <code>MUTE_MODE_OPEN (0L)</code> , <code>MUTE_MODE_QUESTION (1L)</code> , <code>MUTE_MODE_CLOSED (2L)</code> When <code>MUTE_MODE_OPEN</code> mode is enabled any conference participant can talk and mute/unmute himself. When <code>MUTE_MODE_QUESTION</code> mode is enabled all conference participants are muted however any of them can unmute himself to ask a question. When <code>MUTE_MODE_CLOSED</code> mode is enabled all conference participants are muted and can not unmute himself
<code>OperatorStatus operatorStatus</code>	This fields represents the operator's activity, i.e. it contains the data structure that describes the operator's conference
<code>long participantCnt</code>	Number of participants in the conference
<code>long qaMode</code>	This field determines Q&A mode (**): <code>QA_MODE_OPEN (0L)</code> , <code>QA_MODE_CLEAR (1L)</code> , <code>QA_MODE_CLOSED (2L)</code>

[Click here to see conference XML and class definition.](#)

Operator Status (OperatorStatus)

This data structure is designed to show the status of the operator's conference.

Table 9: Properties of OperatorStatus

<code>long engagedConferenceNumber</code>	Conference number of the connected conference
<code>boolean isConnected</code>	This field determines whether the operator's conference is currently connected to the other one (in this case this property is set to true).
<code>boolean isMonitoring</code>	For the operator conference this field determines whether the operator conference is in scanning mode (i.e. surveillance call, usually started when the operator presses *1 on his phone keypad)
<code>long status</code>	This field determines operators conference mode <code>CONFERENCE_REGULAR (0L)</code> , <code>CONFERENCE_OPERATOR (1L)</code> , <code>CONFERENCE_LISTEN (2L)</code> , <code>CONFERENCE_AUTOLISTEN (3L)</code> , <code>CONFERENCE_AUTOLISTEN_SLEEP (4L)</code> , <code>CONFERENCE_TALK (5L)</code>

Note. This data structure was added in version 2.1 and did not exist in version 1.x.

[Click here to see operator status XML and class definition.](#)

ConferenceDR

This data structure is used to describe conference which is already terminated on the bridge. User can not directly create this object.

Table 10: Properties of ConferenceDR

long confereceId	Unique ID assigned by the server
long conferenceNumber	This is a conference number (***)
Date created	Time when this conference was created -first caller arrived
long duration	Number of seconds which have elapsed since the conference was created till the time when it was terminated
Date expirePeriod	Expiration period for shared recording URL
boolean hasPollingResults	Whether or not conference was voted, i.e. whether or not the conference was voted (**)
boolean hasRecording	Whether or not conference was recorded
String jobCode	Active billing (business) code of the conference (*****)
long participantCnt	Number of participants in the conference
long recordingDuration	Recording duration in seconds (***)
String recordingUrl	URL for the audio recording
String sharedRecordingUrl	URL for the shared audio recording
long webRecordingDuration	Conference video duration in seconds (*****)
long webRecordingStatus	Video file conversion status (from 0 till 100) (*****)
String webRecordingUrl	URL for the video recording meta-file (*****)

[Click here to see conferenceDR XML and class definition.](#)

Polling Result (PollingResult)

The PollingResult data structure represents polling results for the specific conference. The conference should be referenced by conferenceId.

Table 11: Properties of PollingResult

Date created	Time when this polling was initiated
Map<object, object> votes	Sequence of option:votesCount pairs (represented by long values)

Note. This data structure was added in version 2.1 and did not exist in version 1.x.

[Click here to see polling result XML and class definition.](#)

Operator's Statistic (OperatorStatistic)

This data structure represents an activity statistic for specific operator in the OPERATOR conference.

Table 12: Properties of OperatorStatistic

String accessCode	Operator's access code
long answersCount	Total number of answers made by operator per specific operator's conference
long answersTime	Total amount of seconds operator spent talking to the customers per specific operator's conference
long avgAnswerTime	Average answer time in seconds for single call per operator's conference, i.e. answersTime/answersCount
long conferenceNumber	Operator's conference number
String name	Operator's name
long timeOnline	Time spent online in seconds, i.e. total amount of time operator was logged in to the interface
long timeWait	Total amount of time operator was waiting for the users

Note. This data structure was added in version 2.1 and did not exist in version 1.x.

[Click here to see operator's statistic XML and class definition.](#)

Session

This data structure represents a single ongoing call on the server. User can not directly create this object. When the call is over server automatically deletes this object.

Normally this data structure is used to get information about call attributes like calling/called number etc. If something needs to be done with the call (mute/hang/hold) the call should be referenced by sessionId.

Table 13: Properties of Session

String accessCode	access code entered by caller
String addressFrom	Full address FROM, i.e. full qualified caller's address
String addressTo	Full address TO, i.e. full qualified callee's address
String bridgeName	Name of hosted bridge (**)
String callee	Information about callee as it is provided in TO field
String caller	Information about caller as it is provided in FROM field (normally the phone number)
String codec	The active audio codec of the session, i.e. the technical name of the codec, that is used by the call, for example PCMU for uLaw, SIREN7 for 722.1, SIREN14 for 722.2, ILBC, etc. (**)
long conferenceId	Conference identifier of the conference this session belongs to (**)
long conferenceNumber	Conference number of the conference this session belongs to
long connectionStatus	Bit-mask that indicates the optional session attributes, for example session direction: 0 - the inbound call, 1 - the outbound call (**)
Date created	Time when this session was created
String customName	custom user name either set from the web or IVR (PIN)
long duration	Number of seconds which have elapsed since the session started
long gainLevel	The microphone volume level of the call, i.e. gain control option; it could be from -10 till 10 or 255; -10 is the quietest (lowest) sound level, 10 is the loudest (highest) sound level, 255 denotes that the microphone level is being automatically adjusted by the backend (**)
boolean isMuted	whether this session is muted or not
boolean isOnHold	whether this session is put on hold by administrator
boolean isOnHoldSelf	whether this session is put on hold by the client (owner)
String jobCode	Active billing (business) code (**)
Date joined	Time when this session joined to the conference
String nodeName	Name of hosted node (**)
String operatorMode	This field represents the operator's activity (for instance, empty, waiting for operator, speaking with operator, etc.). Possible values: <ul style="list-style-type: none"> • null (empty) - the caller does not need operator assistance; • wait - the caller is waiting operator assistance, i.e. the caller is in the operator's queue; • talk - the caller is talking to the operator
long qaStatus	This field represents Q&A mode for current session: QA_STATUS_IDLE (0L), QA_STATUS_RAISEDHAND (1L), QA_STATUS_ACTIVE (2L)

long role	This field determines what role this session has. The roles should be the same as in Confusers. Role helps to verify whether this session is allowed to do recording - MODE_UNDEFINED (0L) MODE_HOST (1L) - host permissions granted, MODE_PARTICIPANT (2L) - caller controls muting, i.e. the session owner can mute/unmute himself, MODE_LISTENER (3L) - the session owner can only listen and can not talk, MODE_RECORDING (4L) - the recording session, MODE_DC_LINK (8L) - distributed conference (DC) link, i.e. the control call between two bridges in distributed conferencing
long sessionId	Unique ID assigned by the session
long status	This field determines whether the current session status: STATUS_IVR (1L) - session is owned by frontend; STATUS_CONFERENCE (2L) - session is owned by backend; STATUS_CLOSED (3L) - session is closed; STATUS_DIALING (4L) - session is dealing
String subconference	If non-empty, denotes current sub-conference of the session (caller) (****)
long subscriberId	ID of subscriber assigned by the session

[Click here to see session XML and class definition.](#)

SessionDR

This data structure represents a single call on the server which is already terminated on the on the bridge. User can not directly create this object.

Table 14: Properties of SessionDR

String accessCode	access code entered by caller
String addressFrom	Full address FROM, i.e. full qualified caller's address
String addressTo	Full address TO, i.e. full qualified callee's address
String bridgeName	Name of hosted bridge (**)
String callee	Information about callee as it is provided in TO field (**)
String caller	Information about caller as it is provided in FROM field (normally the phone number)
long conferenceId	Conference identifier of the conference this session belongs to
long conferenceNumber	Conference number this session belongs to
Date created	Time when this session was created
String customName	Custom user name either set from the web or IVR (PIN)
long disconnectInitiator	Shows who initiated a disconnect (user, bridge): INITIATOR_BRIDGE (2L) - used when session was terminated by bridge; INITIATOR_UNDEFINED (0L) - used when initiator is not defined; INITIATOR_USER (1L) - used when session was terminated by user
String disconnectReason	A string showing detailed info about disconnect
long duration	Number of seconds which have elapsed since the session started and before disconnect
String jobCode	Active billing (business) code (**)
Date joined	Time when this session joined to the conference
String nodeName	Name of hosted node (**)
long role	This field determines what role this sessions had.
long sessionId	Unique ID assigned by the session
long subscriberId	ID of subscriber assigned by the session

Note if the operator was involved into the call – the user called to the operator and the operator attached the user to another conference there would be two SessionDR records with the same session identifier (sessionId). These records will differ by disconnect reason.

[Click here to see sessionDR XML and class definition.](#)

DTMF Event (DtmfEvent)

The DtmfEvent data structure represents a single DTMF command.

Table 15: Properties of DtmfEvent

Date created	Time when this DTMF event was initiated
String dtmf	The DTMF command

Note. This data structure was added in version 2.1 and did not exist in version 1.x.

[Click here to see DTMF event XML and class definition.](#)

Subscriber Conference (SubscriberConference)

This data structure represents the single subscriber's conference. The conference can be either started or not started. The class is being represented by two properties – conference that describes ongoing subscriber conference on the bridge and subscriber that describes the subscriber of the conference, for each conference this subscriber field contains the list of *Confusers* objects for that single conference only. That means that the each instance of the class for the selected conference (conference number) contains the active conference information (if the conference not started only conference number property is populated, conference identifier in this case equal 0) and its subscriber information, the subscriber information could contain up to three conference users information – for the conference host, the conference participant, and the conference listener.

Table 16: Properties of SubscriberConference

Conference conference	The ongoing conference information with its properties
Subscriber subscriber	The subscriber information for the conference including up to three conference users (for host, participant, and listener)

Note. This data structure was added in version 2.2 and did not exist in previous versions.

[Click here to see DTMF event XML and class definition.](#)

Dialout Subscriber (DialoutSubscriber)

This data structure represents information necessary to perform the single dialout. The list (array) of these objects could be used to perform batch (grouped) dialing out, see Chapter 4: Function Reference, Conferences and Calls Management for details.

Table 17: Properties of DialoutSubscriber

String accessCode	The actual access code that should be used to connect to the conference. To connect to sub-conference right after dialout the access code could be formed as <actual access code>_<sub-conference name>
String customName	Custom user name that should be set to the caller
String dnis	The conference DNIS phone number the callers has to be connected to
String phoneNumber	The caller phone number to dial-out

Note. This data structure was added in version 2.3 and did not exist in previous versions.

[Click here to see DTMF event XML and class definition.](#)

Chapter 3: Samples of Functions

WYDE Web Services Initialization

Sample of WYDE Web Services Initialization

To use WYDE Web Services, i.e. to call its methods, they should be pre-initialized and pre-authenticated in your code – you should set web services URL (<http://<WYDE bridge domain>/dnca/jAdmin>), user name (subscriber PIN) and password that should be used in the authentication.

Click here to see sample of the web services initialization source code and configuration file:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Web Methods' XML Requests and Responses

Each web services function (web method) when it is in use sends the XML request to the server and receives the XML response from the server. XML request contains the name of the function that is being used and all parameters of the function; these parameters can be either scalar values or objects represented in XML form. XML response contains the name of the function that is generating this request and the returned value; the returned value can be either void, or scalar value, or object, or list of objects.

All samples given in this guide contains both XMLs: requests sent to server and responses received from server. To view XML samples you may need Internet access and web browser. This section of the guide describes different XML requests and responses that are being generated during web methods calls.

Sample of XML for Function with Multiple Parameters Sent and List of Objects Received

Let's review *getSessionDRs* function.

This function expects four parameters: offset, limit, filter, order – see Chapter 4: Function Reference, Section: CDRs Management for details. For instance we would like to run this function with parameters offset = 0, limit = 3, filter = `created>='2009-10-01' and conferenceNumber=667788`, and order – empty. To execute this call the XML shown in Sample of XML Request for Function with Multiple Parameters Sent will be generated.

This function returns the list of SessionDR objects. In our sample it returns 3 objects, the XML response of this function is shown in Sample of XML Response for Function with List of Objects Received.

Sample of XML for Function with the Object Parameter Sent and the Object Received

Let's review *createSubscriber* function.

This function expects single parameter – the object representing the Subscriber class. Mandatory subscribers attributes (properties) should be populated in XML. Creating the subscriber you can also create his conference users simultaneously (in the same function call) with the subscriber creation (because confusers in the property of the subscriber); to do so you should populate confusers property of the subscriber class. XML generate for subscriber and his conference users creation is shown in Sample of XML Request for Function with the Object Parameter Sent.

This function returns the created Subscriber object. Note that this returned object will not be the same with the object that was sent to the server: the subscriber identifier, default attributes values (such as role, etc.), and additional conference users attributes will be populated in the returned object. The XML response of this function is shown in Sample of XML Response for Function with the Object Received.

Subscribers Management

Sample of Subscriber and his Conference Accounts Creation

Let's review the following scenario:

- we need to create the subscriber;
- when we create the subscriber we need to create three conference accounts (conference users) – the first for moderator, the second for participant, and the third for listener.

To implement this scenario it is necessary to use web method *createSubscriber*. This method allows not only creation of subscribers, but this method also can be used to create conference accounts (conference users) with their attributes that belong to the subscribers.

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Subscribers Filtering, Modifications, Conference Accounts Modifications

Let's review the following scenario:

- we need to find the subscriber that was created in the previous sample using his pin;
- for the selected subscriber we need to modify his password and email;
- for the selected subscriber we need to remove his conference accounts (conference users) with the listener role;
- for the selected subscriber we need to define some custom attributes as well as change access code for his conference accounts with host role.

To implement this scenario it is necessary to use web methods *getSubscribers* and *updateSubscriber*. The *getSubscribers* method is used to filter the subscribers based on different criteria. The *updateSubscriber* method allows not only modification of subscriber's properties, but this method also can be used to create, modify or delete conference accounts (conference users) and conference info with their attributes that belong to this subscriber. As alternative approach of updating of conference info and their attributes information it is possible to use *updateConferenceInfo* method as shown in this sample.

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Subscribers Filtering and Deletion

Let's review the following scenario:

- we need to find out all subscribers who have emails from domain "manage.com";
- for each of these subscribers if the subscriber does not have phone number we need to delete him.

To implement this scenario it is necessary to use web methods *getSubscribers* (to filter the subscribers) and *deleteSubscriber* (to delete the selected subscriber).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Getting Conference Users Information

Let's review the following scenario:

- we need to count conference users (accounts) with for SPECTEL call flow;
- we need to get all conference users (accounts) with for SPECTEL call flow with host role;
- we need to output subscriber ID, conference number, access code for them.

To implement this scenario it is necessary to use web methods *getCallFlows* (to filter the call flows), *getDNISes* (to filter the DNISes), *getConfusersCount* (to get the number of conference users based on criteria) and *getConfusers* (to filter the conference users based on criteria).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Conferences and Calls Management

Sample of Conferences Filtering, Changes Secure Mode, Dropping the Conferences

Let's review the following scenario:

- we need to count how many conferences are currently on the bridge;
- for the selected subscriber we need to drop all conferences if the participants count less than two;
- for unsecured conferences for the selected subscriber with two participants we need to make them secure.

To implement this scenario it is necessary to use web methods *getConferencesCount* (to get the number of active conferences based on criteria), *getConferences* (to filter the conferences based on different criteria), *hangupConference* (to hang-up the selected conference, i.e. to drop all conference calls and terminate the conference), *secureConference* (to make the conference secure, i.e. to move the conference into the state when no new calls are allowed to get in there).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Placing the Entire Conference on Hold, Starting and Stopping Q&A Sessions and Conference Recording

Let's review the following scenario:

- we need to place the specific conference (the conference with specific conference number) on hold;
- we need to wait 1 minute and take this conference off hold;
- after that we need to start conference recording and start Q&A session for this conference;
- we need to wait 1 minute, we assume that conference participants requested to ask questions during this minute;
- we need to let the first participant ask his question (i.e. un-mute him - engage his Q&A session);
- we need to wait 1 minute and then complete the first participant question, i.e. disengage his Q&A session;
- we need to stop Q&A session and stop conference recording for this conference.

To implement this scenario it is necessary to use web methods *getConferences* (to filter the conferences based on different criteria), *getSessions* (to filter the conference calls based on different criteria), *holdConference* (to place the conference on hold), *unHoldConference* (to take the conference off hold), *qaSetMode* (to start and stop Q&A session for the conference; note the this method should used for these purposes starting from version 2.1 only, in version 1.x method *muteConference* was used), *qaEngage* (to engage Q&A session for the conference participant, i.e. to un-mute the participant), *qaDisengage* (to disengage Q&A session for the conference participant, i.e. to mute the participant after he asked his question), *startConferenceRecording* (to start the conference recording), *stopConferenceRecording* (to stop the conference recording).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Conference Polling Sessions

Let's review the following scenario:

- we need to start the polling session for the specific conference (the conference with specific conference number) with available polling options 1, 2, 3;
- we need to wait 1 minute, we assume that conference participants will vote (select one of the available options) during this minute;
- we need to stop the polling session for this conference;
- after that we need to output polling results.

To implement this scenario it is necessary to use web methods *startPolling* (to start the polling for the specified conference with selected options), *stopPolling* (to stop the polling for the specific conference), *getPollingResults* (to get the list of polling results for the conference).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Calls Filtering, Mute the Calls, Dropping the Calls

Let's review the following scenario:

- we need to count how many calls are currently on the bridge;
- for the selected subscriber we need to drop all participants calls if the call duration greater than 10 minutes;
- for remaining participants of the selected subscriber (with call duration less than 10 minutes) we need to mute their calls.

To implement this scenario it is necessary to use web methods *getSessionsCount* (to get the number of active calls based on criteria), *getSessions* (to filter the calls based on different criteria), *hangupSession* (to drop/disconnect the specific call), *muteSession* (to mute the specific call participant).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Setting Custom Name and Placing Calls on Hold

Let's review the following scenario:

- for the conference with specific conference number we need to set custom name for the host "conference moderator";
- for the same conference we need to place all listeners and participants on hold.

To implement this scenario it is necessary to use web methods *getSessions* (to filter the calls based on different criteria), *setCustomName* (to set the custom name for the specific call participant), *holdSession* (to place the call/participant on hold).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

CDRs Management

Sample of Getting Conferences Historical Information

Let's review the following scenario:

- we need to count how many conferences were on the bridge from the beginning of the month;
- for the selected subscriber we need to output his current month conferences information (conference number, conference ID, date and time when the conference occurred, duration, participants count, and info about recording URL if exists), ordered by conference number and conference date.

To implement this scenario it is necessary to use web methods *getConferenceDRsCount* (to return number of ConferenceDRs, i.e. historical conference information, stored in local CDR database based on criteria), *getConferenceDRs* (to filter the historical conference information based on different criteria).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of the Shared Recording Generation

In the previous sample (Sample of Getting Conferences Historical Information) we get conferences with recording. Let's review the following scenario:

- we need to generate recording URL link, that will allow user to download conference recording without authorization during the next hour (for the conference with recording referenced by the conferenceId, that was found in the previous sample);
- we need to output the ConferenceDR object information prior and after shared recording URL generation to see the differences in the object properties.

To implement this scenario it is necessary to use web methods *shareRecording* (to generate shared recording, i.e. recording URL that will be available without authorization) and *getConferenceDR* (to get the single historical conference information based on the conference identifier).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Getting Calls Historical Information

Let's review the following scenario:

- we need to count how many calls were on the bridge from the beginning of the month for the specific conference number;
- for the specific conference number we need to output current month conference calls information (conference number, conference ID, date and time when the call occurred, duration, called number, calling number, custom name, disconnect reason);
- if number of calls to output greater than 5, we should implement paging and output 5 calls on the page.

To implement this scenario it is necessary to use web methods *getSessionDRsCount* (to return number of SessionDRs, i.e. historical calls/sessions information, stored in local CDR database based on criteria), *getSessionDRs* (to filter the historical calls information based on different criteria).

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Sample of Historical Calls Filtering

Let's review the following scenario:

- for the current month we need to output all calls that were connected to the conferences excluding service calls to the recording server initiated by bridge (for instance we should output calling number, called number, conference number, conference identifier, date/time when the call was started, and how long the call was connected to the conference).

To implement this scenario it is necessary to use web method *getSessionDRs* and use the filter that allows to select the requested calls only.

Click here to see sample of the source code, XML requests and responses, screenshots:

- [Sample in this document](#);
- [Sample on the web](#) (requires Internet access and web browser).

Active Speaker Notification

WYDE bridge software has the mechanism allowing finding out who is speaking at the moment and how loud the person is speaking (i.e. the channel volume). Because this information should be available very fast ("on-the-fly"), it would be too costly to call web services each time for these requests. WYDE bridge software uses the lightweight [JSON](#) (JavaScript Object Notation) calls for this purpose.

From the web active talker indicators can be received for one specific conference only. I.e. WYDE software gives this information not for all active conferences, but for requested conferences only. To do that it is necessary to implement http request for the URL:

/jsonASN.jsp?conferenceNumber=667788 (where 667788 is the conference number)

As the response you will get JSON-array, for instance:

```
[
  {"sessionId":"16778157","level":"5"},
  {"sessionId":"16778156","level":"2"}
]
```

Actually the system shows the loudest four persons and their sound volume. If there was no any information returned, that means that everybody keeps silent. The sound level could be from 0 (silence) till 15 (loudest). Note the silence (0) level is not being responded. The minimum level that could be returned is 1.

Below we show the JavaScript code sample how this mechanism can be implemented. The sample shows how to get the active speaker notifications.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/yuiloader/yuiloader-min.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/event/event-min.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/dom/dom-min.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/logger/logger-min.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/json/json-debug.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/connection/connection-min.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/element/element-min.js"></script>
    <script type="text/javascript"
src="http://yui.yahooapis.com/2.8.1/build/button/button-min.js"></script>

    <title>ASN Example</title>
  </head>
  <body>
    <div id="demo_msg"></div>
    <br/><br/>

    <script type="text/javascript">

      // Get the div element in which to report messages from the server
      var msg_section = YAHOO.util.Dom.get('demo_msg');
      msg_section.innerHTML = '';

      var callbacks = {
        // Successful XHR response handler
        success : function (o) {
          // Get the div element in which to report messages from the server
          msg_section = YAHOO.util.Dom.get('demo_msg');
          msg_section.innerHTML = '';

          var messages = [];
          // Use the JSON Utility to parse the data returned from the server
          try {
            messages = YAHOO.lang.JSON.parse(o.responseText);
          }
        }
      }
    </script>
  </body>
</html>
```



```

        catch (x) {
            alert("JSON Parse failed!");
            return;
        }
        // The returned data was parsed into an array of objects.
        // Add a P element for each received message
        for (var i = 0, len = messages.length; i < len; ++i) {
            var m = messages[i];
            var p = document.createElement('p');
            var message_text =
                document.createTextNode("sessionId="+m.sessionId+",
                                         level="+m.level );
            p.appendChild(message_text);
            msg_section.appendChild(p);
        }
    }
};

function getInfo(conf_number){
    if( conf_number>0 ) {

YAHOO.util.Connect.asyncRequest('GET',"http://87.246.167.126/jsonASN.jsp?conferen
ceNumber="+conf_number, callbacks);
    }
}

</script>

<label>Enter conference number</label>
<input type="text" value="" id="conf_number_id"/>
<input type="button" value="Get info!"
onclick="getInfo(document.getElementById('conf_number_id').value)"/>

</body>

</html>

```

Storage Library

Storage is a file manager for WYDE bridge conferences files. It is written as a java web application. It allows you to display contents of the folders, download the files, upload the files, as well as perform different files management tasks, including deletion, renaming, sharing, etc.

Storage application URL is being formed as <http://<Wyde bridge domain>/storage/>. For instance the possible Storage file manager URLs could be <http://192.168.1.5/storage/>.

Storage can be secured with the standard web application security. For this, remove the comment around the `<security-constraint>` entry in *storage/WEB-INF/web.xml* file and only users in role moderator will have access. Your *web.xml* file should contain the following code:

```

<security-constraint>
    <web-resource-collection>
        <web-resource-name>storage</web-resource-name>
        <url-pattern>/list/*</url-pattern>
        <url-pattern>/ctx/*</url-pattern>
        <url-pattern>/dlf/*</url-pattern>
        <url-pattern>/dlx/*</url-pattern>
    
```

```

</web-resource-collection>

<auth-constraint>
  <role-name>manager</role-name>
</auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>MyRealm</realm-name>
</login-config>

<security-role>
  <role-name>manager</role-name>
</security-role>

```

Note that *MyRealm* object should have the specified *manager* role.

Folders Structure and Showing Folder Content for Conference and Bridge Files

To display the folder content the *list* option should be used right after the *storage* portion in the URL.

Storage always displays the contents of one single folder only.

There are two ways to specify the conference folder:

- Specify conference's folder by the DNIS number and access code of the conference:
 - http://<Wyde bridge domain>/storage/list/<dnis number>_<access code>/ for example:
 - http://192.168.1.5/storage/list/8665080012_419881/
- Specify conference's folder by the conference number:
 - <http://<Wyde bridge domain>/storage/list/<conference number>/> for example:
 - <http://192.168.1.5/storage/list/749932/>

Both way are fully equivalent; in both cases the result will be exactly the same. The sample of the folder content returned using the approach described above is shown in Figure 3. As you can see file name, file size, file last modification date, relative URL to download the file, shared/private flag, and sorting order are returned. If the description has been set for the file the rightmost column contains this file description.

File/Folder Name	File Size	Last Modification Date	URL to Download the File	shared/private	order
record dir	Dec 8, 2010 6:11:15 PM	/record private	-1		
public dir	May 18, 2010 1:10:49 AM	/public private	-1		
sample1.jpg	4789 bytes	Apr 22, 2010 12:07:36 PM	/storage/dlf/usr/local/DNCA/var/recordings/932/749932/upload/sample1.jpg	shared	-1
sample2.jpg	3290 bytes	Apr 22, 2010 12:07:36 PM	/storage/dlf/usr/local/DNCA/var/recordings/932/749932/upload/sample2.jpg	shared	-1
sample3.jpg	4220 bytes	Apr 22, 2010 12:07:36 PM	/storage/dlf/usr/local/DNCA/var/recordings/932/749932/upload/sample3.jpg	shared	-1
sample4.jpg	4318 bytes	Apr 22, 2010 12:07:36 PM	/storage/dlf/usr/local/DNCA/var/recordings/932/749932/upload/sample4.jpg	shared	-1

Figure 3: Folder Content Sample

There are three standard folders related to each conference:

- Root folder – contains all files uploaded for this conference:
 - http://.../<dnis number>_<access code>/
 - <http://.../<conference number>/>
 for example:
 - http://192.168.1.5/storage/list/8665080012_419881/
 - <http://192.168.1.5/storage/list/749932/>

- Public folder – contains public files (uploaded or recorded) for this conference:

- http://.../<dnis number>_<access code>/public/
- <http://.../<conference number>/public/>

for example:

- http://192.168.1.5/storage/list/8665080012_419881/public/
- <http://192.168.1.5/storage/list/749932/public/>

- Recording folder – contains audio recordings available for this conference:

- http://.../<dnis number>_<access code>/recordings/
- <http://.../<conference number>/recordings/>

for example:

- http://192.168.1.5/storage/list/8665080012_419881/recordings/
- <http://192.168.1.5/storage/list/749932/recordings/>

The output format for all these folders is the same as it was previously described and shown on Figure 3.

Note that participants can see files from the *public* folder only; hosts have access to all these folders.

Additionally there are some special folders paths (URLs) related to the entire bridge:

- Welcome prompts folder – contains conference (DNIS) welcome prompts sound files, usually it is `/usr/local/DNCA/var/sounds/welcome-prompts/` folder:
 - <http://<Wyde bridge domain>/storage/list/welcome-prompts/>
 - for example:
 - <http://192.168.1.5/storage/list/welcome-prompts/>
- Music-on-hold folder – contains conference (DNIS) music on hold sound files, usually it is `/usr/local/DNCA/var/sounds/moh/` folder:
 - <http://<Wyde bridge domain>/storage/list/moh/>
 - for example:
 - <http://192.168.1.5/storage/list/moh/>
- Operator prerecorded messages folder – contains prerecorded operator messages sound files that could be played either to the users from his operator's queue or to the all users on the bridge or to the specific user's conference, usually it is `/usr/local/DNCA/var/sounds/operator-messages/` folder:
 - <http://<Wyde bridge domain>/storage/list/operator-messages/>
 - for example:
 - <http://192.168.1.5/storage/list/operator-messages/>
- Operator temporary custom messages folder – contains temporary recorded operator messages sound files, i.e. the messages recorded by the operator on his phone, that could be played either to the users from his operator's queue or to the all users on the bridge or to the specific user's conference, usually it is `/usr/local/DNCA/var/recordings/operator-messages/` folder:
 - <http://<Wyde bridge domain>/storage/list/custom-operator-messages/>
 - for example:
 - <http://192.168.1.5/storage/list/custom-operator-messages/>

File upload

To upload one or several files use the same folder URL: http://<Wyde bridge domain>/storage/list/<dnis number>_<access code>/ or <http://<Wyde bridge domain>/storage/list/<conference number>/>.

The example below illustrates how to upload file to the "upload" folder:

```
<form action="http://192.168.1.5/storage/list/8665080012_419881/"
      method="post" enctype="multipart/form-data">
  <input type="file" name="file" size="30">
  <input type="submit" name="command" value="Upload"/>
</form>
```

Files management

There are different commands that can be used within the Storage file manager. The name of the command and one or two possible parameters are being specified in URL query string fields using traditional web approach:

- [http://.../<dnis number>_<access code>/\[public/recordings/?index=<file_name>&command=<command: delete|share|setOrder|setDescription|prepareAudioFile> \[&other_parameter_name=other_parameter_value\]\]](http://.../<dnis number>_<access code>/[public/recordings/?index=<file_name>&command=<command: delete|share|setOrder|setDescription|prepareAudioFile> [&other_parameter_name=other_parameter_value]])
- [http://.../<conference number>/\[public/recordings/?index=<file_name>&command=<command: delete|share|setOrder|setDescription|prepareAudioFile> \[&other_parameter_name=other_parameter_value\]\]](http://.../<conference number>/[public/recordings/?index=<file_name>&command=<command: delete|share|setOrder|setDescription|prepareAudioFile> [&other_parameter_name=other_parameter_value]])

In particular, encoding the query string uses the following rules:

- Letters (A-Z and a-z), numbers (0-9) and the characters '!', '-', '~' and '_' are left as-is;
- All other characters are encoded as %FF hex representation (for example SPACE is encoded as %20, etc.).

There following commands are available in Storage:

- *delete* – deletes the specified file:
 - http://.../?index=<file_name_to_delete>&command=delete
 - for example:
 - http://192.168.1.5/storage/list/8665080012_419881/recordings/?index=333456.mp3&command=delete
 - <http://192.168.1.5/storage/list/749932/recordings/?index=333456.mp3&command=delete>
- *share* – shares the specified file:
 - http://.../?index=<file_name_to_share>&command=share
 - for example:
 - http://192.168.1.5/storage/list/8665080012_419881/recordings/?index=333456.mp3&command=share
 - <http://192.168.1.5/storage/list/749932/recordings/?index=333456.mp3&command=share>

This command works similar to web method **shareRecording**, but you can share any file in any available folders. When you share the file the symbolic link to the specified file is being created in the *public* sub-folder and this file becomes available for the user with participant roles. There is no expiration period for such shared file, but

you can remove the sharing simply by deleting of the created symbolic link in the *public* sub-folder.

Once the file is shared you can see that the *shared/private* column value for this file will be set to *shared*.

- *setOrder* – for the specified file sets list order according to the *order* parameter:
 - http://.../?index=<file_name_to_set_order>&command=setOrder&order=<new_order_value>

for example:

- http://192.168.1.5/storage/list/8665080012_419881/recordings/?index=333456.mp3&command=setOrder&order=3
- <http://192.168.1.5/storage/list/749932/recordings/?index=333456.mp3&command=setOrder&order=3>

When you display the folder content you can see the sorting order for each file; default (i.e. non-defined) sort order is -1. This order is being used to define the order in what the files are displayed using *list* URL option. In addition your external application can use this order to sort the files.

- *setDescription* – sets the description of the specified file according to the *description* parameter:
 - http://.../?index=<file_name_to_set_description>&command=setDescription&description=<new_file_description>

for example:

- http://192.168.1.5/storage/list/8665080012_419881/recordings/?index=333456.mp3&command=setDescription&description=new%20testing%20description
- <http://192.168.1.5/storage/list/749932/recordings/?index=333456.mp3&command=setDescription&description=new%20testing%20description>

Once you changed the description of the file you can see this description in the rightmost column when you are displaying the folder content.

- *prepareAudioFile* – checks availability of the specified file and converts it if needed according to preferred codec:
 - http://.../?index=<file_name_to_prepare_audio_file>&command=prepareAudioFile

for example:

- http://192.168.1.5/storage/list/8665080012_419881/recordings/?index=333456.mp3&command=prepareAudioFile
- <http://192.168.1.5/storage/list/749932/recordings/?index=333456.mp3&command=prepareAudioFile>

This command is used to convert the specified file into preferred codec of your bridge (for instance *.ul* or *.g722*); if needed the file with this extension will be created and as the result it will be possible to playback this file in the conferences.

Chapter 4: Function Reference

Subscribers Management

- **getSubscriber** (long subscriberId) – Returns full information about the Subscriber with the given ID.

Parameters:

subscriberId – The Subscriber identifier

Returns:

Subscriber object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getSubscribers** (long offset, long limit, String filter, String order) – This function returns list of Subscribers that match filter. Offset and limit allow implementing paging on the web server. Please note that field *confusers* in Subscriber will not be populated to avoid huge amount of data to be transferred in case if big request is processing Subscriber objects.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Subscriber field names or composite (compound) statement relative to confuser/conferenceInfo fields names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example pin='12' or pin like '%2%' or subscriberId >= 15.

Empty string or null means no filter.

order - A string specifying Subscriber field name and sort direction.

For example "pin" or "email desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Acceptable fields for filtering and sorting:

- subscriberId
- parentId
- pin
- password
- firstName
- lastName
- email
- address1
- city
- state
- zip
- country

- phoneNumber
- confuser.accessCode
- confuser.confuserId
- confuser.dnisId
- confuser.role
- confuser.subscriberId
- confuser.conferenceInfo.conferenceNumber

Returns:

list of Subscriber objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getSubscribersCount** (String filter) – Returns count of Subscribers that match the given filter.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Subscriber field names or composite (compound) statement relative to confuser/conferenceInfo fields names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example pin='12' or pin like '%2%' or subscriberId >= 15.

Acceptable fields for filtering:

- subscriberId
- parentId
- pin
- password
- firstName
- lastName
- email
- address1
- city
- state
- zip
- country
- phoneNumber
- confuser.accessCode
- confuser.confuserId
- confuser.dnisId
- confuser.role
- confuser.subscriberId
- confuser.conferenceInfo.conferenceNumber

Empty string or null means no filter.

Returns:

long count of Subscribers

Throws Exceptions:

ServerException

AccessDeniedException

- **createSubscriber** (Subscriber subscriber) – Creates a Subscriber. Pay attention to the list of mandatory fields to be filled in.

Parameters:

subscriber – The Subscriber object

Returns:

created Subscriber object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **updateSubscriber** (Subscriber subscriber) – Updates a Subscriber whose ID is presented in s with the information from the structure. Please make sure you filled all information that needs to be in the updated Subscriber. Recommendation is to call **getSubscriber** first, change some info and then call **updateSubscriber**.

Parameters:

subscriber – The Subscriber object

Returns:

updated Subscriber object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **deleteSubscriber** (long subscriberId) – Deletes a Subscriber with the given ID and all subordinate Confusers.

Parameters:

subscriberId – The Subscriber identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **generateSubscriberPin** () – This function returns unique Subscriber pin with respect to codes registered on the local server. This function is helpful for createSubscriber.

Returns:

string Pin Code which is a 6 digit number. For example: 215246.

Throws Exceptions:

ServerException

AccessDeniedException

- **`generateAccessCode`** () – This function returns unique access code with respect to codes registered on the local server. This function is helpful for `createSubscriber` and `createConfuser`.

Returns:

string Access Code which is a 6 digit number. For example: 346217.

Throws Exceptions:

ServerException

AccessDeniedException

- **`generateAccessCodeEx`** (long digits) – This function returns unique access code with the length specified by the argument with respect to access codes registered on the local server. This function is helpful for `createSubscriber` and `createConfuser`.

Parameters:

digits – The length of the generated access code, should be from 1 till 13

Returns:

string Access Code which consists of digits, the length of the access code is specified by the parameter *digits* of this function. For example: 481237854 (if digits=9).

Throws Exceptions:

ServerException

AccessDeniedException

Subscribers' Conference Users Management

- **`getConfuser`** (long confuserId) – This function returns full details about the Confuser referenced by ID.

Parameters:

confuserId – The Confuser identifier

Returns:

Confuser object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **`getConfusers`** (long offset, long limit, String filter, String order) – This function returns the list of Confuser which match the given filter. There are rare cases when this function needs to be called directly as `getSubscriber` returns list of subordinate conference users.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Confuser field names or composite (compound) statement relative to conferenceInfo field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example pin='12' or pin like'%2%' or subscriberId >= 15.

Empty string or null means no filter.

order - A string specifying Subscriber field name and sort direction.

For example "pin" or "email desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Acceptable fields for filtering and sorting:

- accessCode
- confuserId
- dnisId
- role
- subscriberId
- conferenceInfo.conferenceNumber

Returns:

list of Confuser objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getConfusersCount** (String filter) – This function returns number of Confusers that match the given filter.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Confuser field names or composite (compound) statement relative to conferenceInfo field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example accessCode='1001' or subscriberId >= 15.

Acceptable fields for filtering:

- accessCode
- confuserId
- dnisId
- role
- subscriberId
- conferenceInfo.conferenceNumber

Empty string or null means no filter.

Returns:

long count of Confusers

Throws Exceptions:

ServerException

AccessDeniedException

- **`createConfuser`** (`Confuser confuser`) – This function creates a new Confuser. Please note that you can create Confusers by calling **`createSubscriber`** and providing list of Confusers there.

Parameters:

`confuser` – The Confuser object

Required fields:

- `subscriberId`
- `role`
- `dnisId`
- `accessCode`
- `conferenceInfo`

Returns:

created Confuser object

Throws Exceptions:

`ServerException`

`AccessDeniedException`

`ObjectValidationException`

- **`updateConfuser`** (`Confuser confuser`) – This function updates Confuser which is presented in confuser with the information from the structure. Please make sure you filled all information that needs to be in the updated Confuser.

Recommendation is to call **`getConfuser`** first, change some info and then call

`updateConfuser`.

Parameters:

`confuser` – The Confuser object

Returns:

updated Confuser object

Throws Exceptions:

`ServerException`

`AccessDeniedException`

`ObjectValidationException`

- **`deleteConfuser`** (`long confuserId`) – This function deletes Confuser referenced by the ID.

Parameters:

`confuserId` – The Confuser identifier

Returns:

`void`

Throws Exceptions:

`ServerException`

`AccessDeniedException`

`ObjectNotFoundException`

Conference Info Management

- ***getConferenceInfos*** (long offset, long limit, String filter, String order) – This function returns list of ConfInfo objects which are registered for the subscriber on which behalf this call is executed. For administrator it returns list of all registered ConfInfo objects.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more ConfInfo field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example conferenceNumber='12' or conferenceNumber like'%2%'.

Accepted fields:

- conferenceNumber
- description

Empty string or null means no filter.

order - A string specifying ConfInfo field name and sort direction.

For example "conferenceNumber" or "description desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of ConfInfo objects

Throws Exceptions:

ServerException

AccessDeniedException

Note:

This function was created in version 2.1 and did not exist in previous versions.

- ***getConferenceInfosCount*** (String filter) – Returns number of ConfInfo objects that match the given filter.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more ConfInfo fields names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example conferenceNumber='12' or conferenceNumber like'%2%'.

Accepted fields:

- conferenceNumber
- description

Empty string or null means no filter.

Returns:

long count of ConfInfo objects

Throws Exceptions:

ServerException

AccessDeniedException

Note:

This function was created in version 2.1 and did not exist in previous versions.

- **`createConferenceInfo`** (`ConfInfo confInfo`) – This function creates a new `ConfInfo` object. Pay attention to the list of mandatory fields to be filled in.

Parameters:

`confInfo` – The `ConfInfo` object

Required fields:

- `conferenceNumber` (0 means create a new one – in this case description property should contain new conference description and new conference number is being generated)

Note: if attributes property is populated only attributes with `isOverridden=true` will be saved.

Returns:

created `ConfInfo` object

Throws Exceptions:

`ServerException`

`AccessDeniedException`

`ObjectValidationException`

Note:

This function was created in version 2.1 and did not exist in previous versions.

- **`updateConferenceInfo`** (`ConfInfo confInfo`) – This function updates an existing `ConfInfo` object.

Parameters:

`confInfo` – The `ConfInfo` object

Returns:

updated `ConfInfo` object

Throws Exceptions:

`ServerException`

`AccessDeniedException`

`ObjectValidationException`

- **`deleteConferenceInfo`** (`long conferenceNumber`) – This function deletes `ConfInfo` object referenced by the conference number and all assigned confusers (i.e. `Confuser` objects that refer to this conference number).

Parameters:

`conferenceNumber` – The conference number

Returns:

`void`

Throws Exceptions:

`ServerException`

`AccessDeniedException`

`ObjectNotFoundException`

Note:

This function was created in version 2.1 and did not exist in previous versions.

Conferences and Calls Management

- ***getConference*** (long conferenceId) – This function returns full details about the Conference referenced by the ID.

Parameters:

conferenceId – The Conference identifier

Returns:

Conference object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- ***getConferences*** (long offset, long limit, String filter, String order) – This function returns list of Conferences which are registered for the subscriber on which behalf this call is executed.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Conference field names.

Acceptable operators: <=, >=, !=, =, <, >, like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example conferenceNumber='12' or conferenceNumber like'%2%' or duration >= 15 or created like '%11/04/2011%'.

Accepted fields:

- conferenceId
- conferenceNumber
- created ('yyyy.MM.dd/hh:mm' format)
- duration
- participantCnt
- isSecured
- muteMode

Empty string or null means no filter.

order - A string specifying Conference field name and sort direction.

For example "conferenceNumber" or "created desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of Conference objects

Throws Exceptions:

ServerException

AccessDeniedException

- ***getConferencesCount*** (String filter) – This function returns number of Conferences currently running on the server.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Conference field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example conferenceNumber='12' or conferenceNumber like'%2%' or duration >= 15 or created like '%11/04/2011%'.

Accepted fields:

- conferenceId
- conferenceNumber
- created ('yyyy.MM.dd/hh:mm' format)
- duration
- participantCnt
- isSecured
- muteMode

Empty string or null means no filter.

Returns:

long count of Conference objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getSession** (long sessionId) – This function returns full details about the call referenced by the ID provided.

Parameters:

sessionId – The Session identifier

Returns:

Session object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getSessions** (long offset, long limit, String filter, String order) – This function returns list of Sessions (calls) which match the filter provided. There are two parameters offset and limit which help to implement paging on the web application. If this function is called from non admin Subscribers it will returns only Sessions visible for this account. If call doesn't present an access code yet – it is visible only by admin.

Parameters:

conferenceId - Conference identifier. If parameter is less than zero Session objects for all Conference will be returned.

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Session field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example addressTo='12' or addressTo like'%2%' or duration >= 15 or created like '%11/04/2011%' or conferenceId=854123.

Accepted fields:

- sessionId
- subscriberId
- created ('yyyy.MM.dd/hh:mm' format)
- joined ('yyyy.MM.dd/hh:mm' format) (works only when joined the conference)
- duration
- status
- qaStatus (for sort order only, i.e. not for filters)
- role (works only when joined the conference)
- isMuted (works only when joined the conference) true/false values
- addressTo
- addressFrom
- conferenceNumber (works only when joined the conference)
- accessCode (works only when joined the conference)
- subconference (works only when joined the conference)

Empty string or null means no filter.

order - A string specifying Session field name and sort direction.

For example "caller" or "caller desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of Session objects

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getSessionsCount** (String filter) – This function returns number of calls on the bridge which matches the filter provided.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more Session field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example caller='12' or caller like'%2%' or duration >= 15 or created like '%11/04/2011%' or conferenceId=854123.

Accepted fields:

- sessionId
- subscriberId
- created ('yyyy.MM.dd/hh:mm' format)
- joined ('yyyy.MM.dd/hh:mm' format) (works only when joined the conference)
- duration
- status
- role (works only when joined the conference)
- isMuted (works only when joined the conference) true/false values
- addressTo
- addressFrom
- conferenceNumber (works only when joined the conference)

- `accessCode` (works only when joined the conference)
 - `subconference` (works only when joined the conference)
- Empty string or null means no filter.

Returns:

long count of Session objects

Throws Exceptions:

ServerException

AccessDeniedException

- **`hangupConference`** (long `conferenceId`) – This function causes all calls to be dropped from the Conference and Conference to be terminated.

Parameters:

`conferenceId` – The Conference identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **`hangupSession`** (long `sessionId`) – This function disconnects the call reference by the ID. If called not from admin account may return NonAuthorised exception.

Parameters:

`sessionId` – The Session identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **`secureConference`** (long `conferenceId`) – This function moves a Conference referenced by ID into the state when no new calls are allowed to get in there.

Parameters:

`conferenceId` – The Conference identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **`unSecureConference`** (long `conferenceId`) – This function cancels effect of `secureConference`, i.e. new calls can join the Conference.

Parameters:

`conferenceId` – The Conference identifier

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- **holdConference** (long conferenceId) – This function places the conference on hold.

Parameters:

conferenceId – The Conference identifier

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- **unHoldConference** (long conferenceId) – This function places the conference off hold.

Parameters:

conferenceId – The Conference identifier

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- **holdSession** (long sessionId) – This function places the call on hold.

Parameters:

sessionId – The Session identifier

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- **unHoldSession** (long sessionId) – This function places the call off hold.

Parameters:

sessionId – The Session identifier

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- **muteConference** (long conferenceId, long mode) – This function mutes all participants (it doesn't touch moderators). There are 3 mute modes Open (0) – this is when all can speak or mute themselves Relaxed (1) – this is when all participants muted, but they can un-mute themselves Strict (2) – this is when participants cannot un-

mute themselves. If Q&A is enabled they can put themselves into the question queue so moderator can pick a questioner.

Parameters:

conferenceId – The Conference identifier

mode – The mute mode:

```
public static long MUTE_MODE_CLOSED = 2L
public static long MUTE_MODE_OPEN = 0L
public static long MUTE_MODE_QUESTION = 1L
```

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **muteSession** (long sessionId) – This function should be called when the call referenced by ID should be muted.

Parameters:

sessionId – The Session identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **unMuteSession** (long sessionId) – This function should be called when the call referenced by ID should be un-muted.

Parameters:

sessionId – The Session identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **setCustomName** (long sessionId, String name) – Sets the custom name of the caller referenced by ID.

Parameters:

sessionId – The Session identifier

name – The custom name of the caller

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **setSubconference** (long[] sessionIds, String subconference)
– This function attaches (moves) callers to the sub-conference or detaches them from the sub-conference.

Parameters:

sessionIds – The list of sessions identifiers that need to be attached (moved) to sub-conference or detached from it

subconference – If the parameter is not empty, the sessions are being attached to sub-conference (if they are not currently connected to any of the sub-conferences) or moved to sub-conference (if they are currently connected to another sub-conference); non-empty parameter represents the name of the sub-conference up to 16 characters length (only letters and digits are allowed as the name of the sub-conference); if the parameter is empty the sessions are being detached from the sub-conference

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

Note:

This function was created in version 2.3 and did not exist in previous versions.

- **qaEngage** (long sessionId) – Engages Q&A session for the conference participant referenced by ID. This function should be called when the host selected the call to unmute during the Q&A session.

Parameters:

sessionId – The Session identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **qaDisengage** (long sessionId) – Disengages Q&A session for the conference participant referenced by ID. This function should be called when the host wants to mute the questioner and remove him from the question queue during Q&A session.

Parameters:

sessionId – The Session identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- qaEngageNext*** (long conferneceId) – Enables Q&A session for the first call in the queue.
Parameters:
 conferneceId – The conference identifier
Returns:
 void
Throws Exceptions:
 ServerException
 AccessDeniedException
 ObjectNotFoundException
- qaSetMode*** (long conferenceId, long mode) – Starts, stops or clears Q&A queue for the specific conference.
Parameters:
 conferenceId – The conference identifier
 mode – The Q&A conference mode:

```
public static long QA_MODE_CLOSED = 2L
public static long QA_MODE_OPEN = 0L
public static long QA_MODE_CLEAR = 1L
```

 Note: mode QA_MODE_CLOSED (2L) starts Q&A mode for the conference; mode QA_MODE_OPEN (0L) stops Q&A mode for the conference; mode QA_MODE_CLEAR (1L) clears Q&A queue for the conference.
Returns:
 void
Throws Exceptions:
 ServerException
 AccessDeniedException
 ObjectNotFoundException
- qaMuteMode*** (long conferenceId, long mode) – Mutes or un-mutes active Q&A session for the specific conference.
Parameters:
 conferenceId – The conference identifier
 mode – The Q&A active session mode (0 – unmuted, 1 – muted):

```
public static long MUTE_MODE_OPEN = 0L
public static long MUTE_MODE_RELAXED = 1L
```

Returns:
 void
Throws Exceptions:
 ServerException
 AccessDeniedException
 ObjectNotFoundException
- startConferenceRecording*** (long conferenceId, String pin, String accessCode) – This function starts the conference recording. If the conference call flow is CONF, and the recording method (recording_method) call flow attribute value is either “local+pin” or “remote+pin”, the subscriber’s pin (that usually could be requested from the user) and the conference host access code

should be transferred to this method as its parameters `pin` and `accessCode` to perform recording authorization. Otherwise these parameters should be empty.

Parameters:

`conferenceId` – The Conference identifier
`pin` – The Subscriber's pin
`accessCode` – The host access code

Returns:

`void`

Throws Exceptions:

`ServerException`
`AccessDeniedException`
`ObjectNotFoundException`

- **`stopConferenceRecording`** (`long conferenceId`) – This function stops the conference recording.

Parameters:

`conferenceId` – The Conference identifier

Returns:

`void`

Throws Exceptions:

`ServerException`
`AccessDeniedException`
`ObjectNotFoundException`

- **`setConferenceJobCode`** (`long conferenceId`, `String jobCode`) – Sets the specified job code for the specific started conference.

Parameters:

`conferenceId` – The conference identifier
`jobCode` – The job code that should be set

Returns:

`void`

Throws Exceptions:

`ServerException`
`AccessDeniedException`
`ObjectNotFoundException`

Note:

This function was created in version 2.4 and did not exist in previous versions.

- **`getDialoutSubscribers`** (`long conferenceNumber`) – This function returns the list of `DialoutSubscriber` objects that were previously stored for the specific conference (Conference Info).

Parameters:

`conferenceNumber` – The number of conference which `DialoutSubscriber` objects you would like to get

Returns:

list of `DialoutSubscriber` objects

Throws Exceptions:

`ServerException`
`AccessDeniedException`

ObjectNotFoundException*Note:*

This function was created in version 2.3 and did not exist in previous versions.

- **updateDialoutSubscribers** (long conferenceNumber, DialoutSubscriber[] dialoutSubscribers) – This function updates DialoutSubscribers objects and associates them with the specific conference (Conference Info).

Parameters:

conferenceNumber – The number of conference which DialoutSubscriber objects you would like to update

dialoutSubscribers – The list of DialoutSubscriber objects that represents information necessary to perform the specific dialout

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

Note:

This function was created in version 2.3 and did not exist in previous versions.

- **dialout** (DialoutSubscriber[] dialoutSubscribers, String attributes) – This function initiates outgoing calls to the specified phone numbers and tries to connect specified callers to the specific conference. The list of information necessary to perform the specific dial-outs presented in dialoutSubscribers parameter; if the connection is successful the user will be joined to the conference according to the access code and the conference DNIS number specified in list of DialoutSubscriber objects. Parameter attributes can alter some dial-out logic.

Parameters:

dialoutSubscribers – The list of DialoutSubscriber objects that represents information necessary to perform the specific dialout (such as conference DNIS number, callers phone number to dialout, access code that should be used to connect to the conference, and the custom user name that should be set to the caller)

attributes – The custom attributes that can alter some dial-out logic

Returns:

long count of successful dial-outs, i.e. number of calls that were successfully dialed out to the conference

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

ObjectValidationException

- **dialoutEx** (String phoneNumber, String did, long conferenceNumber, String accessCode, String attributes) – This function initiates outgoing call to specified phone number and tries to connect the participant to the specified conference using the access code provided.

Parameters:

phoneNumber – The phone number to dial-out
 did – The bridge phone number the participant has to be connected to
 conferenceNumber – The actual conference number
 accessCode – The actual access code that should be used to connect to the conference; to connect to sub-conference right after dialout the access code could be formed as *<actual access code>_<sub-conference name>*
 attributes – The SIP Header that should be added to the call (or empty string if no SIP header should be added)

Returns:

long identifier of the created session

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException
 ObjectValidationException

- **startListen** (long conferenceId, long targetId, boolean muted) – This function directly connects and starts listen/talk the conference referenced by ID in the second parameter for the operator conference referenced by ID in the first parameter (the same as *4 on touch tone keypad).

Parameters:

conferenceId – The Operator Conference identifier
 targetId – The target Conference identifier (the conference to listen)
 muted – The flag represents should the operator be muted (*true*) or should the operator be able to talk (*false*) in the connected conference

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **stopListen** (long conferenceId) – This function stops listen the conference for the operator conference referenced by ID.

Parameters:

conferenceId – The Operator Conference identifier

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **joinConferences** (long conferenceId, long targetId) – This function connects, i.e. joins to the target conference referenced by ID in the second parameter with the conference referenced by ID in the first parameter; both conferences will be joined and the callers from these conferences will be able to talk to each other.

Parameters:

conferenceId – The source or Operator Conference identifier to join
 targetId – The target Conference identifier to join

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **disconnectConferences** (long conferenceId) – This function destroys the link created between two conference using *joinConferences* function, i.e. it disconnects the source or Operator conference and stops talking for this conference referenced by ID.

Parameters:

conferenceId – The source or Operator Conference identifier

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **startMonitoring** (long conferenceId) – This function starts conference monitoring (surveillance call) for the operator conference referenced by ID (the same as *1 on touch tone keypad).

Parameters:

conferenceId – The Operator Conference identifier

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

Note:

This function replaces *startScan* used in version 1.4.

- **stopMonitoring** (long conferenceId) – This function stops conference monitoring (surveillance call) for the operator conference referenced by ID.

Parameters:

conferenceId – The Operator Conference identifier

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

Note:

This function replaces *stopScan* used in version 1.4.

- **startTalk** (long conferenceId, long sessionId) – This function starts operator conversation with the user from operator queue; the operator conference is referenced by the identifier specified in the first parameter, the call session is referenced by the identifier specified in the second parameter, but if the call session ID is negative or zero the first user from the operator queue will be taken to start his conversation with the operator (the same as *2 on touch tone keypad).

Parameters:

conferenceId – The Operator Conference identifier

sessionId – The Session identifier to start talking with the operator or 0 to start talking with the first user from the queue

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **dropTalk** (long conferenceId) – This functions stops current conversation with the connected user for the operator conference referenced by ID and returns the user to his conference or ivr (the same as *3 on touch tone keypad); the operator is ready to process the next user.

Parameters:

conferenceId – The Operator Conference identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **rejectSession** (long sessionId) – This functions rejects the specific session referenced by ID from the operator queue, i.e. the conversion with the user will be refused and the user will be removed from the operator queue.

Parameters:

sessionId – The Session identifier of the user that should be removed from the operator queue

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **reattachCall** (long sessionId, String did, String accessCode, long role) – This function attaches the call to the conference. Note, additional CDR record is being created for the reattached call.

Parameters:

sessionId – The Session identifier

did – The bridge phone number the participant has to be connected to

accessCode – The actual access code that should be used to connect to the conference

role – The role (mode) that will be granted to the call in the conference:

```
public static long MODE_HOST = 1L
public static long MODE_LISTENER = 3L
public static long MODE_PARTICIPANT = 2L
```

Note if the role can be determined using the access code it has higher priority than the role.

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **setSessionRole** (long sessionId, long role) – This function changes the role of the call in the conference. Note: no additional CDR record is being created for the call.

Parameters:

sessionId – The Session identifier

role – The role (mode) that will be granted to the call in the conference:

```
public static long MODE_HOST = 1L
public static long MODE_LISTENER = 3L
public static long MODE_PARTICIPANT = 2L
```

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

Note:

This function was created in version 2.3 and did not exist in previous versions.

- **setGainLevel** (long sessionId, long level) – Sets the microphone volume level of the call referenced by ID.

Parameters:

sessionId – The Session identifier

level – The microphone volume level, it could be from -10 till 10 or 255; -10 is the quietest (lowest) sound level, 10 is the loudest (highest) sound level, 255 denotes that the microphone level is being automatically adjusted by the backend

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

Note:

This function was created in version 2.1 and did not exist in previous versions.

- **startPolling** (long conferenceId, String keys) – This function starts polling within specific conference with selected options (the same as #5 on touch tone keypad).

Parameters:

conferenceId – The conference identifier
keys – Available options (digits 1, 2, ..., 9, 0)

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

Note:

This function was created in version 2.1 and did not exist in previous versions.

- **stopPolling** (long conferenceId) – This function stops polling within specific conference referenced by conference number (the same as #5 on touch tone keypad).

Parameters:

conferenceId – The conference identifier

Returns:

void

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

Note:

This function was created in version 2.1 and did not exist in previous versions.

- **getPollingResults** (long conferenceId) – This function allows getting list of polling results for the specific conference referenced by the ID.

Parameters:

conferenceId – The Conference identifier

Returns:

list of PollingResult objects

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

Note:

This function was created in version 2.1 and did not exist in previous versions.

Subscribers' Conferences Management

- **getSubscriberConferences** (long offset, long limit, String filter, String order) – This function returns the list of SubscriberConference objects which match the given filter. Each of subscriber's conference object represents subscriber's conference (either started or not started) and its subscriber information

with up to three conference users (for host, participant, and listener) information describing this conference. So the conference information returned by this method is always unique; if the conference is used by multiple subscribers the subscriber who first created it will be returned.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more SubscriberConference field names or composite (compound) statement relative to Subscriber and Conferences fields names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example pin='12' or pin like'%2%' or subscriberId >= 15.

Empty string or null means no filter.

order - A string specifying SubscriberConference field name and sort direction.

For example "pin" or "email desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Subscriber related acceptable fields for filtering and sorting:

- subscriber.subscriberId
- subscriber.parentId
- subscriber.pin
- subscriber.password
- subscriber.firstName
- subscriber.lastName
- fullName
- subscriber.email
- subscriber.address1
- subscriber.city
- subscriber.state
- subscriber.zip
- subscriber.country
- subscriber.phoneNumber
- confuser.hostAccessCode
- confuser.participantAccessCode
- confuser.listenerAccessCode
- confuser.confuserId
- confuser.dnisId
- confuser.role
- confuser.subscriberId
- confuser.conferenceInfo.conferenceNumber
- confuser.dnis.did

Conference related acceptable fields for filtering and sorting:

- conference.conferenceId
- conference.conferenceNumber
- conference.created ('yyyy.MM.dd/hh:mm' format)
- conference.duration

- conference.participantCnt
- conference.isSecured
- conference.muteMode

Please note: all conference filters are being applied to the active conferences only! Otherwise you can use the confuser.conferenceInfo filters (for example to filter by the conference number you can use confuser.conferenceInfo.conferenceNumber field).

Returns:

list of SubscriberConference objects

Throws Exceptions:

ServerException

AccessDeniedException

Note:

This function was created in version 2.2 and did not exist in previous versions.

- **getSubscriberConferencesCount** (String filter) – This function returns number of SubscriberConference that match the given filter, i.e. the number of unique subscriber's conferences.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more SubscriberConference field names or composite (compound) statement relative to Subscriber and Conferences fields names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example pin='12' or pin like '%2%' or subscriberId >= 15.

Empty string or null means no filter.

Subscriber related acceptable fields for filtering and sorting:

- subscriber.subscriberId
- subscriber.parentId
- subscriber.pin
- subscriber.password
- subscriber.firstName
- subscriber.lastName
- fullName
- subscriber.email
- subscriber.address1
- subscriber.city
- subscriber.state
- subscriber.zip
- subscriber.country
- subscriber.phoneNumber
- confuser.hostAccessCode
- confuser.participantAccessCode
- confuser.listenerAccessCode
- confuser.confuserId
- confuser.dnsId
- confuser.role

- confuser.subscriberId
- confuser.conferenceInfo.conferenceNumber
- confuser.dnis.did

Conference related acceptable fields for filtering and sorting:

- conference.conferenceId
- conference.conferenceNumber
- conference.created ('yyyy.MM.dd/hh:mm' format)
- conference.duration
- conference.participantCnt
- conference.isSecured
- conference.muteMode

Please note: all conference filters are being applied to the active conferences only! Otherwise you can use the confuser.conferenceInfo filters (for example to filter by the conference number you can use confuser.conferenceInfo.conferenceNumber field).

Returns:

long count of SubscriberConference objects

Throws Exceptions:

ServerException

AccessDeniedException

Note:

This function was created in version 2.2 and did not exist in previous versions.

CDRs Management

- ***getConferenceDR*** (long conferenceId) – This function returns full details about the ConferenceDR referenced by the ID.

Parameters:

conferenceId – The Conference identifier

Returns:

ConferenceDR object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- ***getConferenceDRs*** (long offset, long limit, String filter, String order) – This function returns list of ConferenceDRs which are registered for the subscriber. For administrator it returns whole list of records.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more ConferenceDR field names.

Acceptable operators: <=, >=, !=, =, <, >, like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example:

```

conferenceId = 5424
duration > 300 and duration < 400
duration > 300 and conferenceNumber = 160
participantCnt > 2 and participantCnt < 22
created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

```

Accepted fields:

- conferenceId
- conferenceNumber
- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)
- duration
- participantCnt

Empty string or null means no filter.

order - A string specifying ConferenceDR field name and sort direction.

For example "conferenceNumber" or "created desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of ConferenceDR objects

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getConferenceDRsCount** (String filter) – This function returns number of ConferenceDRs stored in local CDR db.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more ConferenceDR field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example:

```

conferenceId = 5424
duration > 300 and duration < 400
duration > 300 and conferenceNumber = 160
participantCnt > 2 and participantCnt < 22
created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

```

Accepted fields:

- conferenceId
- conferenceNumber
- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)
- duration
- participantCnt

Empty string or null means no filter.

Returns:

long count of ConferenceDR objects

Throws Exceptions:

ServerException
AccessDeniedException

- **getSessionDR** (long sessionId) – This function returns full details about the SessionDR referenced by the ID.

Parameters:

sessionId – The Session identifier

Returns:

SessionDR object

Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- **getSessionDRs** (long offset, long limit, String filter, String order) – This function returns list of SessionDRs allowed to view.

Parameters:

offset – zero based offset in recordset.

limit – maximum number of objects to return.

filter – The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more SessionDR field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example:

conferenceId = 5424

created > '2011.08.10/00:00' and created < '2011.08.20/00:00'

created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

Accepted fields:

- conferenceId
- conferenceNumber
- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)
- duration
- role
- joined
- customName
- caller;
- callee;
- addressFrom;
- addressTo;
- accessCode;
- disconnectReason;

Empty string or null means no filter.

order – A string specifying SessionDR field name and sort direction.

For example "created desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of SessionDR objects

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getSessionDRsCount** (String filter) – This function returns number of SessionDRs stored in local CDR db.

Parameters:

filter – The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more SessionDR field names.

Acceptable operators: <=, >=, !=, =, <, >, like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example:

conferenceId = 5424

created > '2011.08.10/00:00' and created < '2011.08.20/00:00'

created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

Accepted fields:

- conferenceId
- conferenceNumber
- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)
- duration
- role
- joined
- customName
- caller;
- callee;
- addressFrom;
- addressTo;
- accessCode;
- disconnectReason;

Empty string or null means no filter.

Returns:

long count of SessionDR objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getReportSummary** (String reportType, String filter) – This function returns reporting summary information for conferences, calls, and operator statistic reports according to the specified filter. This information includes number of conferences, number of calls and total calls duration in seconds.

Parameters:

reportType – The type of the report which summary you would like to get; it could be one of the following values: *conferences* – for conferences report, i.e. summary data for *getConferenceDRs* function will be returned if this reportType is specified; *calls* – for calls

report, i.e. summary data for *getSessionDRs* function will be returned if this is reportType specified; *operators* – for operator statistic report, i.e. summary data for *getOperatorsStatistic* function will be returned if this is reportType specified.

filter – The criteria to use to filter the report data. The criteria should be a simple sql conditional statement started with one or more ConferenceDR / SessionDR field names depending on the report type (see detail information for *getConferenceDRs* and *getSessionDRs* functions).

Acceptable operators: \leq , \geq , \neq , $=$, $<$, $>$, like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

Accepted fields:

- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)

For example:

created > '2010.11.10/00:00' and created < '2010.11.20/00:00'

created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

Returns:

string report summary information depending on the type of the report –

- *conferences*
conferences:<number of conferences>;calls:<number of calls joined to these conferences>;seconds:<total duration of the calls joined to these conferences in seconds>
- *calls*
calls:<number of calls>;seconds:<total duration of these calls in seconds>
- *operators*
total_requests_count:<number of requests users generated to operators>;total_request_answered:<number of requests answered by operators>;average_wait_time:<average time users spent in the queue waiting for operators>;

Throws Exceptions:

ServerException
AccessDeniedException

Note:

This function was created in version 2.2 and did not exist in previous versions.

- **getCustomReportRecords** (String reportType, long offset, long limit, String filter, String order) – This function returns the list of strings; each returned string represents custom report data separated by semicolon (;) depending on the type of the report – either for DNIS report (*dnises*) or for Disconnect report (*disconnects*) according to the specified filter and in the specified order. The information returned by this function includes either DNIS numbers, number of calls and total duration of the calls in seconds for DNIS report or disconnect reasons, disconnect causes (standard Q.931 (ISDN) disconnect cause codes list can be used to decode the disconnect reasons in ISDN messages¹), and number of disconnects for Disconnect report.

¹ You can find additional information regarding to Disconnect Cause Codes, including the complete list of the codes using the following URLs:

- <http://en.wikipedia.org/wiki/Q.931>

Parameters:

reportType – The type of the report which data you would like to get; it could be one of the following values: *dnises* – for DNIS report, i.e. CDR data grouped by DNIS number will be returned if this reportType is specified; *disconnects* – for Disconnect report, i.e. CDR data grouped by the disconnect reason will be returned if this reportType is specified.

offset – zero based offset in recordset.

limit – maximum number of objects to return.

filter – The criteria to use to filter the report data. The criteria should be a simple sql conditional statement started with one or more field names depending on the report type (see detail information below in accepted fields lists).

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

Accepted fields for both report types:

- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)
- count

Accepted fields for *dnises*:

- dnis
- duration

Accepted fields for *disconnects*:

- disconnectReason
- disconnectCause

For example:

created > '2010.11.10/00:00' and created < '2010.11.20/00:00'

created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

Note: if you use filter by count, this condition clause should be the last one.

order – A string specifying one of the above mentioned fields names and sort direction.

For example "count desc". The default direction is asc and can be omitted.

Returns:

list of strings that represent semicolon-separated custom report information depending on the type of the report –

- *dnises*
 <DNIS number>;<number of calls to this DNIS number>;<total duration of the calls to this DNIS number in seconds>
- *disconnects*
 <disconnect reason>;<disconnect cause>;<number of calls disconnected by this reason and cause>

Throws Exceptions:

ServerException

AccessDeniedException

Note:

This function was created in version 2.2 and did not exist in previous versions.

-
- <http://networking.ringofsaturn.com/Routers/isdncausecodes.php>

- ***getCustomReportRecordsCount*** (String reportType, String filter) – This function returns number of the specific custom report records that fit with the specified filter. If the custom report type is *dnises* (DNIS report) this function returns number of different DNIS numbers, if the custom report type is *disconnects* (Disconnect report) this function returns number of different disconnect reasons – both according to the filter that was specified.

Parameters:

reportType – The type of the report which data you would like to calculate; it could be one of the following values: *dnises* – for DNIS report, i.e. CDR data grouped by DNIS number will be counted up if this reportType is specified; *disconnects* – for Disconnect report, i.e. CDR data grouped by the disconnect reason will be counted up if this reportType is specified.

filter – The criteria to use to filter the report data. The criteria should be a simple sql conditional statement started with one or more field names depending on the report type (see detail information below in accepted fields lists).

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

Accepted fields for both report types:

- created ('yyyy.MM.dd/hh:mm' format)
- dropped ('yyyy.MM.dd/hh:mm' format)
- count

Accepted fields for *dnises*:

- dnis
- duration

Accepted fields for *disconnects*:

- disconnectReason
- disconnectCause

For example:

created > '2010.11.10/00:00' and created < '2010.11.20/00:00'

created like '%11/04/2011%' and dropped > '2011.11.05/09:00'

Note: if you use filter by count, this condition clause should be the last one.

Returns:

long count of the custom report records depending on the type of the report –

- *dnises*
count of different DNIS numbers
- *disconnects*
count of different disconnect reasons

Throws Exceptions:

ServerException
AccessDeniedException

Note:

This function was created in version 2.2 and did not exist in previous versions.

- ***listAudioFiles*** (long conferenceNumber, String pattern) – This function returns the list of user's audio files (recordings and uploaded streaming audio-files) according to the specified pattern and conference number.

Parameters:

conferenceNumber – The conference number (note: it is not conferenceId)
 pattern – The filename wildcard pattern

Returns:

list of FileDescriptor objects

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **deleteAudioFiles** (long conferenceNumber, String pattern) – This function deletes user's audio files (recordings and uploaded streaming audio) according to the specified pattern and conference number.

Parameters:

conferenceNumber – The conference number (note: it is not conferenceId)
 pattern – The filename wildcard pattern

For example:

deleteAudioFiles(223344, "16777264.")* to delete all audio files for the conference with the number 223344 and the identifier 16777264)

Returns:

long number of deleted files

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **updateFileDescriptor** (long conferenceNumber, FileDescriptor fileDescriptor) – This function allows to change the file description only.

Parameters:

conferenceNumber – The conference number (note: it is not conferenceId)
 fileDescriptor – The FileDescriptor object (with correct description) to update

Returns:

void

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException
 ObjectValidationException

- **shareRecording** (long conferenceId, DateTime expirePeriod, boolean allowDownload) – Usually to get access to the recorded conference files the user should be authorized on the bridge. This function should be used if it is necessary to generate the link to the conference audio files that will be available without authorization; this link will be temporary available and it will be valid limited time only; using this URL any users will be able to listen (download) recording without authorization. The recorded files URL is stored in the recordingUrl property of the ConferenceDR object; the shared recorded files URL, created by this function, is stored in the sharedRecordingUrl property of the ConferenceDR object.

Parameters:

conferenceId – The Conference identifier reference number
 expirePeriod – The period of time over which the shared link will be invalidated
 allowDownload – The flag showing whether mp3 download is allowed or

disallowed

Returns:

string shared recording URL

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

- **getDtmfHistory** (long sessionId) – This function returns list of DTMF commands for the specific Session or SessionDR object referenced by the ID.

Parameters:

sessionId – The Session identifier

Returns:

list of DtmfEvent objects

Throws Exceptions:

ServerException
 AccessDeniedException
 ObjectNotFoundException

Note:

This function was created in version 2.1 and did not exist in previous versions.

- **getOperatorsStatistic** (long offset, long limit, String filter, String order) – This function allows getting list of OperatorStatistic objects. To implement paging you can call it with the proper offset and limit.

Parameters:

offset - zero based offset in recordset.
 limit - maximum number of objects to return.
 filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more OperatorStatistic field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example:

created like '%11/04/2011%' and created > '2011.11.05/09:00'

Accepted fields:

- created ('yyyy.MM.dd/hh:mm' format)

Empty string or null means no filter.

order - A string specifying OperatorStatistic field name and sort direction.

For example "created desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of OperatorStatistic objects

Throws Exceptions:

ServerException
 AccessDeniedException

ObjectNotFoundException*Note:*

This function was created in version 2.1 and did not exist in previous versions.

- ***getOperatorsStatisticCount*** (String filter) – This function returns number of OperatorStatistic objects according to specified filter.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more OperatorStatistic field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example:

created like '%11/04/2011%' and created > '2011.11.05/09:00'

Accepted fields:

- created ('yyyy.MM.dd/hh:mm' format)

Empty string or null means no filter.

Returns:

long count of OperatorStatistic objects

Throws Exceptions:

ServerException

AccessDeniedException

Note:

This function was created in version 2.1 and did not exist in previous versions.

Call Flow and DNIS Management

- ***getCallFlow*** (long callFlowId) – This function returns full details about the CallFlow referenced by the ID provided.

Parameters:

callFlowId – The CallFlow identifier

Returns:

CallFlow object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- ***getCallFlows*** (long offset, long limit, String filter, String order) – This function returns list of CallFlows which match the filter provided. There are two parameters *offset* and *limit* to help to implement paging on the web application. Please note that by default the *attributesTemplate* field in returned call flows is populated with call flow attributes for each returned call flow; if you are getting large number of call flows and/or would like to avoid huge amount of data to be transferred in case if big request is processing call flow objects, to suppress call flow attributes for call flows using this command (i.e. to return null as *attributesTemplate* field value), you should add *stripAttributes* keyword (case-sensitive) to your filter parameter.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more CallFlow field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example name='12' or name like '%2%' or collFlowId >= 15.

Accepted fields:

- callFlowId
- name
- path

Filter samples:

name='SPECTEL'

name like '%C%' stripAttributes

stripAttributes

Empty string or null means no filter.

order - A string specifying CallFlow field name and sort direction.

For example "name" or "name desc". The default direction is asc and can be omitted.

Empty string or null means no order.

Returns:

list of CallFlow objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getCallFlowsCount** (String filter) – This function returns number of CallFlows on the bridge which match the filter provided.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more CallFlow field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example name='12' or name like '%2%' or collFlowId >= 15.

Accepted fields:

- callFlowId
- name
- path

Empty string or null means no filter.

Returns:

long count of CallFlow objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getDNIS** (long dnisId) – This function returns full details about the DNIS referenced by the ID provided.

Parameters:

dnisId – The DNIS identifier

Returns:

DNIS object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getDNISCount** (String filter) – This function returns number of DNISes on the bridge which match the filter provided.

Parameters:

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more DNIS field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

For example did='12' or did like'%2%' or callFlowId >= 15.

Accepted fields:

- callFlowId
- dnisId
- did
- description
- alias.mask
- alias.description

Empty string or null means no filter.

Returns:

long count of DNIS objects

Throws Exceptions:

ServerException

AccessDeniedException

- **getDNISes** (long offset, long limit, String filter, String order) – This function returns list of DNISes (phone numbers) which match the filter provided. There are two parameters *offset* and *limit* to help to implement paging on the web application. Please note that by default the *attributes* field in DNIS is populated with call flow attributes for each returned DNIS number; if you are getting large number of DNISes and/or would like to avoid huge amount of data to be transferred in case if big request is processing DNIS objects, to suppress call flow attributes for DNIS numbers using this command (i.e. to return null as *attributes* field value), you should add *stripAttributes* keyword (case-sensitive) to your filter parameter.

Parameters:

offset - zero based offset in recordset.

limit - maximum number of objects to return.

filter - The criteria to use to filter the rows. The criteria should be a simple sql conditional statement started with one or more DNIS field names.

Acceptable operators: <= , >= , != , = , < , > , like (case sensitive), ilike (case insensitive), and. Note: or-clauses and brackets are not supported.

Filter samples:

did ='12'
 did like '%2%' or callFlowId >= 15
 did like '%12%' stripAttributes
 alias.mask like '%22%' and description like 'test%' stripAttributes
 stripAttributes

Empty string or null means no filter.

order - A string specifying DNIS field name and sort direction.

For example "name" or "name desc". The default direction is asc and can be omitted.

Accepted fields:

- callFlowId
- dnisId
- did
- description
- alias.mask
- alias.description

Empty string or null means no order.

Returns:

list of DNIS objects

Throws Exceptions:

ServerException

AccessDeniedException

- **updateCallFlow** (CallFlow callflow) – The method updates CallFlow object.

Parameters:

callflow – The CallFlow object

Returns:

updated CallFlow object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **createDNIS** (DNIS dnis) – This function creates a new DNIS with the details specified in the input parameter. Please note that only administrator can create new DNISes.

Parameters:

dnis – The DNIS object

Returns:

created DNIS object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **updateDNIS** (DNIS dnis) – This function updates DNIS with the new information. Please note that only administrator has a permission to update DNIS.

Parameters:

dnis – The DNIS object

Returns:

updated DNIS object

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **deleteDNIS** (long dnisId) – This function deletes DNIS referenced by the ID from the server. When DNIS is being deleted all confusers (conference accounts) associated with this DNIS also are being deleted. Please note that only administrator has a permission to delete DNIS.

Parameters:

dnisId – The DNIS identifier

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

- **getServerAttributes** () – This function returns list of system attributes (settings) registered on the bridge along with the current values, i.e. separate Attribute Name – Attribute Value pairs.

Returns:

list of attributes (Attribute objects)

Throws Exceptions:

ServerException

AccessDeniedException

- **setServerAttributes** (Attribute[] attributes) – This function allows setting new values to the system attributes, i.e. separate Attribute Name – Attribute Value pairs.

Parameters:

attributes – The list of Attribute objects that need to be updated

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **getAttributesDescription** (long callflowId) – This function returns the list of call flow attributes with their description for the CallFlow object specified by its identifier. The attribute description is being returned in the *value* field of the returned call flow *Attribute* object.

Parameters:

callflowId – The CallFlow identifier

Returns:

list of Attribute objects

Throws Exceptions:

ServerException

AccessDeniedException

ObjectNotFoundException

Backend and Frontend Services Management

- **getVersion** () – Returns version of the installed software (like 2.4.41.13 for the current version).

Returns:

string product version

Throws Exceptions:

ServerException

AccessDeniedException

ObjectValidationException

- **getBackendInfo** () – Returns some statistic about backend.

Returns:

string status of Backend Service in the textual format

Returns Sample:

```
Welcome to WYDE.MPs admin console 2.4.41.13 compiled Aug 9
2011>Started: Thu Aug 11 11:51:23 2011Call: Now=0; Total=8;
Peak=3; Last=Fri Aug 12 17:24:04 2011Conf: Now=0; Total=3;
Peak=1; Last=Fri Aug 12 16:49:45 2011Brds: Now=1
```

Throws Exceptions:

ServerException

AccessDeniedException

- **getFrontendInfo** (String group) – Returns some statistic about frontend.

Parameters:

group – group name, for example cmdcount-show, confcount-show,
errcount-show, partcount-show, etc. (service functions)

Returns:

string status of Frontend Service in the textual format

Throws Exceptions:

ServerException

AccessDeniedException

- **isBackendUp** () – Returns true if backend is up and running.

Returns:

Boolean true if Backend Service is OK, otherwise – false

Throws Exceptions:

ServerException

AccessDeniedException

- **isFrontendUp** () – Returns true if frontend is up and running and state can not be determined.

Returns:

Boolean true if frontend is up and running, otherwise – false

Throws Exceptions:

ServerException

AccessDeniedException

- **startBackend** () – Tries to start backend with the settings from the DB.

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

- **stopBackend** () – Tries to stop backend.

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

- **startFrontend** () – Tries to start frontend with the settings from the DB.

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

- **stopFrontend** () – Tries to stop frontend.

Returns:

void

Throws Exceptions:

ServerException

AccessDeniedException

Exceptions

- **ServerException** – This exception is thrown to indicate that internal server-side error occurred.
- **AccessDeniedException** – This exception is thrown to indicate that a requested access (to an object or method) is denied. The request access can be denied according to the security policy.
- **ObjectNotFoundException** – This exception is thrown to indicate that requested object can not be found.
- **ObjectValidationException** – This exception is thrown to indicate that specified object can not be saved in its current state. Exception contains the collection of field names that should be checked in `fieldname` property. There are two possible reasons: this field is mandatory (if current value is null) or incorrect value.

If any of these exceptions occurred for all these exceptions `msg` property contains detail description of the error, i.e. the message that could help to determine the reason of the error.

Constants

- ***Subscriber***

```
public static int ROLE_ADMIN = 1L
public static int ROLE_OPERATOR = 2L
public static int ROLE_USER = 3L
```

- ***Conference***

```
public static long MUTE_MODE_CLOSED = 2L
public static long MUTE_MODE_OPEN = 0L
public static long MUTE_MODE_QUESTION = 1L
public static long QA_MODE_CLOSED = 2L
public static long QA_MODE_OPEN = 0L
public static long QA_MODE_CLEAR = 1L
public static long HOLD_MODE_HOST = 1L
public static long HOLD_MODE_PARTICIPANT = 2L
public static long HOLD_MODE_LISTENER = 4L
public static long CONFERENCE_REGULAR = 0L
public static long CONFERENCE_OPERATOR = 1L
public static long CONFERENCE_LISTEN = 2L
public static long CONFERENCE_AUTOLISTEN = 3L
public static long CONFERENCE_AUTOLISTEN_SLEEP = 4L
public static long CONFERENCE_TALK = 5L
```

- ***Session***

```
public static long MODE_HOST = 1L
public static long MODE_LISTENER = 3L
public static long MODE_PARTICIPANT = 2L
public static long MODE_UNDEFINED = 0L
public static long MODE_DC_LINK = 8L
public static long OPERATOR_STATUS_IDLE = 0L
public static long OPERATOR_STATUS_WAIT = 1L
public static long OPERATOR_STATUS_TALK = 2L
public static long QA_STATUS_ACTIVE = 2L
public static long QA_STATUS_IDLE = 0L
public static long QA_STATUS_RISEDHAND = 1L
public static long STATUS_CLOSED = 3L
public static long STATUS_CONFERENC = 2L
public static long STATUS_DIALING = 4L
public static long STATUS_IVR = 1L
```

- ***SessionDR***

```
public static long INITIATOR_BRIDGE = 2L
public static long INITIATOR_UNDEFINED = 0L
public static long INITIATOR_USER = 1L
```

- **Attribute**

```
public static long TYPE_STRING = 0L
public static long TYPE_BILLINGRULE = 1L
public static long TYPE_INT = 2L
public static long TYPE_DTMF = 3L
public static long TYPE_ROLE = 4L
public static long TYPE_CHOICE = 5L
public static long ROLE_CALLFLOW = 3L
public static long ROLE_CONFERENCE = 1L
public static long ROLE_DNIS = 0L
```


Appendix A: Code Samples

WYDE Web Services Initialization

Sample of WYDE Web Services Initialization

```

/*
Sample of WYDE Web Services Initialization
*/
using System;
using System.Xml;
using System.Text;
using WYDEWS.jAdmin;

namespace WYDEWS
{
    /// <summary>
    /// Represents base class for the WYDE web services class (jAdmin)
    /// </summary>
    class myJAdmin : jAdmin.jAdmin
    {
        protected override System.Net.WebRequest GetWebRequest(Uri uri)
        {
            System.Net.HttpWebRequest webRequest =
                (System.Net.HttpWebRequest)base.GetWebRequest(uri);
            webRequest.ProtocolVersion =
                System.Net.HttpVersion.Version10;
            return webRequest;
        }
    }

    /// <summary>
    /// Represents entire jAdmin web service helper class
    /// </summary>
    public class clsjAdmin
    {
        #region [ private fields ]
        private myJAdmin ws;
        private String mLastError;
        #endregion

        #region [ constructors and destructors ]
        /// <summary>
        /// Initializes a new instance of the class (constructor).
        /// </summary>
        public clsjAdmin()
        {
            const String PROC = "clsjAdmin(constructor)";
            String strZone = "";
            try
            {
                // Initialize web service
                strZone = "new myJAdmin()";
                ws = new myJAdmin();

                // WebServiceURL, WebServiceUser, WebServicePassword,
                // WebServiceTimeout parameters: app.config
                strZone = "set web service Url";
                if (!String.IsNullOrEmpty(Utls.AppSettings("WebServiceURL")))
                {
                    ws.Url = Utls.AppSettings("WebServiceURL");
                }

                strZone = "new NetworkCredential()";
                if (!String.IsNullOrEmpty(Utls.AppSettings("WebServiceUser")))
                {
                    ws.Credentials = new System.Net.NetworkCredential(

```

```

        Utils.AppSettings("WebServiceUser"),
        Utils.AppSettings("WebServicePassword"));
    }

    strZone = "set web service Timeout";
    if (Utils.Data2Int(Utils.AppSettings("WebServiceTimeout")) > 0)
    {
        ws.Timeout = Utils.Data2Int(Utils.AppSettings("WebServiceTimeout"));
    }

    strZone = "getVersion";
    ws.getVersion(); // Check if initialization was successful
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName + "." +
        PROC + " (" + strZone + "): " + ex.Message;
}
}
/// <summary>
/// Performs deterministic clean up of the class (destructor).
/// </summary>
~clsjAdmin()
{
    ws.Dispose();
}
#endregion

#region [ properties ]
/// .....
#endregion

#region [ private methods ]
/// .....
#endregion

#region [ public methods ]
/// .....
#endregion
}
}

```

app.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" >
      <section name="WYDEWS.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
    </sectionGroup>
  </configSections>
  <system.serviceModel>
    <bindings />
    <client />
  </system.serviceModel>
  <appSettings>
    <add key="WebServiceURL" value="http://192.168.1.4/dnca/jAdmin"/>
    <add key="WebServiceUser" value="admin"/>
    <add key="WebServicePassword" value="admin"/>
    <add key="WebServiceTimeout" value="120000"/> <!-- in milliseconds -->
  </appSettings>
  <applicationSettings>
    <WYDEWS.Properties.Settings>
      <setting name="WYDEWS_jAdmin_jAdmin" serializeAs="String">
        <value>http://192.168.1.4/dnca/jAdmin</value>
      </setting>
    </WYDEWS.Properties.Settings>
  </applicationSettings>
</configuration>

```

*Web Methods' XML Requests and Responses***Sample of XML Request for Function with Multiple Parameters Sent**

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <getSessionDRs xmlns="dnca">
      <offset>0</offset>
      <limit>3</limit>
      <filter>created>='2009-10-01' and conferenceNumber=667788</filter>
      <order />
    </getSessionDRs>
  </soap:Body>
</soap:Envelope>
```

Sample of XML Response for Function with List of Objects Received

```

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <ns1:getSessionDRsResponse xmlns:ns1="dnca">
      <ns1:out>
        <ns2:SessionDR xmlns:ns2="http://data.dnca.datanaut.com">
          <accessCode xmlns="http://data.dnca.datanaut.com"> 11233 </accessCode>
          <addressFrom xmlns="http://data.dnca.datanaut.com">
            "MZ 2003" <sip:3131@192.168.1.5> </addressFrom>
          <addressTo xmlns="http://data.dnca.datanaut.com">
            "12_11233" <sip:12_11233@192.168.1.5> </addressTo>
          <bridgeName xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
          <callee xmlns="http://data.dnca.datanaut.com"> 12 </callee>
          <caller xmlns="http://data.dnca.datanaut.com"> 3131 </caller>
          <conferenceId xmlns="http://data.dnca.datanaut.com">
            39750 </conferenceId>
          <conferenceNumber xmlns="http://data.dnca.datanaut.com">
            667788 </conferenceNumber>
          <created xmlns="http://data.dnca.datanaut.com">
            2009-10-30T08:49:08-07:00 </created>
          <customName xmlns="http://data.dnca.datanaut.com">
            'MZ 2003' </customName>
          <disconnectInitiator xmlns="http://data.dnca.datanaut.com">
            1 </disconnectInitiator>
          <disconnectReason xmlns="http://data.dnca.datanaut.com">
            Normal </disconnectReason>
          <duration xmlns="http://data.dnca.datanaut.com">
            91 </duration>
          <jobCode xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
          <joined xmlns="http://data.dnca.datanaut.com">
            2009-10-30T08:49:10-07:00 </joined>
          <nodeName xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
          <role xmlns="http://data.dnca.datanaut.com"> 1 </role>
          <sessionId xmlns="http://data.dnca.datanaut.com"> 142018 </sessionId>
          <subscriberId xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
        </ns2:SessionDR>
        <ns2:SessionDR xmlns:ns2="http://data.dnca.datanaut.com">
          <accessCode xmlns="http://data.dnca.datanaut.com"> 1233 </accessCode>
          <addressFrom xmlns="http://data.dnca.datanaut.com">
            "unknown" <sip:192.168.1.9> </addressFrom>
          <addressTo xmlns="http://data.dnca.datanaut.com">
            "12_1233" <sip:12_1233@192.168.1.5> </addressTo>
          <bridgeName xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
          <callee xmlns="http://data.dnca.datanaut.com"> 12 </callee>
          <caller xmlns="http://data.dnca.datanaut.com" />
          <conferenceId xmlns="http://data.dnca.datanaut.com">
            39749 </conferenceId>
          <conferenceNumber xmlns="http://data.dnca.datanaut.com">
            667788 </conferenceNumber>
          <created xmlns="http://data.dnca.datanaut.com">
            2009-10-30T08:47:38-07:00 </created>
          <customName xmlns="http://data.dnca.datanaut.com">
            'unknown' </customName>

```

```

<disconnectInitiator xmlns="http://data.dnca.datanaut.com">
  1</disconnectInitiator>
<disconnectReason xmlns="http://data.dnca.datanaut.com">
  Normal</disconnectReason>
<duration xmlns="http://data.dnca.datanaut.com">87</duration>
<jobCode xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
<joined xmlns="http://data.dnca.datanaut.com">
  2009-10-30T08:47:40-07:00</joined>
<nodeName xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
<role xmlns="http://data.dnca.datanaut.com">2</role>
<sessionId xmlns="http://data.dnca.datanaut.com">142017</sessionId>
<subscriberId xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
</ns2:SessionDR>
<ns2:SessionDR xmlns:ns2="http://data.dnca.datanaut.com">
  <accessCode xmlns="http://data.dnca.datanaut.com">1233</accessCode>
  <addressFrom xmlns="http://data.dnca.datanaut.com">
    "MZ 2003"<sip:3131@192.168.1.5></addressFrom>
  <addressTo xmlns="http://data.dnca.datanaut.com">
    "12_1233" <sip:12_1233@192.168.1.5></addressTo>
  <bridgeName xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
  <callee xmlns="http://data.dnca.datanaut.com">12</callee>
  <caller xmlns="http://data.dnca.datanaut.com">3131</caller>
  <conferenceId xmlns="http://data.dnca.datanaut.com">
    39749</conferenceId>
  <conferenceNumber xmlns="http://data.dnca.datanaut.com">
    667788</conferenceNumber>
  <created xmlns="http://data.dnca.datanaut.com">
    2009-10-30T08:45:49-07:00</created>
  <customName xmlns="http://data.dnca.datanaut.com">
    'MZ 2003'</customName>
  <disconnectInitiator
    xmlns="http://data.dnca.datanaut.com">1</disconnectInitiator>
  <disconnectReason xmlns="http://data.dnca.datanaut.com">
    Normal</disconnectReason>
  <duration xmlns="http://data.dnca.datanaut.com">195</duration>
  <jobCode xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
  <joined xmlns="http://data.dnca.datanaut.com">
    2009-10-30T08:45:51-07:00</joined>
  <nodeName xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
  <role xmlns="http://data.dnca.datanaut.com">2</role>
  <sessionId xmlns="http://data.dnca.datanaut.com">142016</sessionId>
  <subscriberId xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
</ns2:SessionDR>
</ns1:out>
</ns1:getSessionDRsResponse>
</soap:Body>
</soap:Envelope>

```

Sample of XML Request for Function with the Object Parameter Sent

```

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <createSubscriber xmlns="dnca">
      <s>
        <address1 xsi:nil="true" xmlns="http://data.dnca.datanaut.com" />
        <address2 xsi:nil="true" xmlns="http://data.dnca.datanaut.com" />
        <city xmlns="http://data.dnca.datanaut.com">New-York</city>
        <confusers xmlns="http://data.dnca.datanaut.com">
          <Confuser>
            <accessCode>201130</accessCode>
            <attributes xsi:nil="true" />
            <conferenceInfo>
              <description>MMC_JKRAFT</description>
            </conferenceInfo>
            <dnisId>19</dnisId>
            <role>1</role>
          </Confuser>
          <Confuser>
            <accessCode>637387</accessCode>
            <attributes xsi:nil="true" />
            <conferenceInfo xsi:nil="true" />
            <dnisId>19</dnisId>
            <role>2</role>
          </Confuser>
          <Confuser>
            <accessCode>451665</accessCode>
            <attributes xsi:nil="true" />
            <conferenceInfo xsi:nil="true" />
            <dnisId>19</dnisId>
            <role>3</role>
          </Confuser>
        </confusers>
        <country xmlns="http://data.dnca.datanaut.com">US</country>
        <details xsi:nil="true" xmlns="http://data.dnca.datanaut.com" />
        <email xmlns="http://data.dnca.datanaut.com">
          jkraft@phone-mobile.com</email>
        <firstName xmlns="http://data.dnca.datanaut.com">Julie</firstName>
        <lastName xmlns="http://data.dnca.datanaut.com">Kraft</lastName>
        <password xmlns="http://data.dnca.datanaut.com">321</password>
        <phoneNumber xmlns="http://data.dnca.datanaut.com">
          (204) 221-7600</phoneNumber>
        <pin xmlns="http://data.dnca.datanaut.com">jkraft</pin>
        <zip xsi:nil="true" xmlns="http://data.dnca.datanaut.com" />
      </s>
    </createSubscriber>
  </soap:Body>
</soap:Envelope>

```

Sample of XML Response for Function with the Object Received

```

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <ns1:createSubscriberResponse xmlns:ns1="dnca">
      <ns1:out>
        <address1 xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
        <address2 xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
        <city xmlns="http://data.dnca.datanaut.com">New-York</city>
        <confusers xmlns="http://data.dnca.datanaut.com">
          <Confuser>
            <accessCode>201130</accessCode>
            <attributes>
              <Attribute>
                <enumValues />
                <isOverridden>false</isOverridden>
                <name>call_announceparticipantcount</name>
                <role>1</role>
                <type>0</type>
                <value>hpl</value>
              </Attribute>
              <Attribute>
                <enumValues />
                <isOverridden>false</isOverridden>
                <name>call_exit_dtmf</name>
                <role>1</role>
                <type>0</type>
                <value />
              </Attribute>
              <Attribute>
                <enumValues />
                <isOverridden>false</isOverridden>
                <name>call_instructions_dtmf</name>
                <role>1</role>
                <type>0</type>
                <value>hp</value>
              </Attribute>
              <Attribute>
                <enumValues />
                <isOverridden>false</isOverridden>
                <name>call_mute_dtmf</name>
                <role>1</role>
                <type>0</type>
                <value>hp</value>
              </Attribute>
              <Attribute>
                <enumValues />
                <isOverridden>false</isOverridden>
                <name>call_operator_dtmf</name>
                <role>1</role>
                <type>0</type>
                <value />
              </Attribute>
              <Attribute>
                <enumValues />
              </Attribute>
            </attributes>
          </Confuser>
        </confusers>
      </ns1:out>
    </ns1:createSubscriberResponse>
  </soap:Body>
</soap:Envelope>

```



```

    <isOverridden>false</isOverridden>
    <name>call_participantsnumber_dtmf</name>
    <role>1</role>
    <type>0</type>
    <value>hp</value>
  </Attribute>
  <Attribute>
    <enumValues>on,off</enumValues>
    <isOverridden>false</isOverridden>
    <name>conference_callerdb</name>
    <role>1</role>
    <type>0</type>
    <value>off</value>
  </Attribute>
  <Attribute>
    <enumValues />
    <isOverridden>false</isOverridden>
    <name>conference_dialout_dtmf</name>
    <role>1</role>
    <type>0</type>
    <value>h</value>
  </Attribute>
  <Attribute>
    <enumValues />
    <isOverridden>false</isOverridden>
    <name>conference_entryexittones_dtmf</name>
    <role>1</role>
    <type>0</type>
    <value />
  </Attribute>
  <Attribute>
    <enumValues>on,off</enumValues>
    <isOverridden>false</isOverridden>
    <name>conference_entrytones</name>
    <role>1</role>
    <type>0</type>
    <value>on</value>
  </Attribute>
  <Attribute>
    <enumValues>on,off</enumValues>
    <isOverridden>false</isOverridden>
    <name>conference_exittones</name>
    <role>1</role>
    <type>0</type>
    <value>on</value>
  </Attribute>
  <Attribute>
    <enumValues>false,true</enumValues>
    <isOverridden>false</isOverridden>
    <name>conference_hold_participant</name>
    <role>1</role>
    <type>0</type>
    <value>false</value>
  </Attribute>
  <Attribute>
    <enumValues />
    <isOverridden>false</isOverridden>
    <name>conference_lock_dtmf</name>
    <role>1</role>

```

```

        <type>0</type>
        <value>h</value>
    </Attribute>
    <Attribute>
        <enumValues />
        <isOverridden>false</isOverridden>
        <name>conference_maxcalls</name>
        <role>1</role>
        <type>2</type>
        <value>-1</value>
    </Attribute>
    <Attribute>
        <enumValues />
        <isOverridden>false</isOverridden>
        <name>conference_moh</name>
        <role>1</role>
        <type>0</type>
        <value>default</value>
    </Attribute>
    <Attribute>
        <enumValues />
        <isOverridden>false</isOverridden>
        <name>conference_mute_dtmf</name>
        <role>1</role>
        <type>0</type>
        <value>h</value>
    </Attribute>
    <Attribute>
        <enumValues>open,relaxed,strict</enumValues>
        <isOverridden>false</isOverridden>
        <name>conference_mute_listener</name>
        <role>1</role>
        <type>0</type>
        <value>strict</value>
    </Attribute>
    <Attribute>
        <enumValues />
        <isOverridden>false</isOverridden>
        <name>conference_qa_dtmf</name>
        <role>1</role>
        <type>0</type>
        <value>h</value>
    </Attribute>
    <Attribute>
        <enumValues>on,off</enumValues>
        <isOverridden>false</isOverridden>
        <name>conference_realtime</name>
        <role>1</role>
        <type>0</type>
        <value>off</value>
    </Attribute>
    <Attribute>
        <enumValues>first,moderator</enumValues>
        <isOverridden>false</isOverridden>
        <name>conference_start_how</name>
        <role>1</role>
        <type>0</type>
        <value>first</value>
    </Attribute>

```

```

<Attribute>
  <enumValues />
  <isOverridden>false</isOverridden>
  <name>conference_start_wait</name>
  <role>1</role>
  <type>2</type>
  <value>300</value>
</Attribute>
<Attribute>
  <enumValues>last,moderator</enumValues>
  <isOverridden>false</isOverridden>
  <name>conference_stop_how</name>
  <role>1</role>
  <type>0</type>
  <value>last</value>
</Attribute>
<Attribute>
  <enumValues />
  <isOverridden>false</isOverridden>
  <name>conference_stop_wait</name>
  <role>1</role>
  <type>2</type>
  <value>0</value>
</Attribute>
<Attribute>
  <enumValues />
  <isOverridden>false</isOverridden>
  <name>recording_dtmf</name>
  <role>1</role>
  <type>0</type>
  <value>h</value>
</Attribute>
<Attribute>
  <enumValues>last,moderator</enumValues>
  <isOverridden>false</isOverridden>
  <name>recording_stop_how</name>
  <role>1</role>
  <type>0</type>
  <value>last</value>
</Attribute>
<Attribute>
  <enumValues />
  <isOverridden>false</isOverridden>
  <name>recording_stop_wait</name>
  <role>1</role>
  <type>2</type>
  <value>0</value>
</Attribute>
</attributes>
<conferenceInfo>
  <description>MMC_JKRAFT</description>
  <conferenceNumber>916551</conferenceNumber>
</conferenceInfo>
<confuserId>45</confuserId>
<dnisId>19</dnisId>
<role>1</role>
<subscriberId>26</subscriberId>
</Confuser>
<Confuser>

```

```

    <accessCode>637387</accessCode>
    <attributes />
    <conferenceInfo>
      <description>MMC_JKRAFT</description>
      <conferenceNumber>916551</conferenceNumber>
    </conferenceInfo>
    <confuserId>44</confuserId>
    <dnisId>19</dnisId>
    <role>2</role>
    <subscriberId>26</subscriberId>
  </Confuser>
  <Confuser>
    <accessCode>451665</accessCode>
    <attributes />
    <conferenceInfo>
      <description>MMC_JKRAFT</description>
      <conferenceNumber>916551</conferenceNumber>
    </conferenceInfo>
    <confuserId>46</confuserId>
    <dnisId>19</dnisId>
    <role>3</role>
    <subscriberId>26</subscriberId>
  </Confuser>
</confusers>
<country xmlns="http://data.dnca.datanaut.com">US</country>
<created xmlns="http://data.dnca.datanaut.com">
  2009-10-12T00:00:00-07:00</created>
<details xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
<email xmlns="http://data.dnca.datanaut.com">
  jkraft@phone-mobile.com</email>
<firstName xmlns="http://data.dnca.datanaut.com">Julie</firstName>
<lastName xmlns="http://data.dnca.datanaut.com">Kraft</lastName>
<parentId xmlns="http://data.dnca.datanaut.com">1</parentId>
<password xmlns="http://data.dnca.datanaut.com">321</password>
<phoneNumber xmlns="http://data.dnca.datanaut.com">
  (204) 221-7600</phoneNumber>
<pin xmlns="http://data.dnca.datanaut.com">jkraft</pin>
<role xmlns="http://data.dnca.datanaut.com">3</role>
<subscriberId xmlns="http://data.dnca.datanaut.com">26</subscriberId>
<zip xmlns="http://data.dnca.datanaut.com" xsi:nil="true" />
</ns1:out>
</ns1:createSubscriberResponse>
</soap:Body>
</soap:Envelope>

```

Subscribers Management

Sample of Subscriber and his Conference Accounts Creation (Sample_ManageSubscriber1)

```

/*
Sample of Subscriber and his Conference Accounts Creation
Let's review the following scenario:
• we need to create the subscriber;
• when we create the subscriber we need to create three conference accounts
  (conference users) - the first for moderator, the second for participant,
  and the third for listener.
*/
public void Sample_ManageSubscriber1()
{
    // Declare constants
    const int MODE_HOST = 1;           // Moderator
    const int MODE_PARTICIPANT = 2;
    const int MODE_LISTENER = 3;
    const String DNIS = "12";          // We use this DNIS number for sample purposes,
                                        // please use your DNIS number here

    // Declare variables
    Subscriber newSubscriber;
    Subscriber createdSubscriber;
    Confuser moderatorConfuser;
    Confuser participantConfuser;
    Confuser listenerConfuser;
    DNIS[] dnises;
    long dnisId;
    String generatedAccessCode;

    try
    {
        mLastError = "";

        // Create new instance of Subscriber object (to populate new subscriber fields)
        newSubscriber = new Subscriber();

        // Define all mandatory fields and some optional fields
        newSubscriber.pin = "jkraft";
        newSubscriber.password = "321";
        newSubscriber.city = "New-York";
        newSubscriber.country = "US";
        newSubscriber.email = "jkraft@phone-mobile.com";
        newSubscriber.firstName = "Julie";
        newSubscriber.lastName = "Kraft";
        newSubscriber.phoneNumber = "(204) 221-7600";
        // For instance, we do not want to define additional optional properties,
        // such as newOperatorSubscriber.address1, newOperatorSubscriber.address2, etc.

        // Find DNIS 12 (SPECTEL)
        // Note. In this sample we create sample for DNIS 12 (SPECTEL),
        // you can use your DNIS to create your conference accounts
        dnises = ws.getDNISes(0, 0, "did=" + DNIS, null);
        // XML that was sent to the server see here:
        // Sample\_ManageSubscriber1.getDNISes.12\_sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageSubscriber1.getDNISes.12\_received.xml
        if (dnises != null && dnises.Length > 0)
        {
            // Create conference users only if the requested DNIS was found
            dnisId = dnises[0].dnisId; // The ID of DNIS

            // Create new instance and populate Confuser object for the moderator role
            generatedAccessCode = ws.generateAccessCode(); // Generate access code
            // XML that was sent to the server see here:
            // Sample\_ManageSubscriber1.generateAccessCode.sent.xml

```

```
// XML that was received from the server see here:
// Sample ManageSubscriber1.generateAccessCode.received.xml

/*
 * Programmers notes.
 * When we create new conference users (either using createConfuser or
 * createSubscriber methods) if conferenceInfo.conferenceNumber == 0 and
 * conferenceInfo.description != null, the new ConfInfo object will be created,
 * new unique 6-digits conference number will be
 * assigned to this ConfInfo object. The created object can be used in new
 * conference users creation.
 * If when we create the subscriber only one confuser has not null
 * conferenceInfo, one new conference number (conference info) will be created
 * and all other conference users (where conferenceInfo is null) will be created
 * and assigned to this conference info.
 * In the sample below we define conferenceInfo for the moderator confuser only;
 * because we do not define conferenceInfo for the participant and the listener
 * confuser they will be assigned to the same conference number (conferenceInfo)
 * that will be created for the moderator.
 */
moderatorConfuser = new Confuser();
moderatorConfuser.accessCode = generatedAccessCode;
moderatorConfuser.dnisId = dnisId;
moderatorConfuser.conferenceInfo = new ConfInfo();
moderatorConfuser.conferenceInfo.description = "MMC_JKRAFT";
moderatorConfuser.dnisIdSpecified = true;
moderatorConfuser.role = MODE_HOST;
moderatorConfuser.roleSpecified = true;
/*
 * Programmers notes.
 * If you are coding on C# or VB.Net in some cases client web services proxy
 * code can generate additional parameter <property>Specified (Boolean type).
 * This behavior is by design. The issue is with value types that are marked in
 * the WSDL as not being required. Since they are value types, they can't
 * return. The solution that Microsoft implemented was to add a separate Boolean
 * field or property you can set to say whether or not you are supplying the
 * value.
 * This means that when your .NET application wants to call web service, it needs
 * to set the <property>Specified property. This property is not included into XML
 * that will be sent to server, but it is used to generate this XML.
 * dnisIdSpecified, roleSpecified - are samples of such properties.
 */

// Create new instance and populate Confuser object for the participant role
generatedAccessCode = ws.generateAccessCode();

participantConfuser = new Confuser();
participantConfuser.accessCode = generatedAccessCode;
participantConfuser.dnisId = dnisId;
participantConfuser.dnisIdSpecified = true;
participantConfuser.role = MODE_PARTICIPANT;
participantConfuser.roleSpecified = true;

// Create new instance and populate Confuser object for the listener role
generatedAccessCode = ws.generateAccessCode();

listenerConfuser = new Confuser();
listenerConfuser.accessCode = generatedAccessCode;
listenerConfuser.dnisId = dnisId;
listenerConfuser.dnisIdSpecified = true;
listenerConfuser.role = MODE_LISTENER;
listenerConfuser.roleSpecified = true;

// Add moderator and participant conference users to new subscribers
// that should be created
newSubscriber.confusers = new Confuser[3];
newSubscriber.confusers.SetValue(moderatorConfuser, 0);
newSubscriber.confusers.SetValue(participantConfuser, 1);
newSubscriber.confusers.SetValue(listenerConfuser, 2);
}

```

```
// Call web service method createSubscriber (to create new subscriber)
// and his conference accounts
createdSubscriber = ws.createSubscriber(newSubscriber);
// XML that was sent to the server see here:
// Sample ManageSubscriber1.createSubscriber.sent.xml
// XML that was received from the server see here:
// Sample ManageSubscriber1.createSubscriber.received.xml
// Screenshot of new created subscriber see here:
// Sample ManageSubscriber1.createSubscriber.jpg

    return;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_ManageSubscriber1: " + ex.Message;
}
}
```

Sample of Subscribers Filtering, Modifications, Conference Accounts Modifications (Sample_ManageSubscriber2)

```

/*
Sample of Subscribers Filtering, Modifications, Conference Accounts Modifications
Let's review the following scenario:
• we need to find the subscriber that was created in the previous sample using his pin;
• for the selected subscriber we need to modify his password and email;
• for the selected subscriber we need to remove his conference accounts (conference users)
  with the listener role;
• for the selected subscriber we need to define some custom attributes as well as change
  access code for his conference accounts with host role.
*/
public void Sample_ManageSubscriber2()
{
    // Declare constants
    const int MODE_HOST = 1;           // Moderator
    const int MODE_PARTICIPANT = 2;
    const int MODE_LISTENER = 3;
    // Declare variables
    Subscriber[] listSubscribers;
    Subscriber userSubscriber;
    Confuser currentConfuser;
    Confuser[] moderatorConfusers;
    ConfInfo currentConfInfo = null;
    int confusersCount;
    String generatedAccessCode;

    try
    {
        mLastError = "";

        // Find jkraft subscriber (created in previous sample)
        listSubscribers = ws.getSubscribers(0, 0, "pin='jkraft'", "");
        // XML that was sent to the server see here:
        // Sample\_ManageSubscriber2.getSubscribers.pin sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageSubscriber2.getSubscribers.pin received.xml
        /*
        List<Subscriber> getSubscribers(long offset,
                                     long limit,
                                     String filter,
                                     String order)
                                     throws ServerException,
                                     AccessDeniedException
        * This function returns list of Subscribers that match filter.
        * Offset and limit allow to implement paging on the web server.
        * Please note that field confusers in Subscriber will not be populated to avoid huge
        * amount of data to be transferred in case if big request is processed.
        * Parameters:
        *   offset - zero based offset in recordset.
        *   limit - maximum number of objects to return.
        *   filter - The criteria to use to filter the rows. The criteria should be a simple sql
        *             conditional statement started with one or more Subscriber field names.
        *             Acceptable operators: <= , >= , != , = , < , > , like , ilike, and
        *             For example login='12' or login like '%2%' or subscriberId >= 15.
        *             Empty string or null means no filter.
        *   order - A string specifying Subscriber field name and sort direction.
        *             For example "login" or "email desc". The default direction
        *             is asc and can be omitted. Empty string or null means no order.
        * Acceptable fields:
        *   •subscriberId
        *   •parentId
        *   •pin
        *   •password
        *   •firstName
        *   •lastName
        *   •email
        */

```



```

        •address1
        •city
        •country
        •phoneNumber
    * Returns:
      list of Subscriber objects
    */
    if (listSubscribers != null && listSubscribers.Length > 0)
    {
        /*
         * Programmers notes.
         * Because getSubscribers method returns only the list of subscribers with their basic
         * attributes and does not return conferenceInfo attributes property, we need to call
         * getSubscriber method for the subscriber that was found to get his complete set of
         * attributes
         */
        userSubscriber = ws.getSubscriber(listSubscribers[0].subscriberId);
        // XML that was sent to the server see here:
        // Sample ManageSubscriber2.getSubscriber.sent.xml
        // XML that was received from the server see here:
        // Sample ManageSubscriber2.getSubscriber.received.xml

        userSubscriber.password = "654321";
        userSubscriber.email = "jkraft@manage.com";

        if (userSubscriber.confusers != null)
        {
            confusersCount = 0;
            for (int idx = 0; idx < userSubscriber.confusers.Length; idx++)
            {
                currentConfuser = userSubscriber.confusers[idx];

                if (currentConfuser.role == MODE_HOST)
                {
                    generatedAccessCode = ws.generateAccessCode(); // Generate new access code
                    currentConfuser.accessCode = generatedAccessCode;
                    currentConfInfo = currentConfuser.conferenceInfo;
                    confusersCount++;
                }
                else if (currentConfuser.role == MODE_PARTICIPANT)
                {
                    confusersCount++;
                }
                else if (currentConfuser.role == MODE_LISTENER)
                {
                    userSubscriber.confusers[idx] = null;
                }
            }
            moderatorConfusers = new Confuser[confusersCount];
            confusersCount = 0;
            for (int idx = 0; idx < userSubscriber.confusers.Length; idx++)
            {
                if (userSubscriber.confusers[idx] != null)
                {
                    moderatorConfusers.SetValue(userSubscriber.confusers[idx], confusersCount);
                    confusersCount++;
                }
            }
            userSubscriber.confusers = moderatorConfusers;
        }
        // Call web service method updateSubscriber (to modify existing subscriber)
        ws.updateSubscriber(userSubscriber);
        // XML that was sent to the server see here:
        // Sample ManageSubscriber2.updateSubscriber.sent.xml
        // XML that was received from the server see here:
        // Sample ManageSubscriber2.updateSubscriber.received.xml
    }
}

```

```

// Define custom attributes for subscriber's conference info
if (currentConfInfo != null)
{
    foreach (jAdmin.Attribute attr in currentConfInfo.attributes)
    {
        if (attr.name == "conference_entrytones")
        {
            attr.value = "off";
            attr.isOverridden = true;
            attr.isOverriddenSpecified = true;
        }
        if (attr.name == "conference_exittones")
        {
            attr.value = "off";
            attr.isOverridden = true;
            attr.isOverriddenSpecified = true;
        }
        if (attr.name == "conference_start_wait")
        {
            attr.value = "500";
            attr.isOverridden = true;
            attr.isOverriddenSpecified = true;
        }
    }
    // Call web service method updateSubscriber (to modify existing subscriber)
    ws.updateConferenceInfo(currentConfInfo);
    // XML that was sent to the server see here:
    // Sample ManageSubscriber2.updateConferenceInfo.sent.xml
    // XML that was received from the server see here:
    // Sample ManageSubscriber2.updateConferenceInfo.received.xml
}

// Screenshot of updated subscriber see here:
// Sample ManageSubscriber2.updateSubscriber.jpg
// Screenshot of updated subscriber's conference account see here:
// Sample ManageSubscriber2.updateSubscriber confuser.jpg
}

return;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_ManageSubscriber2: " + ex.Message;
}
}

```

Sample of Subscribers Filtering and Deletion (Sample_ManageSubscriber3)

```

/*
Sample of Subscribers Filtering and Deletion
Let's review the following scenario:
• we need to find out all subscribers who have emails from domain "manage.com";
• for each of these subscribers if the subscriber does not have phone number
  we need to delete him.
*/
public void Sample_ManageSubscriber3()
{
    // Declare variables
    Subscriber[] listSubscribers;

    try
    {
        mLastError = "";

        // Find all subscribers who have emails from domain "manage.com"
        listSubscribers = ws.getSubscribers(0, 0, "email like'%@manage.com'", "");
        // XML that was sent to the server see here:
        // Sample\_ManageSubscriber3.getSubscribers.email\_sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageSubscriber3.getSubscribers.email\_received.xml
        // See screenshot of the subscribers filtered by email
        // that were on the bridge prior to the program start:
        // Sample\_ManageSubscriber3.subscribers before.jpg

        if (listSubscribers != null)
        {
            foreach (Subscriber s in listSubscribers)
            {
                if (String.IsNullOrEmpty(s.phoneNumber))
                {
                    // Delete the subscriber
                    ws.deleteSubscriber(s.subscriberId);
                    // XML that was sent to the server see here:
                    // Sample\_ManageSubscriber3.deleteSubscriber.sent.xml
                    // XML that was received from the server see here:
                    // Sample\_ManageSubscriber3.deleteSubscriber.received.xml
                }
            }
        }
        // See screenshot of the subscribers filtered by email
        // that were on the bridge after the program is finished:
        // Sample\_ManageSubscriber3.subscribers after.jpg

        return;
    }
    catch (Exception ex)
    {
        mLastError = "Error in " + this.GetType().FullName +
            ".Sample_ManageSubscriber3: " + ex.Message;
    }
}

```

Sample of Getting Conference Users Information (Sample_ManageConfuser1)

```

/*
Sample of Getting Conference Users Information
Let's review the following scenario:
• we need to count conference users (accounts) with for SPECTEL call flow;
• we need to get all conference users (accounts) with for SPECTEL call flow
  with host role;
• we need to output subscriber ID, conference number, access code for them.
*/
public String Sample_ManageConfuser1()
{
    // Declare constants
    const int MODE_HOST = 1;          // Moderator
    // Declare variables
    long lngConfusersCount;
    CallFlow[] callFlows;
    DNIS[] dnises;
    Confuser[] confusers;
    String dnisIDs = ",";
    String strInfo;

    try
    {
        mLastError = "";
        strInfo = "";

        // Get requested call flow by name
        callFlows = ws.getCallFlows(0, 0, "name='SPECTEL'", "");
        // XML that was sent to the server see here:
        // Sample\_ManageConfuser1.getCallFlows.sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageConfuser1.getCallFlows.received.xml
        /*
        List<CallFlow> getCallFlows(long offset,
                                   long limit,
                                   java.lang.String filter,
                                   java.lang.String order)
                                   throws ServerException,
                                   AccessDeniedException
        * This function returns list of CallFlows which match the filter provided.
        * There are two parameters offset and limit to help to implement paging on the web
        * application. All users can get all CallFlows registered on the bridge. Later there
        * will be introduced a restriction so users are able to see only those CallFlows which
        * are assigned to them.
        * Parameters:
        *   offset - zero based offset in recordset.
        *   limit - maximum number of objects to return.
        *   filter - The criteria to use to filter the rows. The criteria should be a simple sql
        *             conditional statement started with one or more CallFlow field names.
        *             Acceptable operators: <= , >= , != , = , < , > , like , ilike, and
        *             For example name='12' or name like '%2%' or collFlowId >= 15.
        *             Empty string or null means no filter.
        *   order - A string specifying CallFlow field name and sort direction.
        *             For example "name" or "name desc". The default direction is asc and can be omitted.
        *             Empty string or null means no order.
        * Accepted fields:
        *   •callFlowId
        *   •name
        *   •path
        * Returns:
        *   list of CallFlow objects
        */
        if (callFlows != null && callFlows.Length > 0)
        {
            // Get DNISes for the selected call flow
            dnises = ws.getDNISes(0, 0, "callFlowId=" + callFlows[0].callFlowId.ToString(), "");
            // XML that was sent to the server see here:

```

```
// Sample ManageConfuser1.getDNISes.sent.xml
// XML that was received from the server see here:
// Sample ManageConfuser1.getDNISes.received.xml
/*
List<DNIS> getDNISes(long offset,
                    long limit,
                    java.lang.String filter,
                    java.lang.String order)
                    throws ServerException,
                        AccessDeniedException
* This function returns list of DNISes (phone numbers) which match the filter
* provided. There are two parameters offset and limit to help to implement paging on
* the web application. All users can get all numbers registered on the bridge.
* Parameters:
    offset - zero based offset in recordset.
    limit - maximum number of objects to return.
    filter - The criteria to use to filter the rows. The criteria should be a simple
            sql conditional statement started with one or more DNIS field names.
    Acceptable operators: <= , >= , != , = , < , > , like , ilike, and
    For example name='12' or name like'%2%' or collFlowId >= 15.
    Empty string or null means no filter.
    order - A string specifying DNIS field name and sort direction.
            For example "name" or "name desc".
            The default direction is asc and can be omitted.
            Empty string or null means no order.
* Accepted fields:
    •callFlowId
    •dnisId
    •did
    •description
* Returns:
    list of DNIS objects
*/
if (dnises != null && dnises.Length > 0)
{
    foreach (DNIS d in dnises)
    {
        if (dnisIDs.IndexOf(", " + d.dnisId.ToString() + ",") < 0)
        {
            dnisIDs += d.dnisId.ToString() + ",";
        }
    }

    if (dnisIDs.Length <= 2)
    {
        dnisIDs = "";
    }
    else
    {
        if (Utils.LeftString(dnisIDs, 1) == ",")
            dnisIDs = dnisIDs.Substring(1);
        if (Utils.RightString(dnisIDs, 1) == ",")
            dnisIDs = Utils.LeftString(dnisIDs, dnisIDs.Length - 1);
    }

    if (!String.IsNullOrEmpty(dnisIDs))
    {
        // Count how many conference users exist on the bridge for the call flow SPECTEL
        lngConfusersCount = ws.getConfusersCount("dnisId in (" + dnisIDs + ")");
        // XML that was sent to the server see here:
        // Sample ManageConfuser1.getConfusersCount.sent.xml
        // XML that was received from the server see here:
        // Sample ManageConfuser1.getConfusersCount.received.xml
        strInfo += "Number of SPECTEL conference users: " + lngConfusersCount.ToString()
            + ". \n\r";

        // Get conference users for the selected call flow
        confusers = ws.getConfusers(0, 0, "dnisId in (" + dnisIDs + ") and role = "
            + MODE_HOST.ToString(), "subscriberId asc");
        // XML that was sent to the server see here:
        // Sample ManageConfuser1.getConfusers.sent.xml

```

```

// XML that was received from the server see here:
// Sample ManageConfuser1.getConfusers.received.xml
/*
List<Confuser> getConfusers(long offset,
                           long limit,
                           java.lang.String filter,
                           java.lang.String order)
                           throws ServerException,
                           AccessDeniedException
* This function returns the list of Confuser which match the given filter.
* There are rare cases when this function needs to be called directly as
* getSubscriber returns list of subordinate conference users.
* Parameters:
  offset - zero based offset in recordset.
  limit - maximum number of objects to return.
  filter - The criteria to use to filter the rows. The criteria should be a
          simple sql conditional statement started with one or more Confuser
          field names.
  Acceptable operators: <= , >= , != , = , < , > , like , ilike, and
  For example login='12' or login like '%2%' or subscriberId >= 15.
  Empty string or null means no filter.
  order - A string specifying Confuser field name and sort direction.
  For example "name" or "name desc".
  The default direction is asc and can be omitted.
  Empty string or null means no order.
* Accepted fields:
  •subscriberId
  •confuserId
  •role
  •dnisId
  •accessCode
  •conferenceNumber
* Returns:
  list of DNIS objects
*/
if (confusers != null && confusers.Length > 0)
{
    strInfo += "Number of SPECTEL conference users with host role: "
              + confusers.Length.ToString() + ". \n\r";
    strInfo += "subscr.\tconf #\taccess code \n\r";
    foreach (Confuser cu in confusers)
    {
        strInfo += cu.subscriberId.ToString() + "\t"
                  + cu.conferenceInfo.conferenceNumber.ToString() + "\t"
                  + cu.accessCode + "\n\r";
    }
}
else
{
    strInfo += "No SPECTEL conference users with host role found. \n\r";
}
}
}

// Sample of program output: Sample ManageConfuser1.return.jpg
// ** Number of SPECTEL conference users: 4.
// ** Number of SPECTEL conference users with host role: 2.
// ** subscr. conf #      access code
// ** 3      758288      961091
// ** 4      214423      870888

return strInfo;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
                ".Sample_ManageConfuser1: " + ex.Message;
    return mLastError;
}
}

```

Conferences and Calls Management

Sample of Conferences Filtering, Changes Secure Mode, Dropping the Conferences (Sample_ManageConference1)

```

/*
Sample of Conferences Filtering, Changes Secure Mode, Dropping the Conferences
Let's review the following scenario:
• we need to count how many conferences are currently on the bridge;
• for the selected subscriber we need to drop all conferences if the participants count
  less than two;
• for unsecured conferences for the selected subscriber with two participants we need to
  make them secure.
*/
public String Sample_ManageConference1()
{
    // Declare variables
    long lngConferencesCount;
    Conference[] singleParticipantConferences;
    Conference[] twoParticipantsUnsecuredConferences;
    String filterSubscriber;
    String strStatus;

    try
    {
        mLastError = "";
        strStatus = "";

        // See screenshot of the conferences that were started on the bridge prior
        // to the program start: conferences before.jpg

        // Count started conferences
        // We use empty filter parameter to output all conferences
        lngConferencesCount = ws.getConferencesCount("");
        // XML that was sent to the server see here:
        // Sample\_ManageConference1.getConferencesCount.all sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageConference1.getConferencesCount.all received.xml
        strStatus += "Number of started conferences: " + lngConferencesCount.ToString()
            + ". \n\r";

        // Find all subscriber's conferences with the participants count less than two
        filterSubscriber = GetConferenceNumbersBySubscriberPIN("admin");
        // Click here to see GetConferenceNumbersBySubscriberPIN function implementation
        if (!String.IsNullOrEmpty(filterSubscriber))
            filterSubscriber = "conferenceNumber in (" + filterSubscriber + ") and ";
        singleParticipantConferences = ws.getConferences(0, 0,
            filterSubscriber + "participantCnt<2", "");
        // XML that was sent to the server see here:
        // Sample\_ManageConference1.getConferences.single sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageConference1.getConferences.single received.xml
    }
    /*
    List<Conference> getConferences(long offset,
        long limit,
        java.lang.String filter,
        java.lang.String order)
        throws ServerException,
        AccessDeniedException
    * This function returns list of Conferences which are registered for the subscriber
    * on which behalf this call is executed.
    * For administrator it returns list of all registered Conferences.
    * Parameters:
    * offset - zero based offset in recordset.
    * limit - maximum number of objects to return.
    */
}

```

```

    filter - The criteria to use to filter the rows.
           The criteria should be a simple sql conditional statement started with one or
           more Conference field names.
           Acceptable operators: <= , >= , != , = , < , > , like , ilike , and
           For example conferenceNumber='12' or conferenceNumber like '%2%' or duration >= 15.
    order - A string specifying Conference field name and sort direction.
           For example "conferenceNumber" or "created desc".
           The default direction is asc and can be omitted.
           Empty string or null means no order.
* Accepted fields:
  •conferenceId
  •conferenceNumber
  •created ('yyyy.MM.dd/hh:mm' format)
  •duration
  •participantCnt
  •isSecured
  •muteMode
* Empty string or null means no filter.
* Returns:
  list of Conference objects
*/
if (singleParticipantConferences != null && singleParticipantConferences.Length > 0)
{
    foreach (Conference c in singleParticipantConferences)
    {
        ws.hangupConference(c.conferenceId);
        // XML that was sent to the server see here:
        // Sample ManageConference1.hangupConference.sent.xml
        // XML that was received from the server see here:
        // Sample ManageConference1.hangupConference.received.xml
    }
    strStatus += "Number of dropped single participant conferences: "
                + singleParticipantConferences.Length.ToString() + ". \n\r";
}
else
{
    strStatus += "No single participant conferences found. \n\r";
}

// Find subscriber's unsecured conferences with two participants
twoParticipantsUnsecuredConferences = ws.getConferences(0, 0,
    filterSubscriber + "isSecured=0 and participantCnt=2", "");
// XML that was sent to the server see here:
// Sample ManageConference1.getConferences.two\_sent.xml
// XML that was received from the server see here:
// Sample ManageConference1.getConferences.two\_received.xml
if (twoParticipantsUnsecuredConferences != null &&
    twoParticipantsUnsecuredConferences.Length > 0)
{
    foreach (Conference c in twoParticipantsUnsecuredConferences)
    {
        ws.secureConference(c.conferenceId);
        // XML that was sent to the server see here:
        // Sample ManageConference1.secureConference.sent.xml
        // XML that was received from the server see here:
        // Sample ManageConference1.secureConference.received.xml
    }
    strStatus += "Number of two participants conferences made secured: "
                + twoParticipantsUnsecuredConferences.Length.ToString() + ". \n\r";
}
else
{
    strStatus += "No unsecured conferences with two participants found. \n\r";
}

```



```
// See screenshot of the conferences that were on the bridge after the program
// is finished: conferences\_after.jpg
// In this case the program returns the following message:
// Sample\_ManageConference1.return.jpg
// ** Number of started conferences: 2.
// ** Number of dropped single participant conferences: 1.
// ** Number of two participants conferences made secured: 1.

    return strStatus;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
                ".Sample_ManageConference1: " + ex.Message;
    return mLastError;
}
}
```

Sample of Placing the Entire Conference on Hold, Starting and Stopping Q&A Sessions and Conference Recording (Sample_ManageConference2)

```

/*
Sample of Placing the Entire Conference on Hold, Starting and Stopping Q&A Sessions and
Conference Recording
Let's review the following scenario:
• we need to place the specific conference (the conference with specific conference
  number) on hold;
• we need to wait 1 minute and take this conference off hold;
• after that we need to start conference recording and start Q&A session for this
  conference;
• we need to wait 1 minute, we assume that conference participants requested to ask
  questions during this minute;
• we need to let the first participant ask his question (i.e. un-mute him - engage his
  Q&A session);
• we need to wait 1 minute and then complete the first participant question, i.e.
  disengage his Q&A session;
• we need to stop Q&A session and stop conference recording for this conference.
*/
public void Sample_ManageConference2()
{
    // Declare constants
    const int QA_MODE_OPEN = 0;                // Stop Q&A mode for the conference
    const int QA_MODE_CLOSED = 2;             // Start Q&A mode for the conference
    const long CONFERENCE_NUMBER = 667788;    // Default conference number for this sample
    // Declare variables
    Conference[] conferences;
    Session[] sessions;
    long conferenceId;
    long sessionId;

    try
    {
        mLastError = "";

        // See screenshot of the conferences that were started on the bridge prior
        // to the program start: conferences\_before.jpg
        // See screenshot of the selected conference calls that were started on the bridge prior
        // to the program start: calls\_before.jpg

        // Find the conference with the the conference number 667788
        conferences = ws.getConferences(0, 0, "conferenceNumber="
            + CONFERENCE_NUMBER.ToString(), "");
        // XML that was sent to the server see here:
        // Sample\_ManageConference2.getConferences.conferenceNumber\_sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageConference2.getConferences.conferenceNumber\_received.xml

        if (conferences != null && conferences.Length > 0)
        {
            conferenceId = conferences[0].conferenceId;

            // Place the conference on hold
            ws.holdConference(conferenceId);
            // XML that was sent to the server see here:
            // Sample\_ManageConference2.holdConference.sent.xml
            // XML that was received from the server see here:
            // Sample\_ManageConference2.holdConference.received.xml

            // Wait 1 minute (60,000 milliseconds)
            System.Threading.Thread.Sleep(60000);

            // The conference is on hold.
            // See screenshot of the conferences that were on the bridge at this
            // moment: conferences\_pausel.jpg
            // See screenshot of the selected conference calls that were on the bridge at this
            // moment: calls\_pausel.jpg

```

```

// Take the conference off hold
ws.unHoldConference(conferenceId);
// XML that was sent to the server see here:
// Sample ManageConference2.unHoldConference.sent.xml
// XML that was received from the server see here:
// Sample ManageConference2.unHoldConference.received.xml

// Start the conference recording
ws.startConferenceRecording(conferenceId, "", "");
// XML that was sent to the server see here:
// Sample ManageConference2.startConferenceRecording.sent.xml
// XML that was received from the server see here:
// Sample ManageConference2.startConferenceRecording.received.xml

// Start Q&A session for the conference
//ws.muteConference(conferenceId, MUTE_MODE_QUESTION); // version 1.4
ws.qaSetMode(conferenceId, QA_MODE_CLOSED); // version 2.x
// XML that was sent to the server see here:
// Sample ManageConference2.qaSetMode.closed sent.xml
// XML that was received from the server see here:
// Sample ManageConference2.qaSetMode.closed received.xml

// Wait 1 minute (60,000 milliseconds)
System.Threading.Thread.Sleep(60000);

sessions = ws.getSessions(0, 0, "role=2 and conferenceId=" + conferenceId.ToString(), "");
// XML that was sent to the server see here:
// Sample ManageConference2.getSessions.conferenceId sent.xml
// XML that was received from the server see here:
// Sample ManageConference2.getSessions.conferenceId received.xml

if (sessions != null && sessions.Length > 0)
{
    sessionId = sessions[0].sessionId;

    // Engage Q&A session for the first conference participant
    ws.qaEngage(sessionId);
    // XML that was sent to the server see here:
    // Sample ManageConference2.qaEngage.sent.xml
    // XML that was received from the server see here:
    // Sample ManageConference2.qaEngage.received.xml
}
else
{
    sessionId = 0;
}

// Wait 1 minute (60,000 milliseconds)
System.Threading.Thread.Sleep(60000);

// The conference recording is started, the Q&A session is started,
// the first participant is asking a question.
// See screenshot of the conferences that were on the bridge at this
// moment: conferences pause2.jpg
// See screenshot of the selected conference calls that were on the bridge at this
// moment: calls pause2.jpg

if (sessionId > 0)
{
    // Disengage Q&A session for the first conference participant
    ws.qaDisengage(sessionId);
    // XML that was sent to the server see here:
    // Sample ManageConference2.qaDisengage.sent.xml
    // XML that was received from the server see here:
    // Sample ManageConference2.qaDisengage.received.xml
    // See screenshot of the selected conference calls that were on the bridge at this
    // moment: calls point3.jpg
}

```

```

// Stop Q&A session for the conference
//ws.muteConference(conferenceId, MUTE_MODE_OPEN);           // version 1.4
ws.qaSetMode(conferenceId, QA_MODE_OPEN);                   // version 2.x
// XML that was sent to the server see here:
// Sample ManageConference2.qaSetMode.open sent.xml
// XML that was received from the server see here:
// Sample ManageConference2.qaSetMode.open received.xml
// See screenshot of the selected conference calls that were on the bridge at this
// moment: calls point4.jpg

// Stop the conference recording
ws.stopConferenceRecording(conferenceId);
// XML that was sent to the server see here:
// Sample ManageConference2.stopConferenceRecording.sent.xml
// XML that was received from the server see here:
// Sample ManageConference2.stopConferenceRecording.received.xml
}

// See screenshot of the conferences that were on the bridge after
// the program is finished: conferences after.jpg
// See screenshot of the selected conference calls that were on the bridge after
// the program is finished: calls after.jpg

    return;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
                ".Sample_ManageConference2: " + ex.Message;
}
}

```

Sample of Conference Polling Sessions (Sample_ManageConference3)

```

/*
Sample of Conference Polling Sessions
Let's review the following scenario:
• we need to start the polling session for the specific conference (the conference
  with specific conference number) with available polling options 1, 2, 3;
• we need to wait 1 minute, we assume that conference participants will vote
  (select one of the available options) during this minute;
• we need to stop the polling session for this conference;
• after that we need to output polling results.
*/
public String Sample_ManageConference3()
{
    // Declare constants
    const long CONFERENCE_NUMBER = 651077;    // Default conference number for this sample
    const String POLLING_OPTIONS = "123";    // Available polling options
    // Declare variables
    Conference[] conferences;
    PollingResult[] pollingResults;
    long conferenceId;
    String strStatus;

    try
    {
        mLastError = "";
        strStatus = "";

        conferences = ws.getConferences(0, 0, "conferenceNumber="
            + CONFERENCE_NUMBER.ToString(), "");
        // XML that was sent to the server see here:
        // Sample\_ManageConference3.getConferences.conferenceNumber sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageConference3.getConferences.conferenceNumber received.xml
        if (conferences != null && conferences.Length > 0)
        {
            conferenceId = conferences[0].conferenceId;

            // Conference calls before the polling session has been started:
            // Sample\_ManageConference3.conference before.jpg
            ws.startPolling(conferenceId, POLLING_OPTIONS);
            // XML that was sent to the server see here:
            // Sample\_ManageConference3.startPolling.sent.xml
            // XML that was received from the server see here:
            // Sample\_ManageConference3.startPolling.received.xml
            // Conference calls after the polling session has been started:
            // Sample\_ManageConference3.conference after.jpg

            // Wait 1 minute (60,000 milliseconds)
            System.Threading.Thread.Sleep(60000);

            ws.stopPolling(conferenceId);
            // XML that was sent to the server see here:
            // Sample\_ManageConference3.stopPolling.sent.xml
            // XML that was received from the server see here:
            // Sample\_ManageConference3.stopPolling.received.xml

            pollingResults = ws.getPollingResults(conferenceId);
            // XML that was sent to the server see here:
            // Sample\_ManageConference3.getPollingResults.sent.xml
            // XML that was received from the server see here:
            // Sample\_ManageConference3.getPollingResults.received.xml

            if (pollingResults != null && pollingResults.Length > 0)
            {
                strStatus += "Polling results for the conference " + CONFERENCE_NUMBER.ToString()
                    + ". \n\r";
            }
        }
    }
}

```

```

        foreach (PollingResult pr in pollingResults)
        {
            strStatus += pr.created.ToShortDateString() + " "
                + pr.created.ToShortTimeString() + "\n\r";
            foreach (anyType2anyTypeMapEntry s in pr.votes)
            {
                strStatus += "key: " + s.key + " / value: " + s.value + "\n\r";
            }
        }
    }
    else
    {
        strStatus += "No polling results for the conference. \n\r";
    }
}
else
{
    strStatus += "The conference not found. \n\r";
}

// Sample of program output: Sample ManageConference3.return.jpg
// ** Polling results for the conference 651077.
// ** 21.12.2009 14:18
// ** key: 1 / value: 2
// ** key: 2 / value: 0
// ** key: 3 / value: 1
// Polling charts: Sample ManageConference3.conference pollingCharts.jpg

    return strStatus;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_ManageConference3: " + ex.Message;
    return mLastError;
}
}

```

**Sample of Calls Filtering, Mute the Calls, Dropping the Calls
(Sample_ManageCall1)**

```

/*
Sample of Calls Filtering, Mute the Calls, Dropping the Calls
Let's review the following scenario:
• we need to count how many calls are currently on the bridge;
• for the selected subscriber we need to drop all participants calls if the call duration
  greater than 10 minutes;
• for remaining participants of the selected subscriber (with call duration less than 10
  minutes) we need to mute their calls.
*/
public String Sample_ManageCall1()
{
    // Declare variables
    long lngSessionsCount;
    long lngDroppedCount;
    long lngMutedCount;
    Session[] participantsSessions;
    String filterSubscriber;
    String strStatus;

    try
    {
        mLastError = "";
        strStatus = "";

        // See screenshot of the calls that were started on the bridge prior to the program
        // start: calls before.jpg

        // Count started calls
        // We use negative conferenceId parameter and empty filter parameter to output all calls
        lngSessionsCount = ws.getSessionsCount("");
        // XML that was sent to the server see here:
        // Sample\_ManageCall1.getSessionsCount.all sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageCall1.getSessionsCount.all received.xml
        strStatus += "Number of started calls: " + lngSessionsCount.ToString() + ". \n\r";

        // Find all subscriber's calls (sessions) where the role is participant
        filterSubscriber = GetConferenceNumbersBySubscriberPIN("admin");
        // Click here to see GetConferenceNumbersBySubscriberPIN function implementation
        if (!String.IsNullOrEmpty(filterSubscriber))
            filterSubscriber = "conferenceNumber in (" + filterSubscriber + ") and ";
        participantsSessions = ws.getSessions(0, 0, filterSubscriber + "role=2", "");
        // XML that was sent to the server see here:
        // Sample\_ManageCall1.getSessions.participants sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageCall1.getSessions.participants received.xml
    }
    /*
    List<Session> getSessions(long offset,
        long limit,
        java.lang.String filter,
        java.lang.String order)
        throws ServerException,
            AccessDeniedException,
            ObjectNotFoundException

    * This function returns list of Sessions (calls) which match the filter provided.
    * There are two parameters offset and limit which help to implement paging on the web
    * application. If this function is called from non admin Subscribers it will returns
    * only Sessions visible for this account.
    * If call doesn't present an accesscode yet - it is visible only by admin
    * Parameters:
        conferenceId - Conference Identifier.
            If parameter is less than zero Session objects for all Conference will be returned.
        offset - zero based offset in recordset.
        limit - maximum number of objects to return.
        filter - The criteria to use to filter the rows. The criteria should be a simple sql

```

```

        conditional statement started with one or more Session field names.
        Acceptable operators: <= , >= , != , = , < , > , like , ilike , and
        For example addressTo='12' or addressTo like'%2%' or duration >= 15.
    order - A string specifying Session field name and sort direction.
        For example "caller" or "caller desc".
        The default direction is asc and can be omitted.
        Empty string or null means no order.
* Accepted fields:
    •sessionId
    •subscriberId
    •created ('yyyy.MM.dd/hh:mm' format)
    •joined ('yyyy.MM.dd/hh:mm' format) (works only when joined the conference)
    •duration
    •status
    •role (works only when joined the conference)
    •isMuted (works only when joined the conference) true/false values
    •addressTo
    •addressFrom
    •conferenceNumber (works only when joined the conference)
    •accessCode (works only when joined the conference)
* Empty string or null means no filter.
* Returns:
    list of Session objects
*/
if (participantsSessions != null && participantsSessions.Length > 0)
{
    lngDroppedCount = 0;
    lngMutedCount = 0;
    foreach (Session s in participantsSessions)
    {
        if (s.duration > 600)        // 600 seconds = 10 minutes
        {
            ws.hangupSession(s.sessionId);
            // XML that was sent to the server see here:
            // Sample ManageCall1.hangupSession.sent.xml
            // XML that was received from the server see here:
            // Sample ManageCall1.hangupSession.received.xml
            lngDroppedCount++;
        }
        else
        {
            ws.muteSession(s.sessionId);
            // XML that was sent to the server see here:
            // Sample ManageCall1.muteSession.sent.xml
            // XML that was received from the server see here:
            // Sample ManageCall1.muteSession.received.xml
            lngMutedCount++;
        }
    }
    strStatus += "Number of participants' calls: "
        + participantsSessions.Length.ToString() + ". \n\r";
    strStatus += "Number of dropped participants' calls: " + lngDroppedCount.ToString()
        + ". \n\r";
    strStatus += "Number of muted participants' calls: " + lngMutedCount.ToString()
        + ". \n\r";
}
else
{
    strStatus += "No participants' calls found. \n\r";
}

// See screenshot of the calls that were on the bridge after the program is finished:
// calls after.jpg
// In this case the program returns the following message: Sample ManageCall1.return.jpg
// ** Number of started calls: 3.
// ** Number of participants' calls: 2.
// ** Number of dropped participants' calls: 1.
// ** Number of muted participants' calls: 1.

return strStatus;

```



```
    }  
    catch (Exception ex)  
    {  
        mLastError = "Error in " + this.GetType().FullName +  
                     ".Sample_ManageCall1: " + ex.Message;  
        return mLastError;  
    }  
}
```

Sample of Setting Custom Name and Placing Calls on Hold (Sample_ManageCall2)

```

/*
Sample of Setting Custom Name and Placing Calls on Hold
Let's review the following scenario:
• for the conference with specific conference number we need to set custom name for the
  host "conference moderator";
• for the same conference we need to place all listeners and participants on hold.
*/
public void Sample_ManageCall2()
{
    // Declare constants
    const int MODE_HOST = 1;
    const int MODE_PARTICIPANT = 2;
    const int MODE_LISTENER = 3;
    const long CONFERENCE_NUMBER = 667788;    // Default conference number for testing
    // Declare variables
    Session[] conferenceSessions;

    try
    {
        mLastError = "";

        // See screenshot of the calls that were started on the bridge prior to
        // the program start: calls before.jpg
        // See screenshot of the conference calls that were started on the bridge prior to
        // the program start: conference before.jpg

        // Find all calls (sessions) for the conference number 667788
        conferenceSessions = ws.getSessions(0, 0, "conferenceNumber="
            + CONFERENCE_NUMBER.ToString(), "");
        // XML that was sent to the server see here:
        // Sample\_ManageCall2.getSessions.conferenceNumber\_sent.xml
        // XML that was received from the server see here:
        // Sample\_ManageCall2.getSessions.conferenceNumber\_received.xml

        if (conferenceSessions != null && conferenceSessions.Length > 0)
        {
            foreach (Session s in conferenceSessions)
            {
                if (s.role == MODE_HOST)
                {
                    ws.setCustomName(s.sessionId, "conference moderator");
                    // XML that was sent to the server see here:
                    // Sample\_ManageCall2.setCustomName.sent.xml
                    // XML that was received from the server see here:
                    // Sample\_ManageCall2.setCustomName.received.xml
                }
                else if (s.role == MODE_PARTICIPANT || s.role == MODE_LISTENER)
                {
                    ws.holdSession(s.sessionId);
                    // XML that was sent to the server see here:
                    // Sample\_ManageCall2.holdSession.sent.xml
                    // XML that was received from the server see here:
                    // Sample\_ManageCall2.holdSession.received.xml
                }
            }
        }

        // See screenshot of the calls that were on the bridge after
        // the program is finished: calls after.jpg
        // See screenshot of the conference calls that were on the bridge after
        // the program is finished: conference after.jpg

        return;
    }
}

```

```
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_ManageCall2: " + ex.Message;
}
}
```

CDRs Management

Sample of Getting Conferences Historical Information (Sample_InfoConferenceDR1)

```

/*
Sample of Getting Conferences Historical Information
Let's review the following scenario:
• we need to count how many conferences were on the bridge from the beginning of the
  month;
• for the selected subscriber we need to output his current month conferences information
  (conference number, conference ID, date and time when the conference occurred, duration,
  participants count, and info about recording URL if exists), ordered by conference
  number and conference date.
*/
public String Sample_InfoConferenceDR1()
{
    // Declare variables
    long lngConferencesCount;
    ConferenceDR[] conferenceDRs;
    DateTime startDate;
    String filter;
    String strInfo;

    try
    {
        mLastError = "";
        strInfo = "";

        // Count how many conferences were on the bridge from the beginning of the month
        startDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
        lngConferencesCount = ws.getConferenceDRsCount("created>='" +
            + Utils.Date2Sql(startDate) + "'");
        // XML that was sent to the server see here:
        // Sample\_InfoConferenceDR1.getConferenceDRsCount.all\_sent.xml
        // XML that was received from the server see here:
        // Sample\_InfoConferenceDR1.getConferenceDRsCount.all\_received.xml
        strInfo += "Number of current month conferences: " + lngConferencesCount.ToString()
            + ". \n\r";

        // Find all current month conferences for the subscriber
        filter = GetConferenceNumbersBySubscriberPIN("admin");
        // Click here to see GetConferenceNumbersBySubscriberPIN function implementation
        if (String.IsNullOrEmpty(filter))
            filter = "created>='" + Utils.Date2Sql(startDate) + "'";
        else
            filter = "created>='" + Utils.Date2Sql(startDate) + "' and conferenceNumber in ("
                + filter + ")";
        conferenceDRs = ws.getConferenceDRs(0, 0, filter, "conferenceNumber, created");
        // XML that was sent to the server see here:
        // Sample\_InfoConferenceDR1.getConferenceDRs.sent.xml
        // XML that was received from the server see here:
        // Sample\_InfoConferenceDR1.getConferenceDRs.received.xml
        /*
        List<ConferenceDR> getConferenceDRs(long offset,
            long limit,
            java.lang.String filter,
            java.lang.String order)
        * This function returns list of ConferenceDRs which are registered for the subscriber.
        * For administrator it returns whole list of records.
        * Parameters:
        *   offset - zero based offset in recordset.
        *   limit - maximum number of objects to return.
        *   filter - The criteria to use to filter the rows.
        *       The criteria should be a simple sql conditional statement started
        *       with one or more ConferenceDR field names.
        *       Acceptable operators: <= , >= , != , = , < , > , like , ilike, and

```

```

For example:
    conferenceId = 5424
    duration > 300 and duration < 400
    duration > 300 and conferenceNumber = 160
    participantCnt > 2 and participantCnt < 22
    created > '2010.08.07/00:00'
Empty string or null means no filter.
order - A string specifying ConferenceDR field name and sort direction.
    For example "conferenceNumber" or "created desc".
    The default direction is asc and can be omitted.
    Empty string or null means no order.
* Accepted fields:
    •conferenceId
    •conferenceNumber
    •created ('yyyy.MM.dd/hh:mm' format)
    •duration
    •participantCnt
* Returns:
    list (array) of ConferenceDR objects
*/
if (conferenceDRs != null && conferenceDRs.Length > 0)
{
    strInfo += "Number of current month conferences for the subscriber: "
        + conferenceDRs.Length.ToString() + ". \n\r";
    foreach (ConferenceDR cdr in conferenceDRs)
    {
        strInfo += cdr.conferenceNumber.ToString() + "\t"
            + cdr.conferenceId.ToString() + "\t"
            + cdr.created.ToShortDateString() + " " + cdr.created.ToShortTimeString() + "\t"
            + cdr.duration.ToString() + "\t"
            + cdr.participantCnt + "\t"
            + cdr.recordingUrl + "\n\r";
    }
}
else
{
    strInfo += "No current month conferences for the subscriber found. \n\r";
}

// Sample of program output: Sample InfoConferenceDR1.return.jpg
// ** Number of current month conferences: 7.
// ** Number of current month conferences for the subscriber: 6.
// ** 651077 6 15/03/2010 5:01 568 3
// ** 651077 7 15/03/2010 6:40 179 2
// ** 667788 3 15/03/2010 12:50 33 1
// ** 667788 4 15/03/2010 1:16 573 2
// ** 667788 5 15/03/2010 1:27 11824 4 conferences/788/667788/record/5.wav
// ** 667788 8 17/03/2010 12:32 1389 4

    return strInfo;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_InfoConferenceDR1: " + ex.Message;
    return mLastError;
}
}

```

Sample of the Shared Recording Generation (Sample_InfoConferenceDR2)

```

/*
Sample of the Shared Recording Generation
In the previous sample (Sample_InfoConferenceDR1) we get conferences with recording.
Let's review the following scenario:
• we need to generate recording URL link, that will allow user to download conference
  recording without authorization during the next hour (for the conference with recording
  referenced by the conferenceId, that was found in the previous sample);
• we need to output the ConferenceDR object information prior and after shared recording
  URL generation to see the differences in the object properties.
*/
public String Sample_InfoConferenceDR2(long conferenceId)
{
    // Declare constants
    const Boolean ALLOW_DOWNLOAD = true;
    // Declare variables
    ConferenceDR initialConferenceDR;
    ConferenceDR finalConferenceDR;
    DateTime expirePeriod;
    String strInfo;

    try
    {
        mLastError = "";
        strInfo = "";

        // Get initial the ConferenceDR object for the conference referenced by identifier
        initialConferenceDR = ws.getConferenceDR(conferenceId);
        // XML that was sent to the server see here:
        // Sample\_InfoConferenceDR2.getConferenceDR.initial sent.xml
        // XML that was received from the server see here:
        // Sample\_InfoConferenceDR2.getConferenceDR.initial received.xml

        if (initialConferenceDR != null)
        {
            // Calculate the period of time over which the shared link will be invalidated
            expirePeriod = DateTime.Now.AddHours(1);
            // Share the conference recording - generate URL to download
            ws.shareRecording(conferenceId, expirePeriod, ALLOW_DOWNLOAD);
            // XML that was sent to the server see here:
            // Sample\_InfoConferenceDR2.shareRecording.sent.xml
            // XML that was received from the server see here:
            // Sample\_InfoConferenceDR2.shareRecording.received.xml

            // Get final the ConferenceDR object for the conference referenced by identifier
            finalConferenceDR = ws.getConferenceDR(conferenceId);
            // XML that was sent to the server see here:
            // Sample\_InfoConferenceDR2.getConferenceDR.final sent.xml
            // XML that was received from the server see here:
            // Sample\_InfoConferenceDR2.getConferenceDR.final received.xml

            strInfo = "The conference " + conferenceId.ToString()
                + " recording can be download using URL: "
                + finalConferenceDR.sharedRecordingUrl + " till "
                + finalConferenceDR.expirePeriod.ToString() + ". \n\r";
        }
        else
        {
            strInfo = "The conference with ID " + conferenceId.ToString() + " not found. \n\r";
        }

        // Sample of program output: Sample\_InfoConferenceDR2.return.jpg
        // ** The conference 39744 recording can be download using URL:
        // ** conferences/-17-65-6716-42-97111-52-112-17-65-6712627-17-65-67188316-17-65-67.wav
        // ** till 19/03/2010 13:24:37.
    }
}

```

```
        return strInfo;
    }
    catch (Exception ex)
    {
        mLastError = "Error in " + this.GetType().FullName +
                    ".Sample_InfoConferenceDR2: " + ex.Message;
        return mLastError;
    }
}
```

Sample of Getting Calls Historical Information (Sample_InfoSessionDR1)

```

/*
Sample of Getting Calls Historical Information
Let's review the following scenario:


- we need to count how many calls were on the bridge from the beginning of the month for the specific conference number;
- for the specific conference number we need to output current month conference calls information (conference number, conference ID, date and time when the call occurred, duration, called number, calling number, custom name, disconnect reason);
- if number of calls to output greater than 5, we should implement paging and output 5 calls on the page.


*/
public String Sample_InfoSessionDR1()
{
    // Declare constants
    const long PAGE_SIZE = 5; // Page size to display portion of the SessionDR objects
    const long CONFERENCE_NUMBER = 667788; // The conference number to filter the SessionDR objects
    // Declare variables
    long lngSessionsCount;
    SessionDR[] sessionDRs;
    DateTime startDate;
    String filter;
    String strInfo;

    try
    {
        mLastError = "";
        strInfo = "";

        // Generate filter that should be user to retrieve SessionDR objects
        startDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
        filter = "created>=" + Utils.Date2Sql(startDate) + " and conferenceNumber="
            + CONFERENCE_NUMBER.ToString();

        // Count how many calls were on the bridge from the beginning of the month
        // for the specific conference number
        lngSessionsCount = ws.getSessionDRsCount(filter);
        // XML that was sent to the server see here:
        // Sample\_InfoSessionDR1.getSessionDRsCount.sent.xml
        // XML that was received from the server see here:
        // Sample\_InfoSessionDR1.getSessionDRsCount.received.xml
        strInfo += "Number of current month calls for the conference: "
            + lngSessionsCount.ToString() + ". \n\r";

        if (lngSessionsCount > 0)
        {
            for (long page = 0; page * PAGE_SIZE < lngSessionsCount; page++)
            {
                // Find all current month calls for the specific conference number
                sessionDRs = ws.getSessionDRs(page * PAGE_SIZE, PAGE_SIZE, filter, "");
                // This sample runs the loop three times and outputs three pages:
                // Page #1. XML that was sent to the server see here:
                // Sample\_InfoSessionDR1.getSessionDRs.page1.sent.xml
                // Page #1. XML that was received from the server see here:
                // Sample\_InfoSessionDR1.getSessionDRs.page1.received.xml
                // Page #2. XML that was sent to the server see here:
                // Sample\_InfoSessionDR1.getSessionDRs.page2.sent.xml
                // Page #2. XML that was received from the server see here:
                // Sample\_InfoSessionDR1.getSessionDRs.page2.received.xml
                // Page #3. XML that was sent to the server see here:
                // Sample\_InfoSessionDR1.getSessionDRs.page3.sent.xml
                // Page #3. XML that was received from the server see here:
                // Sample\_InfoSessionDR1.getSessionDRs.page3.received.xml
            }
        }
    }
}

```



```

/*
List<SessionDR> getSessionDRs(long offset,
                             long limit,
                             java.lang.String filter,
                             java.lang.String order)
* This function returns list of SessionDRs allowed to view.
* For administrator it returns whole list of records.
* Parameters:
  offset - zero based offset in recordset.
  limit - maximum number of objects to return.
  filter - The criteria to use to filter the rows.
           The criteria should be a simple sql conditional statement started with one
           or more SessionDR field names.
  Acceptable operators: <= , >= , != , = , < , > , like , ilike, and
  For example:
    conferenceId = 5424
    created > '2010.08.07/00:00'
  Empty string or null means no filter.
  order - A string specifying SessionDR field name and sort direction.
           For example "conferenceNumber" or "created desc".
           The default direction is asc and can be omitted.
           Empty string or null means no order.
* Accepted fields:
  •conferenceId
  •conferenceNumber
  •created ('yyyy.MM.dd/hh:mm' format)
  •duration
  •role
  •joined
  •customName
  •caller
  •callee
  •addressFrom
  •addressTo
  •conferenceNumber
  •accessCode
  •disconnectReason
* Returns:
  list (array) of SessionDR objects
*/
if (sessionDRs != null && sessionDRs.Length > 0)
{
    strInfo += "Page #" + (page + 1).ToString()
              + ". Calls (SessionDR objects) on the page: "
              + sessionDRs.Length.ToString() + ". \n\r";
    foreach (SessionDR sdr in sessionDRs)
    {
        strInfo += sdr.conferenceNumber.ToString() + "\t"
                  + sdr.conferenceId.ToString() + "\t"
                  + sdr.created.ToShortDateString() + " "
                  + sdr.created.ToShortTimeString() + "\t"
                  + sdr.duration.ToString() + "\t"
                  + sdr.callee + "\t"
                  + sdr.caller + "\t"
                  + sdr.customName + "\t"
                  + sdr.disconnectReason + "\n\r";
    }
}
}
else
{
    strInfo += "No current month calls for the conference found. \n\r";
}

```

```
// Sample of program output: Sample\_InfoSessionDR1.return.jpg
// ** Number of current month calls for the conference: 11.
// ** Page #1. Calls (SessionDR objects) on the page: 5.
// ** 667788 3 15/03/2010 12:50 33 12 admin Normal
// ** 667788 4 15/03/2010 1:25 28 12 admin 'Guest' MP is unavailable ...
// ** 667788 4 15/03/2010 1:16 573 12 admin MP is unavailable ...
// ** 667788 5 15/03/2010 1:33 244 REC SERVER 12 Normal
// ** 667788 5 15/03/2010 1:27 11824 12 admin Dropped by moderator
// ** Page #2. Calls (SessionDR objects) on the page: 5.
// ** 667788 5 15/03/2010 1:27 11810 12 unknown Dropped by moderator
// ** 667788 5 15/03/2010 1:27 11797 12 admin Guest Dropped by moderator
// ** 667788 8 17/03/2010 12:35 975 12 admin Guest Dropped by moderator
// ** 667788 8 17/03/2010 12:32 1389 12 admin conference moderator Dropped...
// ** 667788 8 17/03/2010 12:53 146 12 admin 'Guest' Dropped by moderator
// ** Page #3. Calls (SessionDR objects) on the page: 1.
// ** 667788 8 17/03/2010 12:42 783 12 unknown Dropped by moderator

return strInfo;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_InfoSessionDR1: " + ex.Message;
    return mLastError;
}
}
```

Sample of Historical Calls Filtering (Sample_InfoSessionDR2)

```

/*
Sample of Historical Calls Filtering
Let's review the following scenario:
*   for the current month we need to output all calls that were connected to the
    conferences excluding service calls to the recording server initiated by bridge
    (for instance we should output calling number, called number, conference number,
    conference identifier, date/time when the call was started,
    and how long the call was connected to the conference).
*/
public String Sample_InfoSessionDR2()
{
    // Declare variables
    SessionDR[] sessionDRs;
    DateTime startDate;
    String filter;
    String strInfo;

    try
    {
        mLastError = "";
        strInfo = "";

        // Generate filter that should be user to retrieve SessionDR objects
        startDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
        filter = "created>='" + startDate.ToString("yyyy.MM.dd") + "/00:00'";
        filter += " and conferenceNumber!=0";
        filter += " and callee!='REC_SERVER'";

        // Get all calls based on the specified criteria
        sessionDRs = ws.getSessionDRs(0, 0, filter, "created");
        // XML that was sent to the server see here:
        // Sample\_InfoSessionDR2.getSessionDRs.sent.xml
        // XML that was received from the server see here:
        // Sample\_InfoSessionDR2.getSessionDRs.received.xml
        /*
        List<SessionDR> getSessionDRs(long offset,
                                    long limit,
                                    java.lang.String filter,
                                    java.lang.String order)
        * This function returns list of SessionDRs allowed to view.
        * For administrator it returns whole list of records.
        * Parameters:
          offset - zero based offset in recordset.
          limit - maximum number of objects to return.
          filter - The criteria to use to filter the rows.
                   The criteria should be a simple sql conditional statement started with one or
                   more SessionDR field names.
          Acceptable operators: <= , >= , != , = , < , > , like , ilike, and
          For example:
            conferenceId = 5424
            created > '2010.08.07/00:00'
            Empty string or null means no filter.
          order - A string specifying SessionDR field name and sort direction.
                   For example "conferenceNumber" or "created desc".
                   The default direction is asc and can be omitted.
                   Empty string or null means no order.
        * Accepted fields:
          •conferenceId
          •conferenceNumber
          •created ('yyyy.MM.dd/hh:mm' format)
          •duration
          •role
          •joined
          •customName
          •caller
          •callee
        */
    }
}

```

```

        •addressFrom
        •addressTo
        •conferenceNumber
        •accessCode
        •disconnectReason
    * Returns:
        list (array) of SessionDR objects
    */
    if (sessionDRs != null && sessionDRs.Length > 0)
    {
        strInfo += "Number of current month calls that match to the specified criteria: ";
        strInfo += sessionDRs.Length.ToString() + ".\n\r";
        strInfo += "callee\tcaller\tconferenceNumber\tconferenceId\tcreated\tin conference\n\r";
        foreach (SessionDR sdr in sessionDRs)
        {
            strInfo += sdr.callee + "\t"
                + sdr.caller + "\t"
                + sdr.conferenceNumber.ToString() + "\t"
                + sdr.conferenceId.ToString() + "\t"
                + sdr.created.ToShortDateString() + " " + sdr.created.ToShortTimeString() + "\t"
                + (sdr.created.AddSeconds((Double) sdr.duration) -
                    sdr.joined).TotalSeconds.ToString() + "\n\r";
        }
    }
    else
    {
        strInfo += "No current month calls that match to the specified criteria. \n\r";
    }

    // Sample of program output: Sample InfoSessionDR2.return.jpg
    // ** Number of current month calls that match to the specified criteria: 18.
    // ** callee      caller      conferenceNumber conferenceId created      in conference
    // ** 8665080012 Moderator-Console 758288      2      10/03/2010 7:08 267
    // ** 8665080012 admin      758288      2      10/03/2010 7:10 93
    // ** 8665080012 admin      758288      2      10/03/2010 7:12 28
    // ** 12          admin      667788      3      15/03/2010 12:50 31
    // ** .....

    return strInfo;
}
catch (Exception ex)
{
    mLastError = "Error in " + this.GetType().FullName +
        ".Sample_InfoSessionDR2: " + ex.Message;
    return mLastError;
}
}

```

Appendix B: Support Resources

If you have difficulty with this guide and any of the procedures listed herein, please contact us using the following support resources.

Support Documentation

In addition to this Guide, you may obtain other WYDE Voice documentation from WYDE Voice or from the WYDE Voice documentation Web site: <http://docs.wydevoice.com/>.

Web Support

Our support website is available 24 hours a day, 7 days a week, and 365 days a year at <http://www.wydevoice.com>. You may download patches, support documentation and other technical support information.

Telephone Support

For difficulties with any procedures described in this Guide, please contact us at 866-508-9020 during our normal phone support hours of 7:00 am to 6:00 pm Pacific Standard Time (PST). An engineer will respond to your inquiry within 24 hours.

Email Support

You may also email us your questions at support@wydevoice.com. We will respond to your question within 24 hours.