Web Service API – Programmer's Guide

(version 1.4.31)

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 WIDE VOICE REPRESENTATIVE FOR A COPY.

IN NO EVENT SHALL WIDE 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 WIDE OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright

For the most current versions of documentation, go to the Wide support Web site:
http://www.wydevoice.com/support

December 15, 2009

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

Button Denotes any button caption

Table of Contents

Symbols and Notations in this Manual 3
Table of Contents 4
Tables List 6
Figures List 7
Chapter 1: Introduction 8
Assumed Skills 8
Web Services 9
Definitions 9
Chapter 2: Data Structures 13
General Data Structure 13
Data Classes (Entities) 15
Subscriber 15
Conference Account – Conference User 15
Conference Info 16
DNIS 16
Call Flow 17
Attribute 17
Conference 18
ConferenceDR 18
Session 19
SessionDR 19
Chapter 3: Samples of Functions 21
Wyde Web Services Initialization 21
Sample of Wyde Web Services Initialization 21
Subscribers Management 21
Sample of Subscriber and his Conference Accounts Creation 21
Sample of Subscribers Filtering, Modifications, Conference Accounts Modifications 21
Sample of Subscribers Filtering and Deletion 22
Sample of Getting Conference Users Information 22
Conferences and Calls Management 22
Sample of Conferences Filtering, Changes Secure Mode, Dropping the Conferences 22
Sample of Placing the Entire Conference on Hold, Starting and Stopping Q&A Sessions and Conference Recording 23
Sample of Calls Filtering, Mute the Calls, Dropping the Calls 23
Sample of Setting Custom Name and Placing Calls on Hold 23
CDRs Management 24
Sample of Getting Conferences Historical Information 24
Sample of the Shared Recording Generation 24
Sample of Getting Calls Historical Information 24
Chapter 4: Function Reference 26
Subscribers Management 26
Subscribers' Conference Users Management 28
Conferences and Calls Management 30
CDRs Management 39

Call Flow, DNIS, and Conference Info Management 43
Backend and Frontend Services Management 48
Exceptions 50
Constants 50
Appendix A: Support Resources 52
Support Documentation 52
Web Support 52
Telephone Support 52
Email Support 52

Tables List

Table 1: Properties of Subscriber 15
Table 2: Properties of Confuser 16
Table 3: Properties of ConfInfo 16
Table 4: Properties of DNIS 16
Table 5: Properties of CallFlow 17
Table 6: Properties of Attribute 17
Table 7: Properties of Conference 18
Table 8: Properties of ConferenceDR 19
Table 9: Properties of Session 19
Table 10: Properties of SessionDR 20

Figures List

Figure 1: The Web Services Architecture 9
Figure 2: The UML Class Diagram 14

Chapter 1: Introduction

Wyde VM 1000 and VM 3000 conferencing bridges 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://38.101.116.27/dnca/jAdmin?wsdl.

This Web Service Interfaces – Programmer's Guide is based on Wyde web services API version **1.4.31**. If you use another version of API the same functions may be different and you may need other version of the guide.

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.

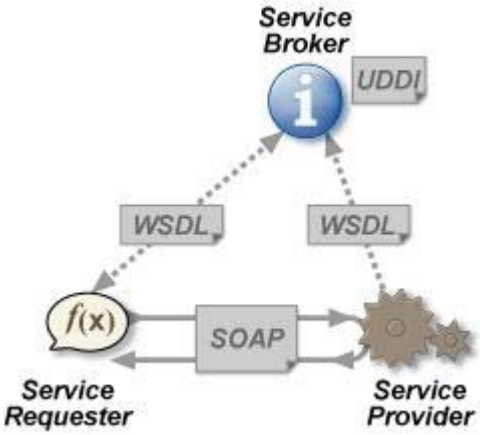

Figure 1: The Web Services Architecture

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 definite 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 outpulsed 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 conferneceId 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 conferneceId 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.

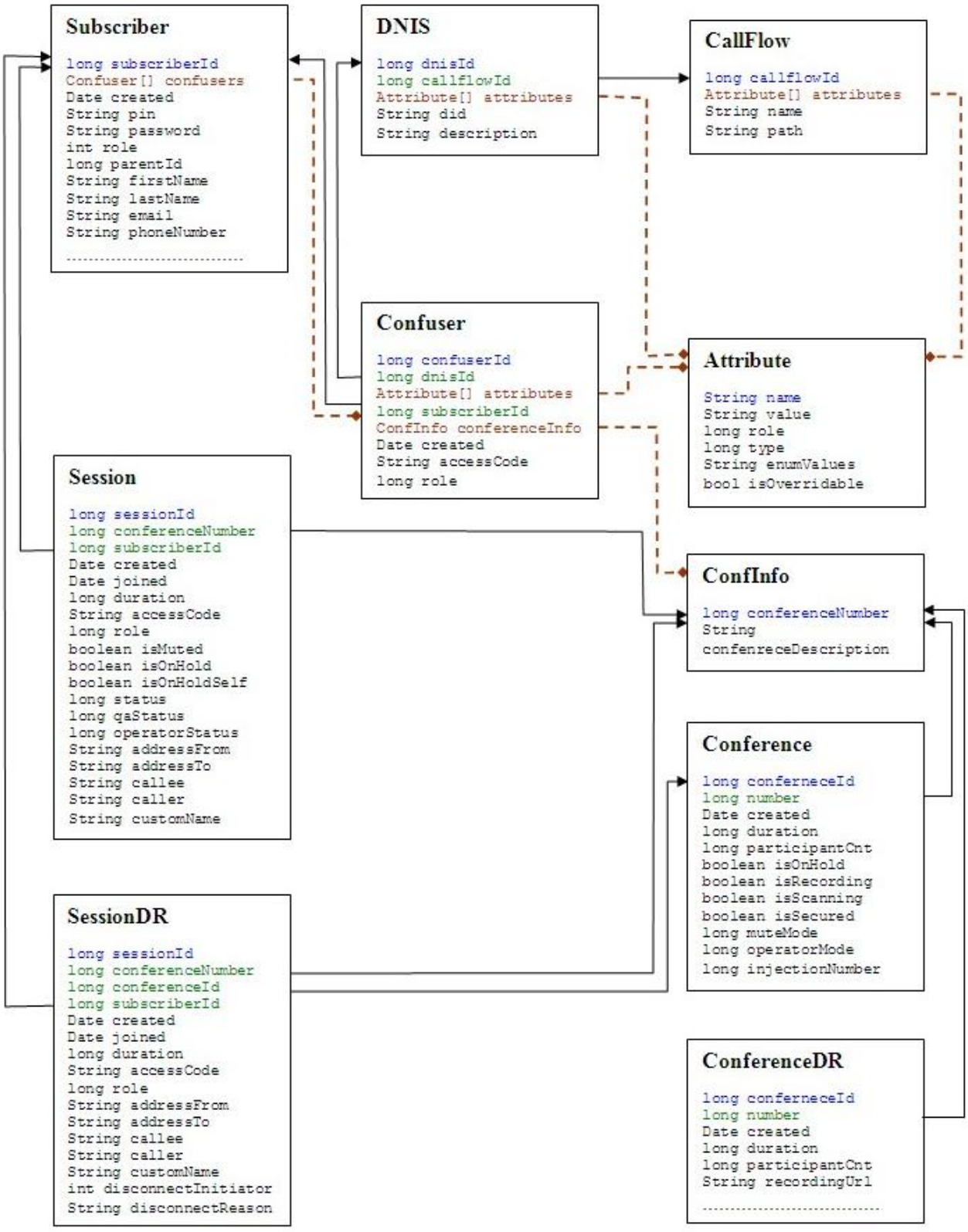

Figure 2: The UML Class Diagram

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 phonenumber used if server needs to dialout 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**
int role	**Subscriber's role (i.e. admin, operator, user, etc.)**
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

Click here to see subscriber XML and class definition.

Conference Account – Conference User

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).

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 (*)
Attribute[] attributes	List of attributes and their values imposed by the calllflow this user is assigned to. These attributes may be overwritten for this particular user or taken from parent or defaults
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

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 conferneceNumber and provide meaningful description of this conference. In this case server automatically assigns a new unique conferenceNumber.

Table 3: Properties of ConfInfo

long conferenceNumber	Identifier of the conference where this user will be assigned after successful authentication. It should be unique across other confNumbers; 0 means create a new one
String confenreceDescription	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

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.

Call Flow

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 5: Properties of CallFlow

long callflowId	**Unique ID assigned by the server**
Attribute[] confuserAttributes	**Template of attributes for DNIS and confusers**
String name	**Callflow name (description) (*)**
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 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 confuesrs can only see those attributes which role matches their own role.

Table 6: 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**
bool isOverridable	**whether or not user can attempt to override this value**
String name	**attribute name like "ALLOW_CONTINUE" (*)**
long role	**confuser Role this attribute belongs to (*)**
long type	**attribute type like TYPE_STRING (0L), TYPE_INT (2L), TYPE_DTMF (3L) (*)**

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 conferneceId, 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 conferneceId. 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 7: Properties of Conference

long conferneceId	Unique ID assigned by the server
Date created	Time when this conference was created - first caller arrived
long duration	Number of seconds which have elapsed since the conference was created
long injectionNumber	For the operator conference this field determines to which conference number the operator conference is connected at the moment
boolean isOnHold	This field determines whether the conference is on hold
boolean isRecording	This field determines whether the conference is recorded
boolean isScanning	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)
boolean isSecured	This field determines whether the conference is secured, i.e. new calls allowed to join to the conference or not
long muteMode	This field determines mute mode MUTE_MODE_OPEN (0L), MUTE_MODE_QUESTION (1L), MUTE_MODE_CLOSED (2L) When MUTE_MODE_OPEN mode is enabled any conference participant can talk and mute/unmute himself. When MUTE_MODE_QUESTION mode is enabled all conference participants are muted however any of them can unmute himself to ask a question. When MUTE_MODE_CLOSED mode is enabled all conference participants are muted and can not
long number	This is a conference number
long operatorMode	This field determines operators conference mode CONFERENCE_REGULAR (0L), CONFERENCE_OPERATOR (1L), CONFERENCE_OPERATOR_LISTEN (2L), CONFERENCE_OPERATOR_AUTOLISTEN (3L), CONFERENCE_AUTOLISTEN_SLEEP (4L)
long participantCnt	Number of participants in the conference

Click here to see conference XML and class definition.

ConferenceDR

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

Table 8: Properties of ConferenceDR

long conferneceId	Unique ID assigned by the server
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
bool hasRecording	Whether or not conference was recorded
long number	This is a conference number
long participantCnt	Number of participants in the conference
String recordingUrl	URL for the recording
String sharedRecordingUrl	URL for shared recording
long ulawDuration	Recording duration in seconds

Click here to see conferenceDR 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 9: Properties of Session

String accessCode	access code entered by caller
String addressFrom	Full address FROM
String addressTo	Full address TO
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 conferenceNumber	ConferneceNumber 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 duration	Number of seconds which have elapsed since the session started
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
Date joined	Time when this session joined to the conference
long operatorStatus	OPERATOR_STATUS_IDLE (0L), OPERATOR_STATUS_WAIT (1L), OPERATOR_STATUS_TALK (2L)
long qaStatus	QA_STATUS_IDLE (0L), QA_STATUS_RAISEDHAND (1L), QA_STAUS_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), MODE_PARTICIPANT (2L), MODE_LISTENER (3L)
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
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 10: Properties of SessionDR

String accessCode	**access code entered by caller**
String addressFrom	**Full address FROM**
String addressTo	**Full address TO**
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 number this session belongs to**
long conferenceNumber	**Conference ID 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)**
int disconnectInitiator	**Shows who initiated a disconnect (user, bridge)**
String disconnectReason	**A string showing detailed info about disconnect**
long duration	**Number of seconds which have elapsed since the session started and before disconnect**
Date joined	**Time when this session joined to the conference**
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**

Click here to see sessionDR 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.

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 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' properties, but this method also can be used to create, modify or delete conference accounts (conference users) with their attributes that belong to this subscriber.

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

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 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.

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 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), *muteConference* (to start and stop Q&A session for the conference), *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 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 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.

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 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 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.

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 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 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.
  Acceptable operators: <= , >= , != , = , < , > , like *
  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
  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.

Acceptable operators: <= , >= , != , = , < , > , like

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

Acceptable fields:

- subscriberId
- parentId
- pin
- password
- firstName
- lastName
- email
- address1
- city
- country
- phoneNumber

Empty string or null means no filter.

Returns:

long count of Subscribers

Throws Exceptions:

ServerException
AccessDeniedException

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

Parameters:

s – The Subscriber object

Returns:

created Subscriber object

Throws Exceptions:

ServerException
AccessDeniedException
ObjectValidationException

- ***updateSubscriber*** `(Subscriber s)` – 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:

s – 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` and `createConfuser`.
  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

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.

Acceptable operators: <= , >= , != , = , < , > , like

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
- confuserId
- role
- dnisId
- accessCode
- 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.

  Acceptable operators: <= , >= , != , = , < , > , like

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

  Acceptable fields:

  - subscriberId
  - confuserId
  - role
  - dnisId
  - accessCode
  - 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

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
  For example number='12' or number like'%2%' or duration >= 15.
  Accepted fields:
  • conferenceId
  • number
  • 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 "number" 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
  For example number='12' or number like'%2%' or duration >= 15.
  Accepted fields:
  • conferenceId
  • number
  • 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 conferenceId, 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

  For example addressTo='12' or addressTo like'%2%' or duration >= 15.

  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.

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*** `(long conferenceId, String filter)` – This function returns number of calls on the bridge which matches the filter provided.
  Parameters:
  conferenceId - Conference identifier. If parameter is less than zero Session objects for all Conference will be counted.
  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
  For example caller='12' or caller like'%2%' or duration >= 15.
  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:
  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 secureConfernece, 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
  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 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
- ***qaEngage*** `(long sessionId)` – Engages Q&A session for the conference participant referenced by ID.
  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.
  Parameters:
  sessionId – The Session identifier
  Returns:
  void
  Throws Exceptions:

ServerException
AccessDeniedException
ObjectNotFoundException

- ***startConferenceRecording*** `(long conferenceId)` – This function starts the conference recording.
  Parameters:
  conferenceId – The Conference identifier
  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
- ***dialout*** `(String phoneNumber, long confuserId, String attributes)` – This function initiates outgoing call to the specified phone number and tries to connect participant to the specific conference.
  Parameters:
  phoneNumber – The phone number to dial-out
  confuserId – The identifier of Confuser which role and access code will be used
  attributes – The custom attributes (reserved field)
  Returns:
  void
  Throws Exceptions:
  ServerException
  AccessDeniedException
  ObjectNotFoundException
  ObjectValidationException
- ***dialoutEx*** `(String calledNumber, String conferenceDID, long conferenceNumber, String accessCode)` – This function initiates outgoing call to specified phone number and tries to connect participant to the specified conference.
  Parameters:
  calledNumber – The phone number to dial-out
  conferenceDID – 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

Returns:
void
Throws Exceptions:
ServerException
AccessDeniedException
ObjectNotFoundException
ObjectValidationException

- ***startScan*** `(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
- ***stopScan*** `(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
- ***startListen*** `(long conferenceId, long targetId)` – This function connects and starts listen 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)
  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

- ***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

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
For example:
conferenceId = 5424
duration > 300 and duration < 400
duration > 300 and number = 160
participantCnt > 2 and participantCnt < 22

created > '2008.08.07/00:00'

Accepted fields:

- conferenceId
- number
- created ('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 "number" 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

  For example:

  conferenceId = 5424
  duration > 300 and duration < 400
  duration > 300 and number = 160
  participantCnt > 2 and participantCnt < 22
  created > '2008.08.07/00:00'

  Accepted fields:

  - conferenceId
  - number
  - created ('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
For example:
conferenceId = 5424
created > '2008.08.10/00:00' and created lt; '2008.08.20/00:00'
Accepted fields:
  - conferenceId
  - number
  - created ('yyyy.MM.dd/hh:mm' format)
  - duration
  - role
  - joined
  - customName
  - caller;
  - calle;
  - addressFrom;
  - addressTo;
  - conferenceNumber;
  - 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

For example:

conferenceId = 5424

created > '2008.08.10/00:00' and created < '2008.08.20/00:00'

Accepted fields:

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

Empty string or null means no filter.

Returns:

long count of SessionDR objects

Throws Exceptions:

ServerException

AccessDeniedException

- ***listAudioFiles*** `(long confNumber, String patter)` – 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:

  confNumber – 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 confNumber, String patter)` – This function deletes user's audio files (recordings and uploaded streaming audio) according to the specified pattern and conference number.

  Parameters:

  confNumber – The conference number (note: it is not conferenceId)

  pattern – The filename wildcard pattern

  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, bool 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

Call Flow, DNIS, and Conference Info 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

- ***getCallFlowDirs*** `()` – This function returns list of all registered call flow directories on the current Wyde bridge.

  Returns:

  list (array) of strings representing the call flow directories

  Throws Exceptions:

  ServerException
  AccessDeniedException

- ***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. 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
  For example name='12' or name like'%2%' or collFlowId >= 15.
  Accepted fields:
  • callFlowId
  • name
  • path
  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
  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
For example name='12' or name like'%2%' or callFlowId >= 15.

Accepted fields:
- callFlowId
- dnisId
- did
- 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.

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
For example name='12' or name like'%2%' or callFlowId >= 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.
Accepted fields:
• callFlowId
• dnisId
• did
• description
Empty string or null means no order.

Returns:
list of DNIS objects
Throws Exceptions:
ServerException
AccessDeniedException

- ***createCallFlow*** `(CallFlow cf)` – The method creates CallFlow object. Note: this function is deprecated and probably will not be included in next versions; it is not recommended to use this function.
  Parameters:
  cf – The CallFlow object
  Returns:
  created CallFlow object
  Throws Exceptions:
  ServerException
  AccessDeniedException
  ObjectValidationException
- ***updateCallFlow*** `(CallFlow cf)` – The method updates CallFlow object.
  Parameters:
  cf – The CallFlow object
  Returns:
  updated CallFlow object
  Throws Exceptions:
  ServerException
  AccessDeniedException
  ObjectValidationException
- ***deleteCallFlow*** `(long callFlowId)` – This function deletes CallFlow referenced by the ID. Note: this function is deprecated and probably will not be included in next versions; it is not recommended to use this function.
  Parameters:
  callFlowId – The CallFlow identifier
  Returns:
  void
  Throws Exceptions:
  ServerException

AccessDeniedException
ObjectNotFoundException

- ***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. 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 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*** `(List<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 id)` – This function returns the collection of Attribute Name – Attribute Description pairs for the specified CallFlow object (actually the list of allowed attributes with descriptions).
Parameters:
id – The CallFlow identifier
Returns:
list of Attribute Name – Attribute Description pairs
Throws Exceptions:
ServerException
AccessDeniedException
ObjectNotFoundException
- ***getConferenceNumbers*** `()` – This function returns the list of registered ConfInfo objects in the client's scope (for admin subscriber it returns the information about all ConfInfo objects, for non admin subscriber it returns the information about ConfInfo objects available for this subscriber only).
Returns:
list of ConfInfo objects
Throws Exceptions:
ServerException
AccessDeniedException

Backend and Frontend Services Management

- ***getVersion*** `()` – Returns version of the installed software (like 1.4.31 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
Throws Exceptions:
ServerException
AccessDeniedException
- ***getFrontendInfo*** `(String group)` – Returns some statistic about frontend.
Parameters:

group – group name, for example confcount, confsize, cmdcount, partcount 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 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
```

- ***Session***

```
public static long INCOMING = 0L
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 OUTGOING = 1L
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_CONFERENCE = 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_DTMF = 3L
public static long TYPE_INT = 2L
public static long TYPE_STRING = 0L
```

Appendix A: Support Resources

During installation, if you have difficulty with any of the installation procedures listed herein, please contact us using the following support resources.

Support Documentation

In addition to this Installation Guide, you may obtain the Wyde Voice Administration Guide from Wyde Voice or from the support section of http://www.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 installation, 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.