



Real Time Interface – Programmer's Guide

(version 2.3)

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/>

April 26, 2012

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

Folder/File Denotes any folders (paths) or files names

`commands` Denotes any callback handlers, DTMF commands, attributes and parameters

Table of Contents

Symbols and Notations in this Manual	3
Table of Contents	4
Figures List	6
Chapter 1: Introduction	7
Assumed Skills	7
Basic Concepts	7
Real Time Interface	7
Session Initiation Protocol (SIP)	8
Real-time Transport Protocol (RTP)	9
Active Speaker Daemon (ASD)	9
Definitions, Acronyms and Abbreviations	10
Chapter 2: RT Interface Transports Protocols	13
SIP Transport	13
Control Protocol Format	13
Keep-Alive Mechanism	14
Joining a Conference	15
Join Response	16
Leaving a Conference	17
Keep-Alive Format	17
Active Speaker Notifications in Real Time Protocol	17
ASD Transport	18
Configuration	18
Authorization	19
Calls to Conferences	19
Active Speaker Notifications in ASD	20
Real Time Commands Requests in ASD	21
Bundling Connections Using Audio Keys	22
Chapter 3: Message Format Reference	24
Request Format	25
ASSOCIATE (Request to Change Audio Key of the Connection)	25
CONF-DROP (Request to Drop the Conference)	25
DROP (Request to Drop the Connection)	25
HOLD (Request to Change Hold Status of the Connection)	26
HOLD-GROUP (Request to Change Group Hold Policy)	26
MUTE (Request to Mute the Connection)	26
MUTE-GROUP (Request to Change Group Mute Policy)	26
MUTE-SELF (Request to Self-Mute the Connection)	27
OPERATOR_CALL_MOVE (Request to Move the Current User to other Conference)	
.....	27
OPERATOR_DIALOUT (Request to Dialout from Operator Conference)	28
OPERATOR_LISTEN (Operator Request to Listen the Conference)	28
OPERATOR_MESSAGE (Operator Request to Control Operator Message)	28
OPERATOR_MESSAGE_TO_CONFERENCE (Operator Request to Record and	
Play Message to the Conference)	29

OPERATOR_REJECT (Operator Request to Reject the User from the Queue)	30
OPERATOR_SCAN (Operator Request to Scan Conferences).....	30
OPERATOR_TALK (Operator Request to Talk to the User).....	31
PLAY-FILE (Request to Control the Audio File Playback)	31
QA-CLEAR-QUEUE (Request to Clear Q&A Queue)	32
QA-MODE (Request to Start/Stop Q&A Mode)	32
QA-MUTE (Request to Mute/Unmute the Current Questioner).....	32
QA-REQUEST (Request to Speak in Q&A Sessions).....	32
QA-TALK (Request to Allow/Disallow Speaking)	33
QA-TALK-NEXT (Request to Move to the Next Questioner)	33
RECORDING (Request to Start/Stop the Conference Recording)	33
SECURE (Request to Secure/Unsecure the Conference).....	33
SET-CUSTOMNAME (Request to Change Custom Name of the Call).....	34
SET-GAIN (Request to Change Volume Level of the Call).....	34
SET-JOBCODE (Request to Change Conference Job Code)	34
SET-ROLE (Request to Change Role of the Connection)	34
START-BROADCAST (Start Broadcast).....	35
Response Format	36
Notification Format	37
NOTIFY-ASSOCIATE	37
NOTIFY-CONFERENCE.....	38
NOTIFY-DROP	39
NOTIFY-GROUP.....	39
NOTIFY-HOLD	39
NOTIFY-JOIN	40
NOTIFY-LOCK	41
NOTIFY-MUTE.....	41
NOTIFY-OPERATOR-CONNECT	41
NOTIFY-OPERATOR-ENGAGE	42
NOTIFY-OPERATOR-MESSAGE	42
NOTIFY-OPERATOR-MESSAGE-TO-CONFERENCE.....	43
NOTIFY-OPERATOR-QUEUE	44
NOTIFY-OPERATOR-SCAN	44
NOTIFY-OPERATOR-SCAN-ACTIVE	44
NOTIFY-OPERATOR-TALK	45
NOTIFY-QA_MODE.....	45
NOTIFY-QA_STATUS	45
NOTIFY-RECORDING.....	46
NOTIFY-SET_CUSTOMNAME.....	46
NOTIFY-SET_GAIN	46
NOTIFY-SET-ROLE	46
NOTIFY-TERMINATION	46
Appendix A: Support Resources	48
Support Documentation.....	48
Web Support.....	48
Telephone Support.....	48
Email Support.....	48

Figures List

Figure 1: Real Time Interface Transports.....	13
Figure 2: SIP Messages Exchange	14
Figure 3: ASD Authorization	19
Figure 4: ASD Call to the Conference	20
Figure 5: Muting the Specific Call in ASD	21

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

- real time (RT) interface,
- web services API,
- 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 real time interface only. Other APIs are being described in the separate documentation (for instance, web services API is being described in "Web Service API – Programmer's Guide", etc.).

Assumed Skills

This call flow development 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)
- Session Initiation Protocol (SIP) basics
- VOIP basics
- TCP/IP networking
- Web Administration Interface – User Guide

Basic Concepts

Real Time Interface

The real time interface in communications between the server and the client performs PUSH notification, i.e. it sends/receives the messages once any of events occurred on the bridge. Comparing RT interface with the web services API, they perform PULL notifications, i.e. the requests are being sent to server using web methods and the client receives responses only on the requests that were sent.

The real time interface can be implemented using two approaches:

- RT interface makes SIP call to the bridge using special client – it performs SIP call to send, receives commands, and exchanges information about the conferences – it is possible to exchange text messages (Info) inside of SIP call;
- RT interface makes the same via another transport protocol – Active Speaker Daemon (ASD) proxy server, the specific service implement and installed as the part of WYDE bridge software.

Session Initiation Protocol (SIP)

The Session Initiation Protocol (SIP) is a signaling protocol, widely used for controlling multimedia communication sessions such as voice and video calls over Internet Protocol (IP). The protocol can be used for creating, modifying and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or several media streams. The modification can involve changing addresses or ports, inviting more participants, adding or deleting media streams, etc. Other feasible application examples include video conferencing, streaming multimedia distribution, instant messaging, presence information and online games.

The SIP protocol is a TCP/IP-based Application Layer protocol. SIP is designed to be independent of the underlying transport layer; it can run on Transmission Control Protocol (TCP), User Datagram Protocol (UDP), or Stream Control Transmission Protocol (SCTP). It is a text-based protocol, incorporating many elements of the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP), allowing for direct inspection by administrators.

SIP is a text-based protocol with syntax similar to that of HTTP. There are two different types of SIP messages: requests and responses. The first line of a request has a method, defining the nature of the request, and a Request-URI, indicating where the request should be sent. The first line of a response has a response code.

The following methods (types) are defined for SIP requests:

- INVITE: Used to establish a media session between user agents; invites users to communication session; usually contains SDP description of the session.
- MESSAGE: Used to transfer instant messages in SIP (instant messaging).
- ACK: Confirms reliable message exchanges (confirms INVITE response receiving).
- BYE: Terminates a session between two users in a conference; can be transferred by any participant of the sessions.
- REGISTER: Used by a UA to notify its current IP address and the URLs for which it would like to receive calls.
- CANCEL: Terminates a pending request (does not affect on completed requests).
- OPTIONS: Requests information about the capabilities of a caller, without setting up a call.
- INFO: Used to transfer the information which does not change communication session state.
- PRACK: Used for temporary acknowledgments.
- SUBSCRIBE: Subscription to receive acknowledgments about the events.
- NOTIFY: Notification about the events for subscriber.
- PUBLISH: Events publishing on server.
- REFER: Receiver request to transfer SIP request.
- UPDATE: Modifies the state of the communication session without changes in the dialog state.

The WYDE bridge software real time interface uses the following SIP requests only: INVITE (to join the conference, special attributes are being added to SIP INVITE for

configuration purposes), MESSAGE (as transport), ACK (to receive notifications/acknowledgments, real time commands are being transferred only after ACK received), and BYE (to leave the conference).

The SIP response types fall in one of the following categories:

- Provisional (1xx): Request received and being processed.
- Success (2xx): The action was successfully received, understood, and accepted.
- Redirection (3xx): Further action needs to be taken (typically by sender) to complete the request.
- Client Error (4xx): The request contains bad syntax or cannot be fulfilled at the server.
- Server Error (5xx): The server failed to fulfill an apparently valid request.
- Global Failure (6xx): The request cannot be fulfilled at any server.

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

- SIP: Session Initiation Protocol – <http://tools.ietf.org/html/rfc3261>
- The Stream Control Transmission Protocol (SCTP) as a Transport for the Session Initiation Protocol (SIP) – <http://tools.ietf.org/html/rfc4168>
- Session Initiation Protocol for Telephones (SIP-T): Context and Architectures – <http://www.ietf.org/rfc/rfc3372.txt>
- Session Initiation Protocol Core – <http://www.ietf.org/dyn/wg/charter/sipcore-charter.html>

Real-time Transport Protocol (RTP)

The Real-time Transport Protocol (RTP) defines a standardized packet format for delivering audio and video over the Internet. RTP is used extensively in communication and entertainment systems that involve streaming media, such as telephony, video teleconference applications and web-based push to talk features. For these it carries media streams controlled by H.323, MGCP, Megaco, SCCP, or Session Initiation Protocol (SIP) signaling protocols, making it one of the technical foundations of the Voice over IP industry.

RTP is usually used in conjunction with the RTP Control Protocol (RTCP). While RTP carries the media streams (e.g., audio and video) or out-of-band events signaling (DTMF in separate payload type), RTCP is used to monitor transmission statistics and quality of service (QoS) information. When both protocols are used in conjunction, RTP is usually originated and received on even port numbers, whereas RTCP uses the next higher odd port number.

The detail information about RTP can be read in the following article:

- RTP: A Transport Protocol for Real-Time Applications – <http://tools.ietf.org/html/rfc3550>

Active Speaker Daemon (ASD)

Active Speaker Daemon (ASD) proxy server is the specific service that acts as an intermediary between the client machine with TCP/IP communications and the WYDE server with SIP communications. From one side it receives TCP/IP messages from the

client, transforms them into SIP protocol, makes SIP call to the server, and sends client's messages to the server using SIP protocol; from another side it converts SIP messages of the server into TCP/IP protocol and sends TCP/IP messages to the client.

The commands, requests, responses and notifications are the same in SIP and in ASD. Using ASD it is possible to use usual telecommunication software (like *telnet*) to exchange (send/receive) the messages between server and client.

Definitions, Acronyms and Abbreviations

In order to discuss the WYDE Real Time Interface development 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:

- **VoIP** – **V**oice **o**ver **I**nternet **P**rotocol, a term that refers to the capture/playback of audio streams and their transmission over IP based networks.
- **End Point (EP)** – A generic term used to denote the application running on end-user machines in a VoIP.
- **Public Switched Telephone Network (PSTN)** – the traditional phone system.
- **Bridge** – A server that hosts voice conferences. Participants can use PSTN or VoIP connections to connect to the bridge. It is responsible for mixing the signals and sending the result back to the participants.
- **Gateway** – A gateway server between PSTN and VoIP, i.e. a server that terminates end point connections and routes VoIP data between an end point and the bridge.
- **Conference User** – A user in a conference. Each connection to the conference bridge is associated with exactly one conference user. An end point can be associated with any number of conference users. A conference user may or may not be associated with an end point. The conference user can have one of the roles: host, participant or listener.
- **Conference** – An audio meeting hosted on a bridge and consisting of PSTN and/or VoIP participants. 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.
- **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.
- **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.
- **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.

- **Session ID** – The unique identifier generated by the bridge for each session (connection, VoIP as well as PSTN) established between a conference user and the bridge. The session id is unique within a given conference.
- **Audio Key** – A key sequence that is used to group different calls from the same conference in a bundle to manage these calls using real-time or another external interface. Audio key is short identifier generated externally and provided to the bridge at the time of joining a conference. Audio key is being generated by real-time application, for instance Moderator-Console, the user can enter the same audio key on his DTMF keypad, usually as #audio key#, these calls (the call from real-time application and the user call to the conference) are being grouped together and the real-time application can manage this user call (the call with the same audio key), for instance mute the call, etc.
- **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.
- **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.
- **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.
- **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. This is the most privileged role in a conference. By default,

connections in this role can send and receive RTP data (i.e. the corresponding participant is allowed to speak and listen). They also are allowed to execute control actions on all connections and roles.

- **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. Connections in this role must be allowed to send and receive RTP data by default. They can execute mute and un-mute commands on their own connections (associated with the same audio key); but not on other connections. They are allowed to drop connections within the same bundle (except where the audio key = 0).
- **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. Connections in this role must not have the privilege to speak. They are allowed to send RTP packets to provide feedback for bandwidth adaptively on the stream sent by the bridge. They are allowed to drop connections that are within the same bundle (except where the audio key = 0). Note: users in listener role can be un-muted to enable them to talk; however, the listener group as a whole will never be un-muted.
- **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.
- **Attribute** – In terms of WYDE API, a data structure is used to carry attributes for call flow, DNIS and conference account (user). 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.

Chapter 2: RT Interface Transports Protocols

In this section we will explain different transport protocols that can be used in real time communications between WYDE bridge and clients software. These transports are SIP transport and ASD (TCP/IP) transport.

Real time protocol is the same for both transports (SIP and ASD). The bridge supports receiving requests and sending responses / notifications on SIP using MESSAGE; the content-type of these messages is "text/plain". But all requests, responses, and notifications are the same for both transports – they are described in Chapter 3: Message Format Reference of this guide.

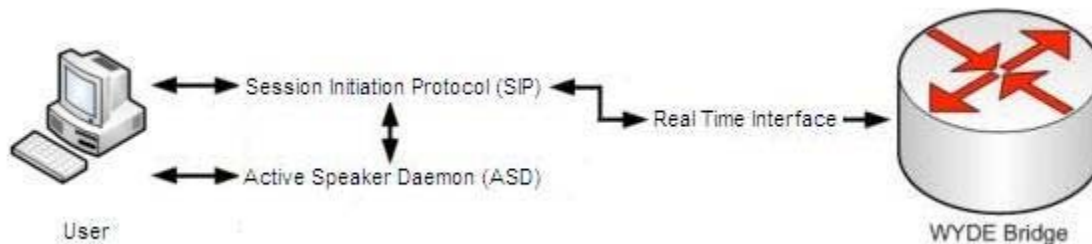


Figure 1: Real Time Interface Transports

SIP Transport

Control Protocol Format

The following figure represents a possible sequence of control messages exchanged between the gateway and the bridge.

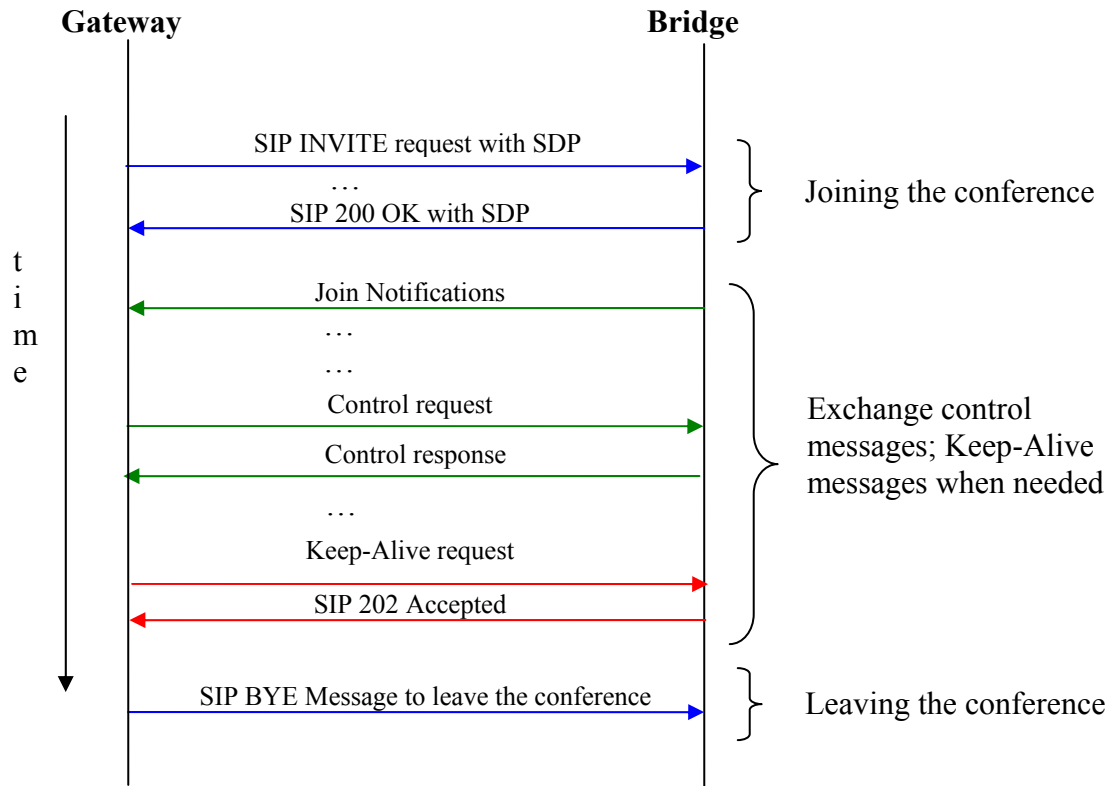


Figure 2: SIP Messages Exchange

Note: SIP Level Acks and other 200 OKs received and sent are not shown for all the messages.

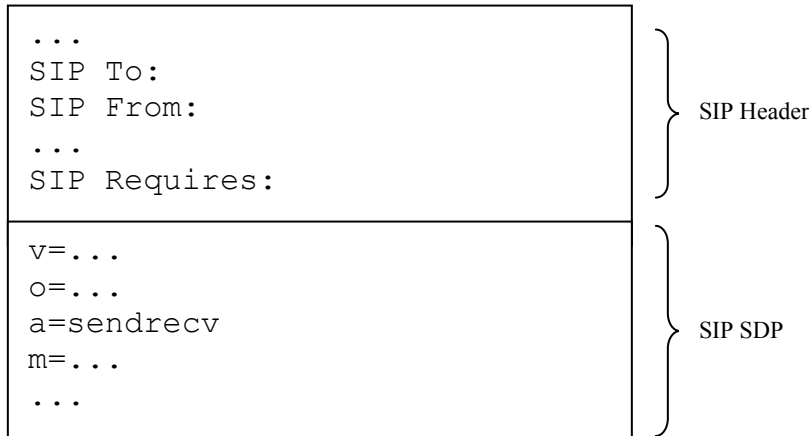
Keep-Alive Mechanism

A connection is considered to be alive based on the activity on its SIP Connection. No assumption is made based on the status of the RTP connection. Keep-Alive messages are periodically sent from the gateway if no control requests need to be sent to the bridge.

- If no Control Request or Keep-Alive has been received for 120 seconds, the bridge marks the connection dead and cleanup the state for the connection.
- The bridge sends a response (SIP Level 202 Accepted or similar) if the connection is alive at its end. A SIP error is returned if the bridge has cleaned up the connection.
- If the bridge has failed to receive a Keep-Alive or failed to send a control message (response or notification) for 120 seconds, the connection is being marked dead and cleaned up.

Joining a Conference

The gateway or client's SIP phone sends a SIP INVITE message to join a conference. The SIP INVITE has the following format:



In this request SIP Header To should have the following format:

```
To:<sip:dnisNumber[_accessCode][_role]]@ipAddress>
```

where

ipAddress – IP address of the bridge;
dnisNumber – DNIS number of the conference;
accessCode – the access code for the conference;
role – the role in the conference, can take on of the values:
 Moderator|Speaker|Listener

Parameters *accessCode* and *role* are optional.

If *accessCode* exists in SIP Header To – the bridge activates *fastJoin* function (conference fast entry); in this case on call entry the call flow executes *fastjoin_handler* instead of *entry_handler* (see “Call Flow Development – Programmer’s Guide” for detail information).

SIP Header From should have the following format:

```
From:<sip:callerNumber@ipAddress>
```

By default the RT protocol and Active Speaker Notifications are switched off.

To switch on the RT protocol for the specific call, SIP INVITE should contain SIP Header Require in the following:

```
Require: version,VoIP[,RT][,NoPrompt]
```

where

version – indicates the version of the control protocol being used. The version specification in this header helps to make changes to the control protocol while maintaining backwards compatibility with previous versions of the protocol/end point.

VoIP – token is used to indicate that this SIP connection is actually associated with a VoIP based end point. PSTN calls forwarded by the SIP gateway will not have this token.

RT – indicates that real time notifications must be sent on this connection. If the *Require* header does not have the *RT* token, notifications must not be sent.

NoPrompt – indicates that the IVR prompts should not be played on the connection at the time of join and the participant must be placed in the conference directly. The *NoPrompt* token will be used for cases where the user reconnects to the bridge – in this case, playing prompts again would not lead to a good user experience.

To switch on the Active Speaker notifications the SDP (Session Description Protocol) request should contain the following attribute:

```
a=x-ActiveSpeaker: on|off
```

This attribute *on* value indicates if Active Speaker updates should be sent on RTP. This attribute *off* value indicates that no such updates are required.

The media attribute (*m=* line in SDP) indicates the type of media transport, type and the format list. If the connection exchanges audio stream, then it will use the *m=audio* line according to the standard specification. For inactive connections, the line will read: “*m=application <source port> udp as*” which indicates that the content exchanged uses a specific application format: *as* (active speaker updates in RTP headers) using UDP as the underlying transport.

Join Response

The bridge responds to the SIP INVITE with provisional responses such as 100, 180 and then provides a final 200 OK response with the answer SDP.

The typical answer SDP has the following fields:

```
v=0
o=root 11406 11406 IN IP4 <[IP Address]>
s=session
i=connectionId:1122
c=IN IP4 <IP Address>
t=0 0
m=audio <Source RTP Port> RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=silenceSupp:off - - - -
```

In addition to the standard fields in the answer SDP, the bridge supports a custom information field that specifies the connection identifier that was assigned to the connection.

The format for the information field is:

```
i=connectionId: [connection id]
```

Example:


```
i=connectionId:1122
```

Sending the session attribute “a=sendrecv|recvonly|sendonly|inactive” in a join response is optional.

For an inactive connection with “m=application” media line, the bridge sends an “m=application” media line in the response.

Leaving a Conference

The gateway sends a “SIP BYE” to leave a conference. The bridge sends a “200 OK” to the BYE.

Keep-Alive Format

The gateway sends a SIP control message with the keyword “KEEP-ALIVE Seq#” to indicate a keep-alive request.

A SIP level 202 Accepted is expected as a response to the KEEP-ALIVE.

Active Speaker Notifications in Real Time Protocol

The streaming protocol provides active speaker notification in real time.

- The bridge supports sending Active Speaker Updates via RTP using the CSRC fields. The CC field of the RTP header indicates the total number of speakers listed in the header. You can read detail information about these field in RTP documentation.
- The bridge uses ConnectionIds in CSRC fields.
- The bridge reserves the least significant 4 bits of the CSRC field to indicate speaker loudness and use the most significant 28 bits to specify the ConnectionId.
- The bridge sends Active Speaker Updates whenever there is a change to the current speaker list. The update specifies the list of currently active speakers. This update is being sent instantaneously (within 10ms of the change).
 - When all speakers are silent, the CC field is being set to 1 and a CSRC value of -1 should be used to positively indicate silence.
- In addition, the bridge sends Active Speaker Updates every 5 seconds even if there is no change in order to let the receiver recover from the loss of an earlier update.

To turn on Active Speaker notifications SIP INVITE message SDP (Session Description Protocol) fields should contain the following attribute:

Active Speaker Update attribute: a=x-ActiveSpeaker:on|off

This attribute when set on indicates that Active Speaker updates should be sent on RTP.

This attribute when set off indicates that no such updates are required.

ASD Transport

When Active Speaker Daemon (ASD) server starts for incoming connections it opens TCP port specified in the configuration file. In addition ASD has its own communication protocol for TCP connections. ASD receives TCP connections, makes authorization, and sends to the bridge the information to what conference it is necessary to be connected. ASD is trying to make control calls to this conference, and sends all messages back to TCP client.

In this approach the client application should not use SIP and implement SIP call to the server. ASD is responsible for all of this; it is performing SIP call to the server and proxies all messages in TCP/IP.

Configuration

On the bridge ASD is being configured in `/usr/local/DNCA/etc/asd.cfg` file. The possible ASD configuration file could be the following:

asd.cfg

```
log-filename = /usr/local/DNCA/log/asd.log
log-level    = event

daemon = 1

local-ip     = 192.168.1.5
sip-server   = 192.168.1.5
sip-port     = 5150
tcp-port     = 5142
rtpmin       = 5142
rtpmax       = 15144

dc-address   = 0.0.0.0:4480

console-address = 127.0.0.1:5140
```

The following parameters can be changed in this `asd.cfg` configuration file:

- `local-ip` – the IP address of the computer where ASD service is running;
- `sip-server` – the IP address of your bridge, i.e. the computer where the WYDE bridge software core components were installed;
- `sip-port` – ASD port for SIP messages exchange;
- `tcp-port` – the port for ASD connections, i.e. the port where ASD is listening the incoming connections;
- `rtpmin` – the minimum allowed RTP port;
- `rtpmax` – the maximum allowed RTP port.

You should update `local-ip` and `sip-server` entries of this file and set your server IP address here. If you wish you can update other configuration parameters in this file as well. After any of these file changes you should execute the command:

```
service asd restart
```

To manage the conferences and calls via ASD the client application must connect to the server via the port specified in configuration file (default 5142). If you would like to use telnet program for testing you can run:

```
telnet <your server IP> <your tcp port>
```

for this purpose, where

- <your server IP> – parameter local-ip from *asd.cfg*;
- <your tcp port> – parameter tcp-port from *asd.cfg*.

Authorization

After you have connected to your ASD server you should authorize your connection. To do that you should run the command:

```
LOGIN
```

(without parameters). As the response you will receive:

```
NONCE <key>
```

(for instance, NONCE W:]F&Th{e}R98cdy*o{%4A}#dZQXz9).

The received key should be encoded and to the server should be sent the command:

```
LOGIN <encoded key>
```

(for instance, LOGIN gATKx4S*C9bRnYmj&QoS@]ReKc<fcW).

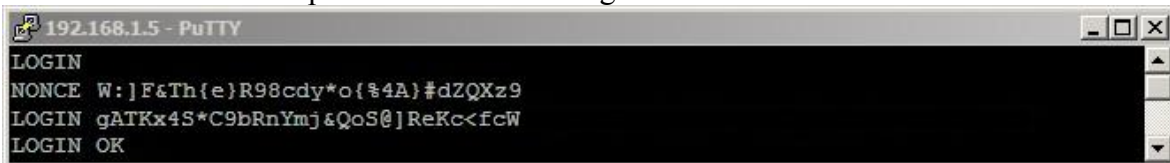
If the login was successful you will receive the message:

```
LOGIN OK
```

If the login was unsuccessful you will receive the message:

```
LOGIN FAIL
```

The ASD authorization process is shown on Figure 3.



```
192.168.1.5 - PuTTY
LOGIN
NONCE W:]F&Th{e}R98cdy*o{%4A}#dZQXz9
LOGIN gATKx4S*C9bRnYmj&QoS@]ReKc<fcW
LOGIN OK
```

Figure 3: ASD Authorization



Please contact WYDE Voice Support Team if you need algorithm how the received key could be encoded.

Calls to Conferences

To make the call to specific conference you should use the following command:

```
CONFERENCE <DNIS number> <access code>
```

```
[<role: host|participant>] [<custom name>]
```

- `<DNIS number>` – the parameter denotes DNIS (phone) number to call;
- `<access code>` – the parameter denotes the actual access code that should be used to connect to the conference;
- `<role: host|participant>` – the optional parameter denotes the role that should be granted to the call in the conference;
- `<custom name>` – the optional parameter denotes the custom name that should be assigned to the call in the conference.

For instance you can run

```
CONFERENCE 8665089020 1111
```

to call to DNIS number 8665089020 and use access code 1111.

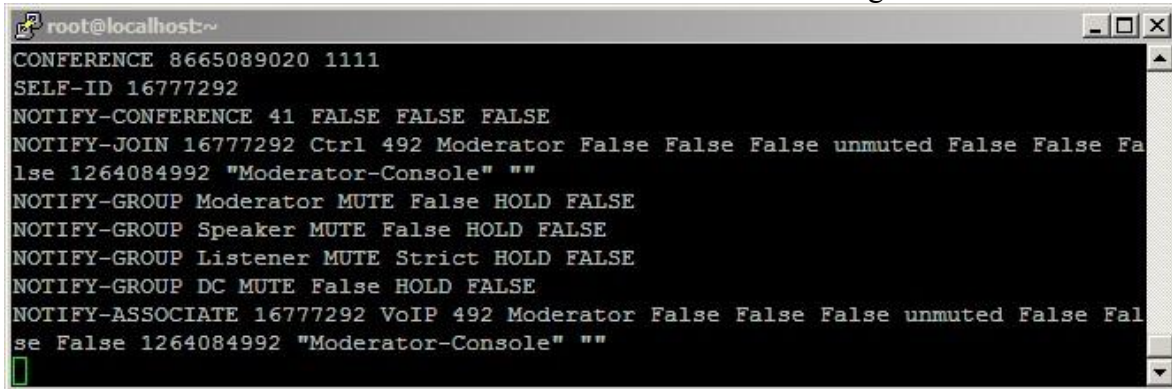
If you successfully connected to the conference you will receive the message:

```
SELF-ID <sessionId>
```

(for instance, `SELF-ID 16777292`); 16777292 is the session identifier of the call.

If connection to the conference has been successful all SIP messages will be sent to the client until the conference or the call is over. I.e. you will get all notification messages (see section: Notification Format), including active speaker notifications (see section: Active Speaker Notifications in Real Time Protocol and section: Active Speaker Notifications in ASD).

ASD call to the conference and initial notifications are shown on Figure 4.



```
root@localhost:~
CONFERENCE 8665089020 1111
SELF-ID 16777292
NOTIFY-CONFERENCE 41 FALSE FALSE FALSE
NOTIFY-JOIN 16777292 Ctrl 492 Moderator False False False unmuted False False Fa
lse 1264084992 "Moderator-Console" ""
NOTIFY-GROUP Moderator MUTE False HOLD FALSE
NOTIFY-GROUP Speaker MUTE False HOLD FALSE
NOTIFY-GROUP Listener MUTE Strict HOLD FALSE
NOTIFY-GROUP DC MUTE False HOLD FALSE
NOTIFY-ASSOCIATE 16777292 VoIP 492 Moderator False False False unmuted False Fal
se False 1264084992 "Moderator-Console" ""
```

Figure 4: ASD Call to the Conference

If the connection to the conference is unsuccessful, you will receive the message:

```
DROPPED
```

and the connection will be closed by foreign host.

Active Speaker Notifications in ASD

ASD receives active speaker notifications of the connected conference in RTP stream and these notifications are being transformed into notifications in the following format:

```
AS <sessionId> <level1> [, <sessionId2> <level2> [,
<sessionId3> <level3> [, <sessionId4> <level4>]]]
```

- `<sessionId1> ... <sessionId4>` – the tokens indicate the sessionId of the connections;
- `<level1> ... <level4>` – the tokens indicate the sound volume level, they could be 0 (silence) or from 1 (quietest) till 15 (loudest).

Note:

if sound level in the conference is 0 (silence) the first token will be equal to 268435455, i.e. the returned notification will be:

```
AS 268435455 0
```

Otherwise the notification will contain non-silent sessions' identifiers and their sound levels from 1 till 15 (for four loudest sessions as maximum) for the connected conference.

Real Time Commands Requests in ASD

Using ASD you can execute any of the real time bridge request commands. To do so you should run:

```
RT <RT request command>
```

- `<RT request command>` – the parameter denotes any of WYDE bridge request command described in section: Request Format.

After you send the request you will receive the response message that contains the identifier of your request and the response code (error code) as described in section: Response Format.

When the requested command is executed you will receive the notifications message, see section: Notification Format for details.

For instance, if you would like to mute the specific session you can run the command:

```
RT MUTE 1 strict 16777293
```

where 1 – your request identifier, 16777293 – the session identifier you would like to mute.

As the response you will receive:

```
RESPONSE 1 0
```

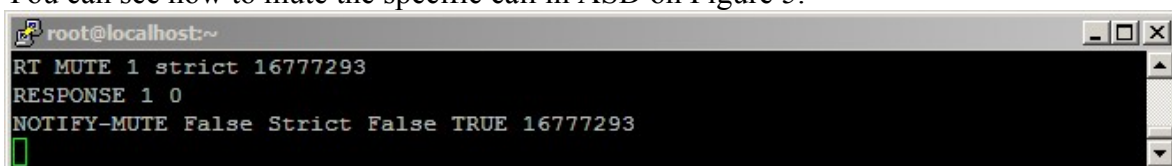
where 1 – your request identifier, 0 – the response code indicating success.

When your request is implemented you will receive the notification message:

```
NOTIFY-MUTE False Strict False TRUE 16777293
```

where 16777293 – the session identifier you are muting.

You can see how to mute the specific call in ASD on Figure 5.



```
root@localhost:~
RT MUTE 1 strict 16777293
RESPONSE 1 0
NOTIFY-MUTE False Strict False TRUE 16777293
```

Figure 5: Muting the Specific Call in ASD

Bundling Connections Using Audio Keys

Audio Key – is short identifier generated externally and provided to the WYDE bridge at the time of joining a conference.

- The WYDE bridge is capable of associating each connection with an audio key, which is a 32-bit unsigned integer, presented by participants at the time of joining the conference; however, it is not mandatory for the audio key to be presented at join time – the bridge is able to associate an audio key value of zero in cases where it is not presented at join time.
- The WYDE bridge is able to group a set of connections that have the same audio key into a single logical bundle; note: connections with zero audio key assigned by the bridge are not form a logical bundle.
- The WYDE bridge allows the audio key associated with a connection (PSTN or VoIP) to be changed.

The bundling connections are being used because a single end point may have multiple connections to the bridge. For instance an end point can use a PSTN line to speak and listen in the conference; but also it can use a separate connection to send and receive control data. Thus there is a one-to-many mapping between a given audio key and individual connection identifiers.

When the control call is being made on the bridge the audio key is being assigned to this call by *mf* (multi-frontend) service. This audio key is unique within the conference. When voice (PSTN) call is being made on the bridge the audio key is not assigned to this call, i.e. its audio key is 0. For any connected calls the control call receives notification NOTIFY-JOIN, this notification has the audio key token (see section: Notification Format for details); for voice calls it equals to 0, for control calls it is non-zero numeric value unique within the conference.

The audio key can be set (or changed) for the PSTN (voice) call using your telephone DTMF keypad (default sequence is #9<audio key>#). The audio key also can be changed for the control call using ASSOCIATE (Request to Change Audio Key of the Connection), see section: Request Format. In both cases the control call receives notification NOTIFY-ASSOCIATE, this notification has the audio key token (see section: Notification Format for details).

Bundling connections is working the same way regardless of what transport is being used – SIP transport or ASD transport. The logic and using approach is the same in both cases. If ASD transport is used the ASD connection receives generated audio key (in NOTIFY-JOIN) and it receives all other notifications whenever there is a change in the conference state. If SIP transport is used to receive generated audio key and all notifications of the conference the SIP INVITE must contain the following SIP Header Require:

```
Require: version,VoIP,RT
```

and the media attribute (m= line in SDP) must be the following:

```
m=application <source port> udp as
```

where <source port> – is the port where the client application receives RTP data.

As it already was told all calls with the same audio key are being joined into logical groups – bundles. If the calls have the same audio keys one of the calls in the bundle can manage another call in the same bundle. For instance if we have two calls with the same audio key: the first – control call and the second – voice call, the control call can manage the voice call which has the same audio key that the control call has.

Thus if you would like to group control and voice calls together the workflow should be the following:

- i) the control call is being made on the bridge and receives the audio key;
- ii) the voice call is being made on the bridge, the audio key of this call is zero;
- iii) the audio key of the control call should be told to the voice call owner;
- iv) the caller of the voice call should enter the received audio key on his phone using
#9<audio key># keys sequence;
- v) the control call receives notification with new audio key of the voice call that has been changed.

After that these calls are grouped together into a single logical bundle and the voice call can be managed by the control call.

Note that all these calls are different calls, they have their own session identifiers and they are represented by different CDR records. However the client's application can consider them as the same calls and show grouped information as the single record if it is necessary.

Chapter 3: Message Format Reference

Control messages exchanged between the bridge and the gateway/user are of 3 types: requests, responses, and notifications.

- The bridge supports receiving requests and sending responses / notifications on SIP using MESSAGE; the content-type of these messages is "text/plain".
- The control messages begin with a keyword which identifies the type of message. The keyword is followed by a set of tokens. The keyword and tokens are delimited by one or more white space characters (excluding newline).
Notifications can have multiple lines, with each line representing a single command or notification. The lines are delimited by a '\n' character.

These control messages are the same regardless of transport (SIP or ASD) used; see Chapter 2: RT Interface Transports Protocols for details.

When you would like to execute any command you should send the request message to the bridge. Once the request is received by the bridge, the response message is being generated and returned; the response message contains the identifier of your request and the response code (error code). Requested commands are being executed asynchronously; when the requested command or any other conference event is occurred the notification message will be sent by the bridge back to you.

Request Format

Following is the format of all commands that an end point can execute on the bridge. Each request is identified by a Keyword. All requests have a requestId (unique request identifier) that must be returned by the bridge in the response. All requests identify the target of the request either by the sessionId or by their role.

For any request commands text arguments that contain spaces should be either single quoted using apostrophe (') or double quoted ("). If the text argument also contains the same quotes that were used in the quotation they should be shielded (sheltered) using backslash (i.e. \' or \"). This rule is being applied for such arguments as `customName` of `SET-CUSTOMNAME`, `jobCode` of `SET-JOBCODE`, `filename` of `PLAY-FILE`, etc.



By default, all requests can be executed by the moderator. Participants can send requests to their own connections and to the connections of the same bundle (the same audio key) with them. Additional privilege information is discussed with each request, as applicable.

ASSOCIATE (Request to Change Audio Key of the Connection)

Format:

ASSOCIATE <requestId> <sessionId> [<Audio-Key>]

Assign (associate) new audio key for the specific session. If <Audio-Key> parameter omitted the audio key will be set to 0 – the call does not belong to any bundles.

Privilege:

Non-moderator roles can execute this request only on their own connection (but not on other connections in the bundle).



You can associate the audio key for the voice call only if there is the control call in the conference that already has the same audio key and the roles of both calls (the control call and the voice call) are the same.

CONF-DROP (Request to Drop the Conference)

Format:

CONF-DROP <requestId>

Privilege:

The request is permitted for moderator roles only.

DROP (Request to Drop the Connection)

Format:

DROP <requestId> <sessionId>

Privilege:

This request can be executed by non-moderator roles against connections in their own bundle. Bundles with Audio-Key=0 are excluded in these operations.

HOLD (Request to Change Hold Status of the Connection)Format:

HOLD <requestId> <True|False> <sessionId>

True indicates that the connection must be placed on hold; False indicates that the connection must be released from hold.

Privilege:

This request can be executed by non-moderator roles but only against connections in their own bundle. Bundles with Audio-Key=0 are excluded in these operations.

HOLD-GROUP (Request to Change Group Hold Policy)Format:

HOLD-GROUP <requestId> <True|False> <Moderator|Speaker|Listener>

True indicates that the role must be placed on hold; False indicates that the role must be released from hold.

Privilege:

The request is permitted for moderator roles only.

MUTE (Request to Mute the Connection)Format:

MUTE <requestId> <Strict|Relaxed|False> <sessionId>

Strict indicates that the connection must be placed in strict mute; Relaxed indicates relaxed mute and False indicates that the connection must be unmuted.

Privilege:

This request can only be executed by Moderators. If Moderators issue this against the connection in their own bundle, it is not treated as MUTE-SELF.

MUTE-GROUP (Request to Change Group Mute Policy)Format:

MUTE-GROUP <requestId> <Strict|Relaxed|False>
<Moderator|Speaker>

Strict indicates that the role must be placed in strict mute; Relaxed indicates relaxed mute and False indicates that the role must be unmuted.

Privilege:

The request is permitted for moderator roles only.

Exception:

MUTE-GROUP for muting a group of moderators excludes all connections that belong to the same bundle as the initiating moderator.

MUTE-GROUP <requestId> <False> <Listener> is not allowed (as this has the potential of un-muting of 1000 listeners - making the conference unusable).

MUTE-SELF (Request to Self-Mute the Connection)Format:

MUTE-SELF <requestId> <Strict|Relaxed|False> <sessionId>

Strict indicates that the connection must be placed in strict mute, Relaxed indicates relaxed mute and False indicates that the connection must be unmuted.

Privilege:

This request can be executed by non-moderator roles but only against connections in their own bundle. Moderators muting other connections must not use this request.

Note that, if a moderator issues a MUTE-SELF it will still not affect the Moderator mute field.

This request additionally requires that no IVR (“muted” or “unmuted”) be played on its execution.

OPERATOR_CALL_MOVE (Request to Move the Current User to other Conference)Format:

OPERATOR_CALL_MOVE <requestId> <newDIDNumber> <newAccessCode>
[<newRole: Host|Participant|Listener>]

The request moves the user that currently is talking to the operator to another conference: newDIDNumber indicates new (i.e. target) conference DNIS (DID) number where the call should be moved; newAccessCode indicates the access code that should be used to join to new (i.e. target) conference; optional newRole indicates the role that will be granted to the call when it joins to new conference (applicable only for call flows without authorization, for example CONF call flow). The same action can be performed by the operator conference hosts using default *5 on DTMF keypad of their phones.

Privilege:

The request is permitted for moderator control call only.

OPERATOR_DIALOUT (Request to Dialout from Operator Conference)Format:**OPERATOR_DIALOUT** <requestId> <peerNumber>

The request performs dialout from the operator conference: `peerNumber` denotes the phone number you wish to dial. The same action can be performed by the operator conference hosts using default *7 on DTMF keypad of their phones.

Privilege:

The request is permitted for moderator control call only.

OPERATOR_LISTEN (Operator Request to Listen the Conference)Format:**OPERATOR_LISTEN** <requestId> {True <ConfNumber> Direct|Shunt
<MuteMode: True|False>}|False

The request starts and stops other conference listening from the operator conference: `True` as the first parameter indicates that the listening should be started, `False` as the single parameter indicates that the listening should be stopped; `ConfNumber` – the number of the conference you wish to start listening (must be indicated if `True` option is specified); `Direct|Shunt` – `Direct` denotes that the operator should be directly connected to the requested conference (in this case only operator can hear the requested conference, the user connected to the operator can not hear that conference), `Shunt` denotes that between operator conference and requested conference is being made the shunt and both conferences can hear each other (operator and the connected user both can hear the requested conference); `MuteMode` – `True` denotes that the operator is muted, so he can only hear the specified conference and he is unable to talk, `False` denotes that the operator is not muted, so he can hear and talk in the specified conference. The same action can be performed by the operator conference hosts using default *4 on DTMF keypad of their phones, the operator also has the options to connect with current user (`Shunt` mode) and without current user (`Direct` mode).

Privilege:

The request is permitted for moderator control call only.

OPERATOR_MESSAGE (Operator Request to Control Operator Message)Format:**OPERATOR_MESSAGE** <requestId> {Start <Destination: queue|all>
[<message>]}|{Stop}

The request either initiates recording and playing back of the operator message to operator's queue and to the all users on the bridge or clears (i.e. switches off) the operator message:

- `Start` as the parameter indicates that the message should be recorded and/or activated:

- `<Destination: queue|all>` as the parameter indicates where the message should be played – `queue` indicates that the message should be sent to the users from the operator's queue, `all` indicates that the message should be sent to all users on the bridge;
- `<message>` as the parameter specifies prerecorded audio file that will be played to the callers, if the parameter is omitted the operator will be asked to leave his message using his voice call;
- ✓ if there is no voice call connected to the conference the command returns the error;
- ✓ if the operator voice call is connected to the operator conference and `<message>` parameter is omitted the operator will be asked to leave his message after the tone and press # when it is done, after the message has been recorded (and # was pressed) this message will be played back to the operator and he will have the options either press 1 to send the recorded message to the specified destination or press * to cancel (this request command works this way regardless there is or there is no previously recorded operator message);
- `Stop` as the parameter indicates that the operator message should be cleared, i.e. playing this message should be stopped.

The similar actions can be performed by the operator conference hosts using default *8 on DTMF keypad of their phones.

Prerecorded messages are being stored in the `$varlib_dir/sounds/operator-messages` folder; messages recorded by the operator are being temporary stored in the `$varlib_dir/recordings/operator-messages` folder.

Privilege:

The request is permitted for moderator control call only.

OPERATOR_MESSAGE_TO_CONFERENCE (Operator Request to Record and Play Message to the Conference)

Format:

```
OPERATOR_MESSAGE_TO_CONFERENCE <requestId> <ConfNumber>
  <role: Moderator|Speaker|Listener> [<message>]
```

The request initiates recording and/or playing back of the operator message to the user's conference callers of the specified roles:

- `<ConfNumber>` indicates the number of the user's conference where the operator message should be played; if 0 is specified as the target conference number the message will be recorded and played to the conference to which the operator is currently connected or to whose user the operator is currently talking;
- `<role: Moderator|Speaker|Listener>` specifies the callers' role to which the message should be played, if the message should be played to multiply roles they all could be specified separated by colon (:);
- `<message>` as the parameter specifies prerecorded audio file that will be played to the callers, if the parameter is omitted the operator will be asked to leave his message using his voice call:

- if there is no voice call connected to the conference the command returns the error `PermDenied`;
- if parameter `<ConfNumber>` is omitted, but the operator neither connected to the user's conference, nor talking to the user from the conference, the command returns the error `Invalid`;
- if the operator voice call is connected to the operator conference and the target user's conference is established the operator will be asked to leave his message after the tone and press # when it is done, after the message has been recorded (and # was pressed) this message will be played back to the operator and he will have the options either press 1 to send the recorded message to the conference or press * to cancel.

The similar actions can be performed by the operator conference hosts using default *9 on DTMF keypad of their phones.

Prerecorded messages are being stored in the `$varlib_dir/sounds/operator-messages` folder; messages recorded by the operator are being temporary stored in the `$varlib_dir/recordings/operator-messages` folder.

Privilege:

The request is permitted for moderator control call only.

OPERATOR_REJECT (Operator Request to Reject the User from the Queue)

Format:

OPERATOR_REJECT `<requestId>` `<confNumber>` `<sessionId>`

The request rejects the user from the operator queue, i.e. the conversion with the user will be refused and the user will be removed from the operator queue. The argument `<confNumber>` denotes the number of the conference where the caller came from (0 for IVR calls); the argument `<sessionId>` denotes the caller session identifier.

Privilege:

The request is permitted for moderator control call only.

OPERATOR_SCAN (Operator Request to Scan Conferences)

Format:

OPERATOR_SCAN `<requestId>` `True|False`

The request starts and stops conferences monitoring (surveillance call), i.e. the operator will switch between the conferences and hear each conference 30 seconds: `True` denotes that the monitoring should be started; `False` denotes that the monitoring should be stopped. The same action can be performed by the operator conference hosts using default *1 on DTMF keypad of their phones.

Privilege:

The request is permitted for moderator control call only.

OPERATOR_TALK (Operator Request to Talk to the User)Format:

```
OPERATOR_TALK <requestId> {True [<confNumber>
    <sessionId>]} | False
```

The request starts and stops talking to the user either from the operator queue or from the user conference, even if the user did not request operator assistance.

- **True** denotes that the talking should be started:
 - if <confNumber> <sessionId> arguments are omitted the first caller from the operator queue will be taken;
 - if the arguments <confNumber> <sessionId> are specified the specific caller either from the operator queue or from the user conference will be taken (<confNumber> – the conference number where the caller that should be taken came from (0 for IVR calls), <sessionId> – the caller session identifier);
 - ✓ to implement this request first the system looks into operator's queue trying to find out the user with <confNumber> <sessionId> specified, if such user not found in the queue next the system searches such user with <sessionId> specified in the active conference with the number <confNumber>;
- **False** denotes that the talking should be stopped.

To connect to the next user, i.e. receive the call from the user from the operator queue, the operator conference hosts can use DTMF keypad of their phones and press default *2 to start talking; *3 can be used to stop talking to user.

Privilege:

The request is permitted for moderator control call only.

PLAY-FILE (Request to Control the Audio File Playback)Format:

```
PLAY-FILE <requestId> <command> [<arguments>]
```

Performs the management (control) of the audio file playback in the conference.

Commands and arguments:

- **assign** <dir> <filename> – assign the file for the playback:
 - <dir> – either *record* for the conference recorded files (i.e. previous this conference recordings) or *upload* for the uploaded files (i.e. the files uploaded via web);
 - <filename> – the audio file name without extension, this file should be in the conference *recording* folder (usually */usr/local/DNCA/var/recordings/* folder) subfolder, either *record* subfolder or *upload* subfolder for the specific conference; if *filename* contains spaces use single or double quotes to define this argument as described in the beginning of this section;
- **seek** <offset> <whence> – seek the audio file playback indicator (pointer) on *offset* seconds relative to the parameter *whence*:
 - 0 – starting from the beginning of the file;

- 1 – starting from the current position in the file;
- 2 – starting from the end of the file;
- `start` – start the playback from the current position;
- `stop` – stop the playback.

Privilege:

The request is permitted for moderator control call only.

QA-CLEAR-QUEUE (Request to Clear Q&A Queue)Format:

QA-CLEAR-QUEUE <requestId>

Clears Q&A questions queue for the active Q&A session. It is similar to default *05 pressed on the DTMF keypad.

Privilege:

The request is permitted for moderator roles only.

QA-MODE (Request to Start/Stop Q&A Mode)Format:

QA-MODE <requestId> <True|False>

Starts and stops Q&A mode for the conference. It is similar to default *01/*03 pressed on the DTMF keypad: `True` indicates that the Q&A mode should be started; `False` indicates that the Q&A mode should be stopped.

Privilege:

The request is permitted for moderator roles only.

QA-MUTE (Request to Mute/Unmute the Current Questioner)Format:

QA-MUTE <requestId> <True|False>

Mutes (last parameter: `True`) or un-mutes (last parameter: `False`) the current questioner in Q&A session. It is similar to default *04 pressed on the DTMF keypad.

Privilege:

The request is permitted for moderator roles only.

QA-REQUEST (Request to Speak in Q&A Sessions)Format:

QA-REQUEST <requestId> <sessionId> <True|False>

Sends the request to speak (to request question) when Q&A session is started (last parameter: `True`) or cancels the request (last parameter: `False`). It is similar to default *6 pressed on the DTMF keypad.

Privilege:

This request can be executed by non-moderator roles but only against connections in their own bundle.

QA-TALK (Request to Allow/Disallow Speaking)

Format:

QA-TALK <requestId> <sessionId> <True|False>

Allows (last parameter: `True`) or disallows (last parameter: `False`) speaking (asking the questions) for the specific call identified by <sessionId>.

Privilege:

The request is permitted for moderator roles only.

QA-TALK-NEXT (Request to Move to the Next Questioner)

Format:

QA-TALK-NEXT <requestId>

Moves to the next questioner in Q&A queue when Q&A session is started. It is similar to default *02 pressed on the DTMF keypad. If the active questioner exists he will be muted; the next questioner in the queue will then be unmuted to ask his question.

Privilege:

The request is permitted for moderator roles only.

RECORDING (Request to Start/Stop the Conference Recording)

Format:

RECORDING <requestId> <True|False>

Starts (last parameter: `True`) or stops (last parameter: `False`) the conference recording.

Privilege:

The request is permitted for moderator roles only.

SECURE (Request to Secure/Unsecure the Conference)

Format:

SECURE <requestId> <True|False>

Request to make the conference secure or unsecure. When the conference is secured no other participants can join to this conference. `True` indicates that the conference must be secured; `False` indicates that the conference must be unsecured.

Privilege:

The request is permitted for moderator roles only.

SET-CUSTOMNAME (Request to Change Custom Name of the Call)Format:

SET-CUSTOMNAME <requestId> <sessionId> <customName>

Request to change (set) the custom name of the specified call identified by <sessionId>. If customName contains spaces use single or double quotes to define this argument as described in the beginning of this section.

Privilege:

This request can be executed by non-moderator roles only on their own connection (but not on other connections in the bundle). Moderators can change the custom name for any call in the conference.

SET-GAIN (Request to Change Volume Level of the Call)Format:

SET-GAIN <requestId> <value> <sessionId>

The request to change (set) the volume level of the call. The <value> token could be from -10 till 10 or 255; it indicates what volume level should be set for the call, 255 denotes that the volume level is being automatically generated by the backend.

Privilege:

This request can be executed by non-moderator roles but only against connections in their own bundle. Moderators can change the volume level for any call in the conference.

SET-JOBCODE (Request to Change Conference Job Code)Format:

SET-JOBCODE <requestId> <jobCode>

Changes the job code for the conference. If jobCode contains spaces use single or double quotes to define this argument as described in the beginning of this section.

Privilege:

The request is permitted for moderator roles only.

SET-ROLE (Request to Change Role of the Connection)Format:

SET-ROLE <requestId> <sessionId>
<newRole: Moderator|Speaker|Listener>

Privilege:

The request is permitted for moderator roles only. This request is allowed as long as the correct new role is specified.

START-BROADCAST (Start Broadcast)Format:

START-BROADCAST <requestId>

On execution of this command all listeners are being released from hold.

Privilege:

The request is permitted for moderator roles only.

Note:

A NOTIFY-GROUP message is being sent to moderators on execution of this command as this command results in changing the hold status of the listener group.

Response Format

All requests from the end point require a response from the bridge. The following format is being followed for all responses.

Format:

RESPONSE <requestId> <ResponseCode> [<ResponseMessage>]

Responses are identified by the keyword “RESPONSE”. The `requestId` that was given in the Request is being sent back in the response. The Response Code could be one of the following:

- 0 – Indicating success
- 1 – Indicating a failure due to insufficient privilege to execute command [Permission Denied]
- 2 – Indicating a failure because the connection was not found [Connection Not Found]
- 3 – Indicating server error
- 4 – Indicating that wrong input arguments were transferred to the request command [Invalid Input]
- 5 – Indicating that exception occurred during the request command execution [Exception]

If the exception occurred (`ResponseCode=5`) the Response Message could be returned explaining the reason why the exception occurred.

For example the `RESPONSE` output could be the following:

```
RESPONSE 44 5 There are no more users in the queue.
```

where 44 – the `requestId`, 5 – the response code [Exception], *There are no more users in the queue.* – the textual reason of the exception.

Notification Format

Notifications are sent to conference users whenever there is a change in the conference state.

The notifications messages must not be larger than 1300 bytes or be at least 200 bytes less than the path MTU. In cases where the notifications exceed this size limit, the bridge fragments the ordered set of notifications and sends each fragment as a separate notification to the gateway (so the large messages are split into smaller messages).

When the control call connects to any conference it receives all notifications that allow retrieving the state of the connected conference. For example if the control call connects to operator conference it could receive operator's notifications in the following order:

- NOTIFY-OPERATOR-ENGAGE True . . . – if there is the user engaged by the operator;
- NOTIFY-OPERATOR-TALK True . . . – if the operator talks to the user when the user requested operator assistance in waiting mode, i.e. default *01 keys on DTMF keypad were used;
- NOTIFY-OPERATOR-CONNECT True . . . – if the operator connected to the user's conference after the user has requested operator assistance in non-waiting mode, i.e. default *02 keys on DTMF keypad were used;
- NOTIFY-OPERATOR-SCAN True – if conference scanning mode is switched on for the operator;
- NOTIFY-OPERATOR-SCAN-ACTIVE True <confNumber> – if the scanner is listening the specific conference;
- NOTIFY-OPERATOR-QUEUE Add . . . – for each session (call) from the operator queue, if the operator queue is not empty.

NOTIFY-ASSOCIATE

The bridge sends this notification in response of ASSOCIATE request.

Format:

```
NOTIFY-ASSOCIATE <sessionId> <carrier: VoIP|PSTN> <audiokey>
<role: Moderator|Speaker|Listener>
<muteSelf: strict|relaxed|False>
<muteModer: strict|relaxed|False>
<muteQA: strict|relaxed|False> <muteActual: True|False>
<holdSelf: True|False> <holdModer: True|False>
<holdActual: True|False> <createTime>
<callerNumberDoubleQuoted> <customNameDoubleQuoted> <gain>
```

- <sessionId> – the token indicates the sessionId of the connection;
- <carrier: VoIP|PSTN> – the token indicates the carrier type: either VoIP or PSTN;

- `<audioKey>` – the token indicates the audio-key associated with the connection; the bridge set it to zero if no audio key was provided by the connection at join time;
- `<role: Moderator|Speaker|Listener>` – the token indicates the role: Moderator, Speaker or Listener;
- `<muteSelf: strict|relaxed|False>` – the token indicates the self mute state of the connection, i.e. the status of muting the call made by caller;
- `<muteModer: strict|relaxed|False>` – the token indicates the moderator mute state of the connection, i.e. the status of muting the call made by moderator;
- `<muteQA: strict|relaxed|False>` – the token indicates the Q&A mute state of the connection, i.e. the status of Q&A mode of the call;
- `<muteActual: True|False>` – the token indicates the actual mute state of the connection: True if muted, False if un-muted;
- `<holdSelf: True|False>` – the token indicates the self hold state of the connection: is the connection placed on hold or not by caller;
- `<holdModer: True|False>` – the token indicates the moderator hold state of the connection: is the connection moderator placed on hold or not by moderator;
- `<holdActual: True|False>` – the token indicates the actual hold state of the connection: is the entire connection placed on hold or not;
- `<createTime>` – the token indicates session (call) creation time in seconds starting from 1/1/1970;
- `<callerNumberDoubleQuoted>` – the token indicates the caller number in double quotes ("");
- `<customNameDoubleQuoted>` – the token indicates the custom name of the caller in double quotes ("");
- `<gain>` – the token indicates the volume level of the call (it could be from -10 till 10 or 255 if the volume level is being set automatically by the backend).

NOTIFY-CONFERENCE

The keyword identifying the notification is NOTIFY-CONFERENCE. Each NOTIFY-CONFERENCE keyword is followed by tokens indicating the state of the conference.

Format:

```
NOTIFY-CONFERENCE <confNumber> <confId>
<lockStatus: True|False> <recStatus: True|False>
<QAStatus: True|False> <createTime>
```

- `<confNumber>` – the token indicates the number of the conference;
- `<confId>` – the token indicates the conference identifier;
- `<lockStatus: True|False>` – the token indicates is the conference locked (secured), i.e. can or cannot other participants join to the conference;
- `<recStatus: True|False>` – the token indicates is the conference being recorded or not;
- `<QAStatus: True|False>` – the token indicates is Q&A session started for the conference or not;

- `<createTime>` – the token indicates conference creation time in seconds starting from 1/1/1970.

NOTIFY-DROP

The bridge sends an update of connections that were cleaned up, connections that left the conference, connections that were dropped using a NOTIFY-DROP update.

Format:

NOTIFY-DROP `<sessionId>`

- `<sessionId>` – the token indicates the sessionId of the connection.

NOTIFY-GROUP

The bridge sends an update about changes to the policy table (at present this notification is being sent only to the moderator group):

Format:

NOTIFY-GROUP `<role: Moderator|Speaker|Listener> MUTE
<mute: STRICT|RELAXED|False> HOLD <hold: True|False>`

- `<role: Moderator|Speaker|Listener>` – the token indicates the role: Moderator, Speaker or Listener;
- `<mute: STRICT|RELAXED|False>` – the token indicates the mute state of the group;
- `<hold: True|False>` – the token indicates the hold state of the group: is the group placed on hold or not.

Note:

The NOTIFY-GROUP message is being sent to moderators along with the initial set of NOTIFY-JOIN messages too.

NOTIFY-HOLD

Format:

NOTIFY-HOLD `<holdSelf: True|False>
<holdModerator: True|False> <holdActual: True|False>
<sessionId>`

- `<holdSelf: True|False>` – the token indicates the self hold state of the connection: is the connection placed on hold or not by caller;
- `<holdModerator: True|False>` – the token indicates the self hold state of the connection: is the connection placed on hold or not by moderator;
- `<holdActual: True|False>` – the token indicates the actual hold state of the connection: is the entire connection placed on hold or not;
- `<sessionId>` – the token indicates the sessionId of the connection.

NOTIFY-JOIN

The bridge sends a notification to VoIP participants (immediately after they join) about the current state of the conference. The keyword identifying the notification is NOTIFY-JOIN. Each NOTIFY-JOIN keyword is followed by tokens indicating the state of each connection.

Format:

```
NOTIFY-JOIN <sessionId> <carrier: VoIP|PSTN> <audioKey>
<role: Moderator|Speaker|Listener>
<muteSelf: strict|relaxed|False>
<muteModer: strict|relaxed|False>
<muteQA: strict|relaxed|False> <muteActual: True|False>
<holdSelf: True|False> <holdModer: True|False>
<holdActual: True|False> <createTime>
<callerNumberDoubleQuoted> <customNameDoubleQuoted> <gain>
```

- <sessionId> – the token indicates the sessionId of the connection;
- <carrier: VoIP|PSTN> – the token indicates the carrier type: either VoIP or PSTN;
- <audioKey> – the token indicates the audio-key associated with the connection; the bridge set it to zero if no audio key was provided by the connection at join time;
- <role: Moderator|Speaker|Listener> – the token indicates the role: Moderator, Speaker or Listener;
- <muteSelf: strict|relaxed|False> – the token indicates the self mute state of the connection, i.e. the status of muting the call made by caller;
- <muteModer: strict|relaxed|False> – the token indicates the moderator mute state of the connection, i.e. the status of muting the call made by moderator;
- <muteQA: strict|relaxed|False> – the token indicates the Q&A mute state of the connection, i.e. the status of Q&A mode of the call;
- <muteActual: True|False> – the token indicates the actual mute state of the connection: True if muted, False if un-muted;
- <holdSelf: True|False> – the token indicates the self hold state of the connection: is the connection placed on hold or not by caller;
- <holdModer: True|False> – the token indicates the moderator hold state of the connection: is the connection moderator placed on hold or not by moderator;
- <holdActual: True|False> – the token indicates the actual hold state of the connection: is the entire connection placed on hold or not;
- <createTime> – the token indicates session (call) creation time in seconds starting from 1/1/1970;
- <callerNumberDoubleQuoted> – the token indicates the caller number in double quotes ("");
- <customNameDoubleQuoted> – the token indicates the custom name of the caller in double quotes ("");

- `<gain>` – the token indicates the volume level of the call (it could be from `-10` till `10` or `255` if the volume level is being set automatically by the backend).

The initial state updates a `NOTIFY-JOIN` line for each connection.

The bridge sends a message about participants joining the conference to other participants in the conference using the same `NOTIFY-JOIN` format.

NOTIFY-LOCK

Format:

NOTIFY-LOCK `<lockStatus: True|False>`

- `<lockStatus: True|False>` – the token indicates is the conference locked (secured), i.e. can or cannot other participants join to the conference.

NOTIFY-MUTE

The bridge sends an update when the mute status or audio key changes.

Format:

NOTIFY-MUTE `<muteSelf: strict|relaxed|False>`

`<muteModer: strict|relaxed|False>`

`<muteQA: strict|relaxed|False>` `<muteActual: True|False>`

`<sessionId>`

- `<muteSelf: strict|relaxed|False>` – the token indicates the self mute state of the connection, i.e. the status of muting the call made by caller;
- `<muteModer: strict|relaxed|False>` – the token indicates the moderator mute state of the connection, i.e. the status of muting the call made by moderator;
- `<muteQA: strict|relaxed|False>` – the token indicates the Q&A mute state of the connection, i.e. the status of Q&A mode of the call;
- `<muteActual: True|False>` – the token indicates the actual mute state: `True` if muted, `False` if un-muted;
- `<sessionId>` – the token indicates the `sessionId` of the connection.

NOTIFY-OPERATOR-CONNECT

The bridge sends an update when the operator is being connected to another conference, for instance when the request `OPERATOR_LISTEN` (Operator Request to Listen the Conference) is implemented.

Format:

NOTIFY-OPERATOR-CONNECT `{True <confNumber>`

`<Mode: Direct|Shunt>` `<Mute: True|False>}|False`

- `True` – in the first token indicates that the operator is being connected to the conference, `False` – in the first token indicates that the operator is being disconnected from the conference;

- `<confNumber>` – the token indicates the number of the conference the operator connected and started listening;
- `<Mode: Direct|Shunt>` – the token indicates the connection mode: `Direct` denotes that the operator is being directly connected to the requested conference (in this case only operator can hear the requested conference, the user connected to the operator can not hear that conference), `Shunt` denotes that between operator conference and requested conference is being made the shunt and both conferences can hear each other (operator and the connected user both can hear the requested conference);
- `<Mute: True|False>` – the token indicates is the operator muted or not: `True` denotes that the operator is muted, so he can only hear the specified conference and he is unable to talk, `False` denotes that the operator is not muted, so he can hear and talk in the specified conference.

NOTIFY-OPERATOR-ENGAGE

The bridge sends an update when the operator engages conversation with user and when the operator disengages this conversation. When the operator engages the user first the bridge sends this notification **NOTIFY-OPERATOR-ENGAGE** `True` . . . and next the bridge either sends **NOTIFY-OPERATOR-TALK** `True` . . . notification if the user requested operator assistance in waiting mode (i.e. default *01 keys on DTMF keypad were used) or came from IVR, or sends **NOTIFY-OPERATOR-CONNECT** `True` . . . notification if the user requested operator assistance in non-waiting mode (i.e. using default *02 keys on DTMF keypad). When the operator disengages the user first the bridge either sends **NOTIFY-OPERATOR-TALK** `False` notification (if the user previously requested operator assistance in waiting mode or came from IVR) or sends **NOTIFY-OPERATOR-CONNECT** `False` notification (if the user previously requested operator assistance in non-waiting mode keys) and next the bridge sends this notification **NOTIFY-OPERATOR-ENGAGE** `False`.

Format:

NOTIFY-OPERATOR-ENGAGE {`True` `<confNumber>` `<sessionId>`}|`False`

- `True` – in the first token indicates that the operator engages conversation with user, `False` – in the first token indicates that the operator disengages conversation with user;
- `<confNumber>` – the token indicates the number of the conference where the user is came from, if the user conference is not defined (for instance when the user came to the operator queue after unsuccessful attempts to enter access codes) the conference number in this token is equal 0;
- `<sessionId>` – the token indicates the `sessionId` of the call that was engaged by the operator.

NOTIFY-OPERATOR-MESSAGE

The bridge sends this notification either when the operator message was activated or when the operator message was deactivated (cleared). If the operator message was activated or

deactivated this notification is being sent to all operator control calls connected to the bridge. In addition when new operator control call connects to the bridge and there is the active operator message, this control call also receives this notification informing that the operator message is active.

Format:

NOTIFY-OPERATOR-MESSAGE {True <Destination: queue|all>
<msgType: om|omc>:<msgName> <opConfNumber>}|False

- True|False – the token indicates that the operator message is recorded and/or activated (True) or that the operator message is deactivated (False);
- <Destination: queue|all> – the token is being sent only if the first token is True, and it indicates either the active operator message is being played to the users from the operator's queue (queue) or it is being played to the all users on the bridge (all);
- <msgType: om|omc>:<msgName> – the token is being sent only if the first token is True, and it indicates the operator message type and file name:
 - om – indicates that the prerecorded file has been used;
 - omc – indicates that the operator recorded the message using his voice call;
 - msgName – indicates either prerecorded message file name from the *\$varlib_dir/sounds/operator-messages* folder (together with om token) or temporary recorded operator message file name from the *\$varlib_dir/recordings/operator-messages* folder (together with omc token);
- <opConfNumber> – the token is being sent only if the first token is True, and it indicates the operator conference number where the message was recorded.

NOTIFY-OPERATOR-MESSAGE-TO-CONFERENCE

The bridge sends this notification when the operator message to user's conference was activated. If the operator message to conference was activated this notification is being sent to all operator control calls connected to the bridge.

Format:

NOTIFY-OPERATOR-MESSAGE-TO-CONFERENCE <confNumber>
<role: Moderator|Speaker|Listener>
<msgType: om|omc>:<msgName> <opConfNumber>

- <confNumber> – the token indicates the number of the user's conference where the operator message is being played;
- <role: Moderator|Speaker|Listener> – the token indicates the role: Moderator, Speaker or Listener to which the operator message is being played;
- <msgType: om|omc>:<msgName> – the token indicates the operator message type and file name:
 - om – indicates that the prerecorded file has been used;
 - omc – indicates that the operator recorded the message using his voice call;

- `msgName` – indicates either prerecorded message file name from the `$varlib_dir/sounds/operator-messages` folder (together with `om` token) or temporary recorded operator message file name from the `$varlib_dir/recordings/operator-messages` folder (together with `omc` token);
- `<opConfNumber>` – the token indicates the operator conference number where the message was recorded.

NOTIFY-OPERATOR-QUEUE

The bridge sends this notification when either the user was added to the operator queue or when the user was removed from the operator queue.

Format:

```
NOTIFY-OPERATOR-QUEUE {ADD <confNumber> <sessionId> <flag>} |
{DEL <confNumber> <sessionId>}
```

- `{ADD|DEL}` – the token indicates was the user placed into the operator queue (`ADD`) or was the user removed from the operator queue (`DEL`);
- `<confNumber>` – the token indicates the number of the conference where the user is came from, if the user conference is not defined (for instance when the user came to the operator queue after unsuccessful attempts to enter access codes) the conference number in this token is equal 0;
- `<sessionId>` – the token indicates the `sessionId` of the call that was added to or removed from the operator queue;
- `<flag>` – the token value `False` indicates that the user is waiting operator assistance in the operator queue (i.e. the user has requested operator assistance in waiting mode after default *01 keys were pressed), the token value `True` indicates that the request to operator assistance was post by the user (i.e. the user has requested operator assistance in non-waiting mode after default *02 keys were pressed) and he is waiting operator in his own conference.

NOTIFY-OPERATOR-SCAN

The bridge sends an update when the operator started or stopped the conference monitoring, i.e. surveillance call.

Format:

```
NOTIFY-OPERATOR-SCAN True|False
```

- `True|False` – the token indicates was the conference scanning started (`True`) or stopped (`False`).

NOTIFY-OPERATOR-SCAN-ACTIVE

The bridge sends an update when the operator connected or disconnected to/from the specific conference when conference monitoring (surveillance call) is started.

Format:**NOTIFY-OPERATOR-SCAN-ACTIVE** {True <confNumber>}|False

- True|False – the token indicates that the operator was connected to the conference (True) or disconnected from the conference (False);
- <confNumber> – the token indicates the number of the specific conference the operator connected during the scanning process.

NOTIFY-OPERATOR-TALK

The bridge sends an update when the operator started or stopped talking to the user from his queue.

Format:**NOTIFY-OPERATOR-TALK** {True <confNumber> <sessionId>}|False

- True|False – the token indicates that the operator is started talking to the user from his queue (True) or stopped talking to the user (False);
- <confNumber> – the token indicates the number of the conference where the user is came from, if the user conference is not defined (for instance when the user came to the operator queue after unsuccessful attempts to enter access codes) the conference number in this token is equal 0;
- <sessionId> – the token indicates the sessionId of the call the operator is talking with.

NOTIFY-QA_MODEFormat:**NOTIFY-QA_MODE** <QAStatus: True|False>

- <QAStatus: True|False> – the token indicates is Q&A session started for the conference.

NOTIFY-QA_STATUSFormat:**NOTIFY-QA_STATUS** <sessionId> <QARequest: True|False>
<QATalk: True|False> <QARequestTime: True|False>

- <sessionId> – the token indicates the sessionId of the connection;
- <QARequest: True|False> – the token indicates if the participant requested to talk when Q&A session is started;
- <QATalk: True|False> – the token indicates if the participant allowed to talk after his request to talk when Q&A session is started (when QATalk is True, QARequest is always True as well);
- <QARequestTime> – the token indicates Q&A request creation time in seconds starting from 1/1/1970.

NOTIFY-RECORDINGFormat:**NOTIFY-RECORDING** <recStatus: True|False>

- <recStatus: True|False> – the token indicates is the conference being recorded or not.

NOTIFY-SET_CUSTOMNAME

The following updates are sent when the caller's custom name has been changed:

Format:**NOTIFY-SET_CUSTOMNAME** <sessionId> <customNameDoubleQuoted>

- <sessionId> – the token indicates the sessionId of the connection;
- <customNameDoubleQuoted> – the token indicates the custom name of the caller in double quotes (").

NOTIFY-SET_GAIN

The following updates are sent when the caller's volume level has been changed (for instance when the request SET-GAIN (Request to Change Volume Level of the Call) is implemented):

Format:**NOTIFY-SET_GAIN** <sessionId> <gain>

- <sessionId> – the token indicates the sessionId of the connection;
- <gain> – the token indicates the volume level of the call (it could be from -10 till 10 or 255 if the volume level is being set automatically by the backend).

NOTIFY-SET-ROLE

The following updates are sent when the SET-ROLE request is implemented:

Format:**NOTIFY-SET-ROLE** <sessionId>
<role: Moderator|Speaker|Listener>

- <sessionId> – the token indicates the sessionId of the connection;
- <role: Moderator|Speaker|Listener> – the token indicates the role: Moderator, Speaker or Listener.

NOTIFY-TERMINATION

The following updates are sent when the call is being terminated by the bridge. This notification is being sent only to the call that will be dropped, and it will be sent prior to close the connection of the call.

Format:

NOTIFY-TERMINATION <sessionId> "<DisconnectReason>"

- <sessionId> – the token indicates the sessionId of the connection;
- <DisconnectReason> – the token indicates the reason why the call has been disconnected.

Appendix A: 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.