



Command Line Administration Interface – User Guide

(version 2.1)

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://www.wydevoice.com/support>

August 19, 2010

Symbols and Notations in this Manual

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



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

Figure	Denotes any illustration
Table	Denotes any table
Text	Denotes any text output
<i>Folder/File</i>	Denotes any folders (paths) or files names
commands	Denotes any commands, attributes and parameters

Table of Contents

Symbols and Notations in this Manual	3
Table of Contents	4
Tables List	11
Figures List	12
Chapter 1: Introduction	14
Assumed Skills	14
Architecture Overview	14
Administration Interfaces	15
Hardware	16
Voice Clients	16
Integration Adapters	16
Definitions, Acronyms and Abbreviations	16
Chapter 2: Command Line Interface	20
Prerequisites	21
Accessing the Command Line Administration Interface	21
Using the <code>wyde</code> Command Line Utility	21
Using the <code>mfi</code> Console	24
Subscriber User Management	27
Create a Subscriber	27
View/Modify a Subscriber	28
Delete a Subscriber	28
Create a Conference Account	29
Configuration of the Conferences	30
View/Modify a Conference Account	31
Delete a Conference Account	32
Override a Call Flow Attribute Value for a Conference	32
View Call Flow Attributes Values for a Conference	33
Delete a Conference Call Flow Attribute Redefinition	34
Call Flow and DNIS Management	35
Add a Call Flow	35
View a Call Flow	36
Delete a Call Flow	37
Set a Call Flow Attribute Value	37
View Call Flow Attributes Values	38
Reload All Call Flows	39
Update Call Flow Attributes Definition in a Database	39
Create a DNIS Association	40
View/Modify a DNIS Association	41
Create a DNIS Number Alias	41
View DNIS Number Aliases	42
Delete a DNIS Number Alias	43
Delete a DNIS Association	43
Override a Call Flow Attribute Value for a DNIS	44
View Call Flow Attributes Values for a DNIS	45

Delete a DNIS Call Flow Attribute Redefinition	45
Reload All DNISes Caches	46
Conferences and Calls Management	47
View Conferences and Calls in Progress.....	47
View Conferences in Progress.....	47
View Conference Calls in Progress.....	49
Show Calls that have not placed to Conferences.....	51
Calls Management using <i>mf</i> Console	52
Dropping Call Participants	52
Mute Call Participants	53
Placing Call Participants on Hold.....	53
Set Custom Name for a Call.....	54
Set Audio Key for a Call	54
Conferences Management using <i>mf</i> Console.....	55
Dropping a Conference.....	56
Conference Mute and Q&A Modes.....	56
Q&A Sessions	57
Placing a Conference on Hold.....	59
Broadcast a Conference for Listeners.....	60
Making a Conference Secure.....	61
Setting a Conference Job Code.....	61
Recording a Conference	62
Playing an Audio File to a Conference.....	62
Dialing another User.....	64
Move a Call to another Conference.....	65
Making a Shunt between Two Conferences	68
Polling.....	68
Scheduling	69
Conferences and Calls Management using <i>asterisk</i> Console	73
Using the IVR/ <i>asterisk</i> Console	73
View Conferences in Progress.....	74
View Conference Calls in Progress.....	75
Show Calls that have not placed to Conferences.....	76
Dropping Conference Call Participants.....	76
Show Session Events Queue	77
Show Node Statistics.....	77
Conferences and Calls Management using <i>mp</i> Console.....	78
Using the <i>mp</i> Console.....	78
Show Different <i>mp</i> Information	78
Start and Stop <i>mp</i> Console Timers	80
Dropping Boards and Calls	81
Restarting and Stopping <i>mp</i> Boards and Logs	81
Operator Conferences	83
Operator Conferences Management using <i>mf</i> Console	83
Samples and Use-Cases of Operator Conference Procedures	88
WYDE Bridge Administration	89
Monitoring.....	89

WYDE Bridge Settings Management.....	92
Bridge Configuration Changes	94
Dialout Settings Configuration	95
WYDE Bridge Configuration Save and Restore	96
Nodes Administration.....	97
Distributed Conferencing Administration	100
Peers Management.....	105
Calls Transferring.....	106
Audio Prompts Management.....	106
Licensing	108
Authorization Adapters and Methods.....	108
Add an Authorization Adapter	110
Delete an Authorization Adapter.....	111
View Authorization Adapters.....	111
Add an Authorization Method.....	112
Delete an Authorization Method	113
Modify an Authorization Method.....	113
View Authorization Methods	114
Sample of Authorization Adapters for LDAP and Radius	115
Billing.....	122
Add a Billing Adapter	127
Delete a Billing Adapter.....	128
Modify a Billing Adapter	129
View Billing Adapters.....	129
Add a Billing Rule.....	130
Delete a Billing Rule	131
Modify a Billing Rule.....	131
View Billing Rules	132
Samples of Billing Adapters.....	133
Database Administration	136
Chapter 3: Command Reference.....	138
wyde Command Reference.....	138
ast-status (Show WYDE <i>asterisk</i> Status)	138
auth-adapter-add (Add <i>auth</i> Adapter)	138
auth-adapter-del (Delete <i>auth</i> Adapter).....	138
auth-adapter-show (Show <i>auth</i> Adapters).....	138
auth-method-add (Add <i>auth</i> Method)	138
auth-method-del (Delete <i>auth</i> Method)	139
auth-method-set (Set <i>auth</i> Method).....	139
auth-method-show (Show <i>auth</i> Methods).....	139
billing-adapter-add (Add Billing Adapter)	139
billing-adapter-del (Delete Billing Adapter)	139
billing-adapter-set (Set Billing Adapter Properties).....	139
billing-adapter-show (Show Billing Adapters).....	140
billing-rule-add (Add Billing Rule).....	140
billing-rule-del (Delete Billing Rule).....	140

billing-rule-set (Set Billing Rule)	140
billing-rule-show (Show Billing Rules)	141
bridge-add (Add WYDE Bridge).....	141
bridge-del (Delete WYDE Bridge)	141
bridge-show (Show WYDE Bridges).....	141
callflow-add (Add Call Flow).....	141
callflow-attr-set (Set Call Flow Attribute)	141
callflow-attr-show (Show Call Flow Attributes)	141
callflow-attr-update-db (Update Call Flow Attributes Definition in a Database)	142
callflow-del (Delete Call Flow)	142
callflow-reload (Reload All Call Flows).....	142
callflow-show (Show Call Flows Table).....	142
conference-attr-del (Remove Conference Attribute Redefinition).....	142
conference-attr-set (Set Conference Attribute).....	142
conference-attr-show (Show Conference Attributes).....	142
config-restore (Restore WYDE Configuration)	143
config-save (Save WYDE Configuration).....	143
confuser-add (Add Conference User).....	143
confuser-del (Delete Conference User)	143
confuser-show (Show Conference Users Table).....	144
db (Connect to WYDE Main, i.e. <i>dnca</i> , Database)	144
db-bil (Connect to WYDE Billing, i.e. <i>dnca_calls</i> , Database).....	144
db-init (Initialize of WYDE Database)	144
db-patch (Apply Last Patches for Databases)	144
did-add (Add DNIS/DID Number)	144
did-alias-add (Add DNIS/DID Number Alias)	144
did-alias-del (Delete DNIS/DID Number Alias).....	145
did-alias-show (Show DNIS/DID Number Aliases)	145
did-alias-show-all (Show all DID Number Aliases)	145
did-attr-del (Remove DNIS/DID Attribute Redefinition)	145
did-attr-set (Set DNIS/DID Attribute).....	145
did-attr-show (Show DNIS's Attributes).....	145
did-del (Delete DNIS/DID Number).....	145
did-reload (Reload all DNISes/DIDs Caches)	146
did-show (Show DNISes/DIDs Table).....	146
drop-call (Drop Call).....	146
drop-conf (Drop Conference)	146
help (Show Help Page and Exit).....	146
ivr (Connect to IVR/ <i>asterisk</i> Console)	146
node-add (Add Node to the WYDE Bridge).....	147
node-del (Delete Node from the WYDE Bridge).....	147
node-set (Set Node Properties)	147

node-show (Show Nodes of WYDE Bridge)	147
register-license (Register License)	148
set-email (Change Email Address)	148
set-ip (Change IP Address)	148
settings-edit (Edit WYDE System Settings)	148
settings-show (Show WYDE Settings)	148
settings-update (Update WYDE Settings)	149
show-conf (Show Conference or Conferences List)	149
status (Show WYDE Status)	149
subscriber-add (Add Subscriber)	149
subscriber-del (Delete Subscriber)	149
subscriber-show (Show Subscribers Table)	150
transfer (Transfer Calls)	150
version (Show WYDE Version)	150
watch (Watch WYDE Status)	150
mf Console Command Reference	151
call-associate (Set Bundle for the Call)	151
call-custom-name (Set Custom Name for the Call)	151
call-drop (Drop Call in the Conference)	151
call-hold (Hold Call)	152
call-move (Move Call to Other Conference)	152
call-mute (Mute Call)	152
call-qa-request (Start/Stop Q&A Request for the Call)	152
call-qa-talk (Enable/Disable Q&A Session for the Call in the Queue)	153
callflow-reload (Reload Call Flows)	153
cmdcount-show (Display Values of <i>Command</i> Counters)	153
conf-broadcast (Start/Stop Broadcast Mode for Listeners)	155
conf-drop (Drop Conference)	155
conf-hold-group (Hold Group)	155
conf-jobcode (Set Job Code for the Conference)	155
conf-mute-group (Mute Group)	156
conf-play-file (Manage of Playing File to the Conference)	156
conf-polling (Conference Polling)	157
conf-qa-mode (Manage Q&A Sessions)	157
conf-qa-mute (Mute/Unmute Active Q&A Session)	157
conf-qa-talk (Enable Q&A Session for the First Call in the Queue)	158
conf-recording (Start/Stop Conference Recording)	158
conf-schedule-extend (Extend Scheduled Conference Duration)	158
conf-schedule-incsize (Resize Scheduled Conference Subscription)	158
conf-secure (Secure Conference)	158
conf-shunt (Make/Drop Shunt between Two Conferences)	159
confcount-show (Display Values of <i>confcount</i> Counters)	159
dc-show-bridges (Show Known DC Bridges)	159

dc-show-links (Show DC Links).....	160
dialout (Do Dialout).....	160
dialout-attr (Show Dialout Attributes for Specified Conference)	160
did-reload (Reload DID Entries).....	160
errcount-show (Display Values of <i>Error</i> Counters).....	160
freenumbers-show (Show Free Number Leases).....	161
help (Show Help for Console Commands).....	161
moh-reload (Reload Customer's Music-On-Hold Prompts)	161
node-reload (Reload <i>MF</i> Cluster Nodes List).....	161
node-show (Show <i>MF</i> Cluster Nodes List).....	161
op-call-move (Move User that Currently Talking with Operator to other Conference)	162
op-dialout (Initiate Dialout from Operator's Console).....	162
op-listen (Listen Conference)	162
op-queue (Display Operator Calls Queue).....	163
op-scan (Scan Conferences).....	163
op-show (Display Operators)	163
op-talk (Operator Talk to User)	164
partcount-show (Display Values of <i>partcount</i> Counters).....	164
peer-reload (Reload Peers).....	165
quit (Quit Console).....	165
set-log-level (Set Logger Level).....	165
settings-reload (Reload System Settings)	165
show (Show Conferences and Calls).....	165
transfer (Transfer Calls).....	166
welcomeprompt-reload (Reload Customer's Welcome Prompts).....	166
asterisk Console Command Reference.....	167
wyde drop session (Drop Session)	167
wyde show conferences (Show Active Conferences)	167
wyde show conference (Show Conference Members).....	167
wyde show session queue (Show Session Events Queue).....	167
wyde show sessions (Show Sessions not Attached to Conference)	167
wyde show statistic (Show Statistic).....	167
mp Console Command Reference	168
drop (Drop Boards and Calls)	168
kill (Stop Specific Timer or All Timers).....	168
restart (Restart <i>mp</i> Boards and Logs)	168
show (Show Different <i>mp</i> Statistics).....	169
stop (Stop <i>mp</i> Components)	169
timer (Start Timer Running <i>show</i> Command)	169
Appendix A: Code Samples	170
Authorization Adapters	170
Sample of Authorization Adapter for Windows Active Directory (WinLdap).....	170
Sample of Authorization Adapter for WYDE Radius (WYDERadius)	173

Billing Adapters.....	177
Sample of Billing Adapter for Windows PostgreSQL Database (WINPGSQL)	177
Sample of Billing Adapter for Microsoft SQL Database (MSSQL)	178
Appendix B: Support Resources	179
Support Documentation.....	179
Web Support.....	179
Telephone Support.....	179
Email Support.....	179

Tables List

Table 1: <i>wyde</i> Command Line Utility Available Commands.....	22
Table 2: <i>mf</i> Console Utility Available Commands.....	25
Table 3: Show Conferences Columns	48
Table 4: Show Conference Calls Columns.....	50
Table 5: Q&A Sessions Management Samples Using <i>mf</i> Console Commands	59
Table 6: <i>asterisk</i> Console Utility Available <i>wyde</i> Commands	74
Table 7: <i>mp</i> Console Utility Available Commands.....	78
Table 8: Operator Conference Management Samples Using <i>mf</i> Console Commands	87
Table 9: Active Directory Conference Accounts Data.....	116
Table 10: <i>CDR.log</i> File Data Structure	123
Table 11: Local <i>dnca_calls</i> Database <i>conferencedr</i> and <i>calls</i> Tables Data Structure.....	123
Table 12: Input Data Format for Billing Adapters	126

Figures List

Figure 1: The WYDE Bridge Architecture	15
Figure 2: <i>wyde help</i> Command Output.....	21
Figure 3: <i>wyde help version</i> and <i>wyde version</i> Commands Output Sample.....	24
Figure 4: <i>mf</i> Console, <i>help</i> Command Output.....	25
Figure 5: <i>mf</i> Console <i>help quit</i> and <i>quit</i> Commands Output Sample	26
Figure 6: <i>wyde help subscriber-add</i> and <i>wyde subscriber-add</i> Commands Output Sample	28
Figure 7: <i>wyde subscriber-show</i> Command Output Sample	28
Figure 8: <i>wyde help confuser-add</i> and <i>wyde confuser-add</i> Commands Output Sample	30
Figure 9: <i>wyde help confuser-show</i> and <i>wyde confuser-show</i> Commands Output Sample..	31
Figure 10: <i>wyde help conference-attr-set</i> and <i>wyde conference-attr-set</i> Commands Output Sample	33
Figure 11: <i>wyde conference-attr-show</i> Command Output Sample	34
Figure 12: <i>wyde help callflow-add</i> and <i>wyde callflow-add</i> Commands Output Sample.....	36
Figure 13: <i>wyde callflow-show</i> Command Output Sample	37
Figure 14: <i>wyde help callflow-attr-set</i> and <i>wyde callflow-attr-set</i> Commands Output Sample	38
Figure 15: <i>wyde callflow-attr-show</i> Command Output Sample	39
Figure 16: <i>wyde help did-add</i> and <i>wyde did-add</i> Commands Output Sample	41
Figure 17: <i>wyde did-show</i> Command Output Sample	41
Figure 18: <i>wyde help did-alias-add</i> and <i>wyde did-alias-add</i> Commands Output Sample ..	42
Figure 19: <i>wyde did-alias-show</i> Command Output Sample.....	43
Figure 20: <i>wyde help did-attr-set</i> and <i>wyde did-attr-set</i> Commands Output Sample	44
Figure 21: <i>wyde did-attr-show</i> Command Output Sample	45
Figure 22: <i>wyde help show-conf</i> and <i>wyde show-conf</i> Commands Output Sample	47
Figure 23: <i>mf</i> Console <i>help show</i> and <i>show</i> Commands Output Sample	48
Figure 24: <i>wyde show-conf</i> Commands Output Sample for the Specific Conferences	49
Figure 25: <i>mf</i> Console <i>show</i> Commands Output Sample for the Specific Conferences	50
Figure 26: <i>wyde show-conf</i> Command Output Sample for the Calls that have not placed to Conferences	51
Figure 27: <i>mf</i> Console <i>show</i> Command Output Sample for the Calls that have not placed to Conferences	52
Figure 28: <i>mf</i> Console <i>call-associate</i> Commands Output Sample.....	55
Figure 29: <i>mf</i> Console <i>conf-play-file</i> Commands Output Sample.....	63
Figure 30: <i>mf</i> Console <i>dialout-attr</i> and <i>dialout</i> Commands Output Sample.....	65
Figure 31: <i>mf</i> Console <i>call-move</i> Command Output Sample	67
Figure 32: Starting <i>asterisk</i> Console	73
Figure 33: <i>asterisk</i> Console, <i>help wyde</i> Command Output	74
Figure 34: <i>asterisk</i> Console <i>quit</i> Command Output Sample	74
Figure 35: <i>asterisk</i> Console <i>wyde show conferences</i> and its <i>help</i> Commands Output Sample	75
Figure 36: <i>asterisk</i> Console <i>wyde show conference</i> and its <i>help</i> Commands Output Sample	75
Figure 37: <i>asterisk</i> Console <i>wyde show sessions</i> and its <i>help</i> Commands Output Sample..	76

Figure 38: <i>asterisk</i> Console <i>wyde show session queue</i> and its <i>help</i> Commands Output Sample	77
Figure 39: <i>asterisk</i> Console <i>wyde show statistic</i> and its <i>help</i> Commands Output Sample ..	77
Figure 40: Starting <i>mp</i> console, Available Commands	78
Figure 41: <i>mp</i> Console <i>show</i> Commands Output Sample	79
Figure 42: <i>mp</i> Console <i>timer</i> and <i>kill</i> Commands Output Sample	80
Figure 43: <i>mp</i> Console <i>drop</i> Commands Output Sample	81
Figure 44: <i>mp</i> Console <i>restart</i> Commands Output Sample	82
Figure 45: <i>mf</i> Console <i>op-show</i> and <i>op-queue</i> Commands Output Sample	85
Figure 46: <i>mf</i> Console <i>freenumbers-show</i> Command Output Sample	90
Figure 47: <i>wyde ast-status</i> Command Output Sample	91
Figure 48: <i>wyde status</i> Command Output Sample	91
Figure 49: <i>wyde help settings-edit</i> and <i>wyde settings-edit</i> Commands Output Sample	92
Figure 50: <i>wyde help settings-show</i> and <i>wyde settings-show</i> Commands Output Sample ..	94
Figure 51: <i>wyde help config-save</i> and <i>wyde config-save</i> Commands Output Sample	96
Figure 52: <i>wyde node-show</i> and <i>mf</i> Console <i>node-show</i> Commands Output Sample	100
Figure 53: <i>mf</i> Console <i>show</i> Conference – Different Nodes Calls Command Output Sample	100
Figure 54: <i>bridges</i> Table – Distributed Conferences Configuration	103
Figure 55: Show Distributed Conference on <i>WYDE5</i> Bridge	104
Figure 56: Show Distributed Conference on <i>WYDE31</i> Bridge	104
Figure 57: <i>mf</i> Console <i>dc-show-bridges</i> Command Output Sample on <i>WYDE5</i> Bridge ...	104
Figure 58: <i>mf</i> Console <i>dc-show-bridges</i> Command Output Sample on <i>WYDE31</i> Bridge ..	105
Figure 59: <i>mf</i> Console <i>dc-show-links</i> Command Output Sample on <i>WYDE5</i> Bridge	105
Figure 60: <i>mf</i> Console <i>dc-show-links</i> Command Output Sample on <i>WYDE31</i> Bridge	105
Figure 61: Audio File Conversion using <i>transcoder.x</i> Utility Sample	107
Figure 62: Register New/Updated Licenses on the Bridge	108
Figure 63: <i>wyde help auth-adapter-add</i> and <i>wyde auth-adapter-add</i> Commands Output Sample	111
Figure 64: <i>wyde auth-adapter-show</i> Command Output Sample	112
Figure 65: <i>wyde help auth-method-add</i> and <i>wyde auth-method-add</i> Commands Output Sample	113
Figure 66: <i>wyde help auth-method-set</i> and <i>wyde auth-method-set</i> Commands Output Sample	114
Figure 67: <i>wyde auth-method-show</i> Command Output Sample	115
Figure 68: Active Directory Conference Accounts and Conference Numbers Data	115
Figure 69: <i>wyde help billing-adapter-add</i> and <i>wyde billing-adapter-add</i> Commands Output Sample	128
Figure 70: <i>wyde help billing-adapter-set</i> and <i>wyde billing-adapter-set</i> Commands Output Sample	129
Figure 71: <i>wyde billing-adapter-show</i> Command Output Sample	130
Figure 72: <i>wyde help billing-rule-add</i> and <i>wyde billing-rule-add</i> Commands Output Sample	131
Figure 73: <i>wyde help billing-rule-set</i> and <i>wyde billing-rule-set</i> Commands Output Sample	132
Figure 74: <i>wyde billing-rule-show</i> Command Output Sample	132

Chapter 1: Introduction

This is the Administration guide for the WYDE conferencing bridges (like SB-HD100, SB-HD1000, and SB-HD10000). Within this guide you will learn how to perform the basic day to day administration tasks for these units using command line interface.

Assumed Skills

This administration guide assumes you have a working knowledge of the following technologies and skills:

- PC usage
- System administration
- Linux/CentOS basics
- VOIP basics
- TCP/IP networking
- Web Administration Interface – User Guide (recommended)

Architecture Overview

The WYDE architecture is made up of both hardware as well as software services (as shown in Figure 1) that work together to provide the best carrier-class, wideband conferencing available.

WYDE services is not only turnkey software solution, it is the component that can be easily integrated into other products. The WYDE Bridge can be controlled either using web services or using real-time interface. Web services send requests to the bridge and receive information about status of the bridge. The real time interface makes call to the bridge using special client, perform SIP call to send and receive commands and exchange information about the conferences.

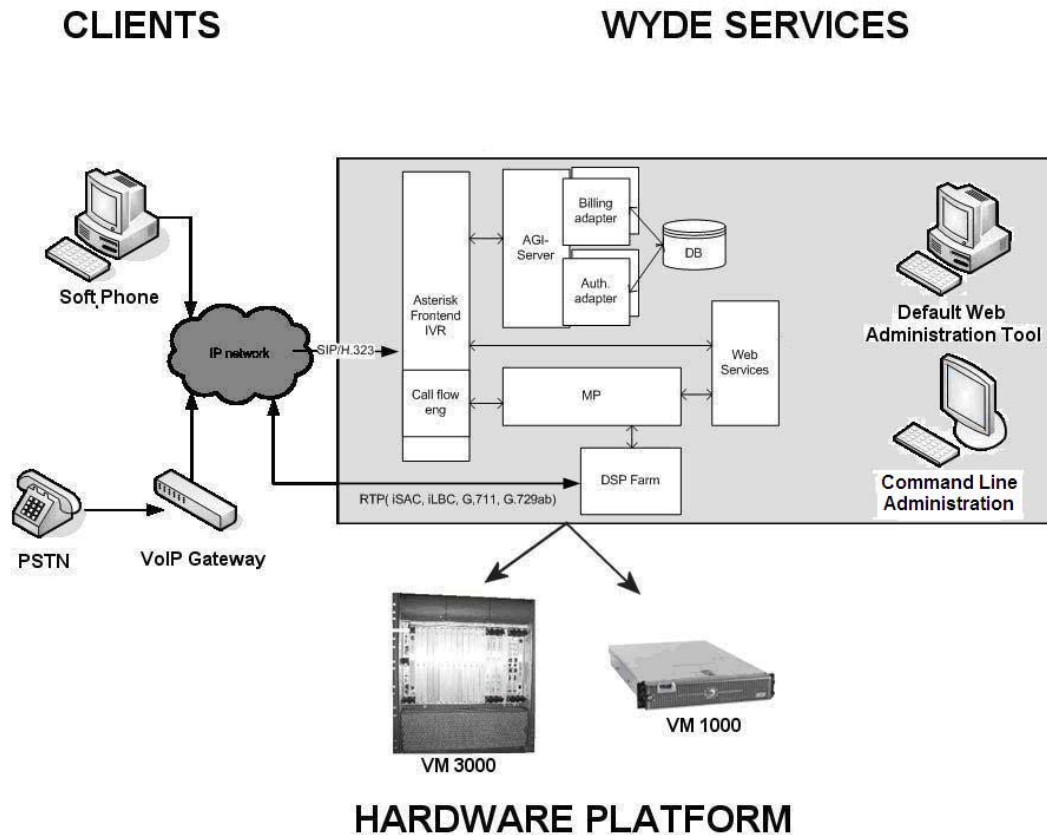


Figure 1: The WYDE Bridge Architecture

Administration Interfaces

There are two basic interfaces to the WYDE conference bridges that allow an administrator to make changes and administrate the operation of the system. These interfaces include:

- The WYDE Command Line Administration Interface
- The WYDE Web Administration Interface

Each interface uses different methods to administrate the various functions of the WYDE conference bridges. It should be noted that there are several places where you can only perform a function in one of the interfaces and there is no corresponding function in the other interface.

This Guide describes WYDE Command Line Administration Interface only. If you need WYDE Web administration interface documentation, please see “Web Administration Interface – User Guide”.

Hardware

There are three primary WYDE platforms – the SB-HD100, the SB-HD1000 and the SB-HD10000. They are designed to be attached to your network in appliance-fashion. They are, for all intents and purposes, servers that can be installed and managed as such.

For more information on the WYDE hardware and its specifications – please visit www.wydevoice.com.

Voice Clients

There are three possible clients for WYDE bridge: soft phones, IP phones, and regular (PSTN) telephones. The most famous soft phones are Counterpath X-Lite, Flaphone, SJ Labs SJphone; also you can use Skype as your soft phone. The most famous manufacturers of IP phones are Audio Codes (with Audio Codes 320HD), Polycom HD Voice (with Polycom 550HD), and Snom (with Snom 320). In case of the regular telephone – there is a need to have a separate unit – a voice over IP gateway in order to use the phones, for example Cisco AS 5400 or similar.

Integration Adapters

WYDE can be integrated into an enterprise infrastructure through the set of adapters. There are three points of integration:

- **CDR storage** – A storage location for the individual call records. This can be SQL database or something else.
- **Authentication service** – This allows the WYDE software to integrate into the enterprise authentication systems. This could be a SQL database, RADIUS, LDAP, or other.
- **Call/Conference management** – This is the ability to manage conference calls, exposed through the Web API for integration with enterprise web sites.

Definitions, Acronyms and Abbreviations

In order to discuss the VMware virtual WYDE Voice demo server setup process 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 guide:

- **VoIP** – Voice over Internet Protocol, 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.
- **Node** – A computer with the *asterisk* service installed and running. The *asterisk* is being installed in *Frontend* components installation. If you are performing cluster

installation you can have multiple nodes, i.e. multiple *asterisk* computers in your WYDE bridge environment.

- **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.
- **Distributed Conference** – A conference that is taken place on the different bridges simultaneously. That means that the calls are being made to the different bridges, but these calls are participating in the same conference.
- **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.
- **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 web services 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.
- **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 unmuted to enable them to talk; however, the listener group as a whole will never be unmuted.

Chapter 2: Command Line Interface

The WYDE Command Line Administration Interface allows an administrator to make changes and administrate the operation of the system of the WYDE conference bridges. It consists of the following utilities:

- *wyde* command;
- *mf* console;
- *mp* console;
- *asterisk* (IVR) console.

The WYDE Command Line Administration Interface provides direct access to the embedded Linux subsystems of the WYDE conferencing appliances. The command line interface is the powerful tool to administer the bridge – using it you have more administration options at your disposal. In addition, you have direct access to the configuration of the various services that make up the WYDE architecture.

The WYDE Command Line Administration Interface is included with the WYDE conference bridges to allow you administrating of the WYDE bridges. Using *wyde* command you can manage users/subscribers, change call flow/DNIS information for various types of calls, and configure the WYDE bridge. Using *mf* console you can manage conferences and individual calls on-the-fly, i.e. control the conferences and calls that currently are in progress on the bridge.

As subscriber management you can create/view/delete them – manage their PIN, passwords, define subscriber properties, such as first and last names, password, email. In addition you can create/view/delete subscribers' conference accounts – define their conferences, used DNISes and access codes.

As DNIS management you can create/modify/delete DNIS associations of the actual inbound DNIS numbers and call flows that are used to service these numbers. In addition the command line interface allows you managing of call flows and their attributes.

As conference management you can manage the conferences and calls that currently are in progress, you can view started conferences and calls, as well as set and change some of their modes – mute the conferences and participants, placing the call on hold, making the call secure, record the call, etc.

In addition you can use command line administration interface to change system preferences, edit WYDE bridge parameters, perform configuration of the bridge, and monitor WYDE bridge activity.



It is possible to render your system inoperable if you are not familiar with the administration of Linux systems. Changing any of the bridge preferences, unless instructed to do so by WYDE technical support, also can render your system inoperable.

Prerequisites

You may use the WYDE command line administration interface to administer various components of the WYDE system. Before you do, however, you must have the following prerequisites met:

- A computer with a network connection to the TCP/IP subnet where the WYDE appliance is connected;
- SSH secure client to access to that computer – for instance PuTTY for Windows or SSH for Linux.

Accessing the Command Line Administration Interface

To access command line interface you should connect to Linux (CentOS) computer with the WYDE conferencing bridge software installed. You should use SSH secure client and connect to the proper IP address where the WYDE services are running.

Using the *wyde* Command Line Utility

The *wyde* command line utility is the tool that can be used to administrate the WYDE services. With it you can view and make changes to the configuration of the various WYDE services, manage subscribers and their conferences, DNISes, call flows, etc. From the command prompt, you can use the utility like so

```
wyde command [arguments]
```

where

- *command* – the specific *wyde* command that describes what kind of task you would like to accomplish;
- *arguments* – optional command arguments, i.e. options of the command that you are using.

To see the list of all available *wyde* command line utility commands you should use the command:

```
wyde help
```

You will see the screen similar to shown on Figure 2. All available *wyde* commands are listed in Table 1, the command reference is given in Chapter 3: Command Reference, Section: *wyde* Command Reference. These commands will be described in more detail later in this Guide.



```
root@ZILBER:~
login as: root
root@192.168.1.5's password:
Last login: Sun Mar 21 17:58:16 2010 from 192.168.1.99
[root@ZILBER ~]# wyde help
Syntax:
    wyde command [arguments]

Commands:
    ast-status          - Show WYDE asterisk status
    auth-adapter-add    - Add auth adapter.
    auth-adapter-del    - Delete auth adapter.
```

Figure 2: *wyde help* Command Output

Table 1: wyde Command Line Utility Available Commands

Commands	Description
ast-status	Show WYDE asterisk status.
auth-adapter-add	Add auth adapter.
auth-adapter-del	Delete auth adapter.
auth-adapter-show	Show auth adapters.
auth-method-add	Add auth method.
auth-method-del	Delete auth method.
auth-method-set	Setup auth method.
auth-method-show	Show auth methods.
billing-adapter-add	Add billing adapter.
billing-adapter-del	Delete billing adapter.
billing-adapter-set	Set billing adapter properties.
billing-adapter-show	Show billing adapters.
billing-rule-add	Add billing rule.
billing-rule-del	Delete billing rule.
billing-rule-set	Set billing rule.
billing-rule-show	Show billing rules.
bridge-add	Add WYDE bridge.
bridge-del	Delete WYDE bridge.
bridge-show	Show WYDE bridges.
callflow-add	Add callflow.
callflow-attr-set	Set callflow attribute.
callflow-attr-show	Show callflow attributes.
callflow-attr-update-db	Update callflow attributes in a database.
callflow-del	Delete callflow.
callflow-reload	Reload all callflows.
callflow-show	Show callflows table.
conference-attr-del	Remove conference attribute redefinition.
conference-attr-set	Set conference attribute.
conference-attr-show	Show conference attributes.
config-restore	Restore WYDE configuration.
config-save	Save WYDE configuration.
confuser-add	Add conference user.
confuser-del	Delete conference user.
confuser-show	Show conference users table.
db	Connect to WYDE database.
db-bil	Connect to WYDE billing database.
db-init	Initialize of WYDE database.
db-patch	Apply last patches for database.
did-add	Add DID number.
did-alias-add	Add DID number alias.
did-alias-del	Delete DID number alias.
did-alias-show	Show DID number aliases.
did-alias-show-all	Show all DID number aliases.
did-attr-del	Remove DNIS attribute redefinition.
did-attr-set	Set DNIS attribute.

Commands	Description
did-attr-show	Show DNIS's attributes.
did-del	Delete DID number.
did-reload	Reload all DID's caches.
did-show	Show DID's table.
drop-call	Drop call.
drop-conf	Drop conference.
help	Show help page and exit.
ivr	Connect to IVR console.
node-add	Add node to the WYDE bridge.
node-del	Delete node from the WYDE bridge.
node-set	Set node properties.
node-show	Show nodes of WYDE bridge.
register-license	Register License.
set-email	Change email address.
set-ip	Change ip address.
settings-edit	Edit WYDE system settings.
settings-show	Show WYDE settings.
settings-update	Update WYDE settings.
show-conf	Show conference or conferences list.
status	Show WYDE status.
subscriber-add	Add subscriber.
subscriber-del	Delete subscriber.
subscriber-show	Show subscribers table.
transfer	Transfer calls.
version	Show WYDE version.
watch	Watch WYDE status.

If you need detail help about any of these commands you should use the command:

```
wyde help command
where
```

- *command* – the specific *wyde* command on which you would like to get help.

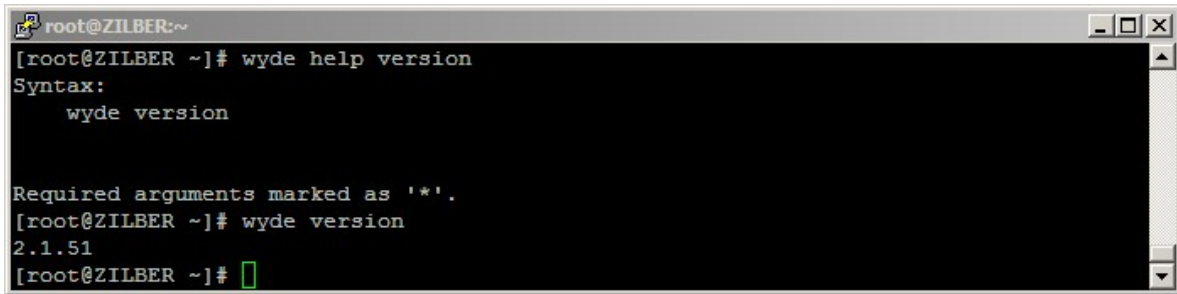
For instance, if you would like to get help on *version* command you should use the following command:

```
wyde help version
```

If you would like to show current WYDE version you should use the command:

```
wyde version
```

Sample of both these commands output is shown on Figure 3.

A terminal window titled 'root@ZILBER:~' with a standard Linux window control bar. The terminal shows the following text:

```
[root@ZILBER ~]# wyde help version
Syntax:
    wyde version

Required arguments marked as '*'.
[root@ZILBER ~]# wyde version
2.1.51
[root@ZILBER ~]#
```

Figure 3: *wyde help version* and *wyde version* Commands Output Sample

Using the *mf* Console

The *mf* (Multi Frontend Dispatcher) console is the tool that can be used to administrate the conferences and calls that currently are on the WYDE bridge. Once a conference has begun, you can manage it using *mf* console, you can view started conferences and calls, as well as set and change some of their modes – mute the conferences and participants, placing the call on hold, making the call secure, record the call, etc. To enter into this console you should just run from the command prompt the following command:

mf

The current version of WYDE services will be prompted to you one you login into the console.

To implement any *mf* console command you should type

command [arguments]

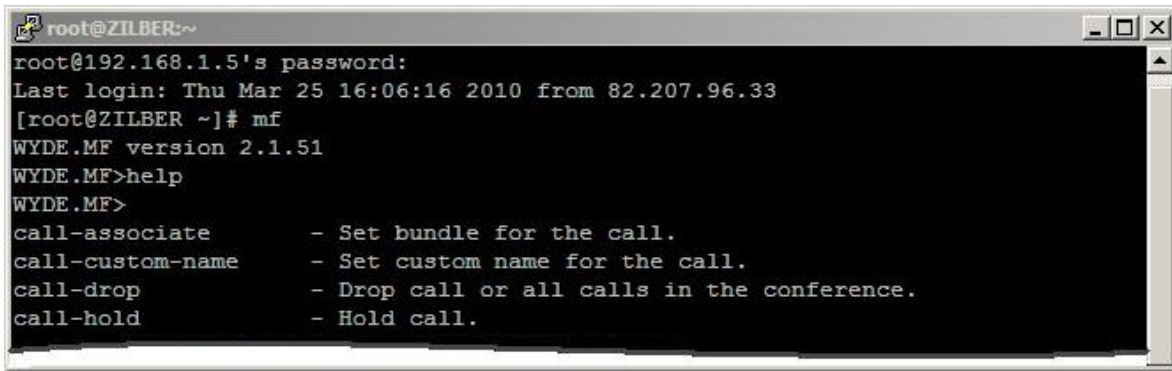
once you have entered into console, where

- *command* – the specific *mf* console command that describes what kind of task you would like to accomplish;
- *arguments* – optional command arguments, i.e. options of the command that you are using.

To see the list of all available *mf* console commands in the console you should type the command:

help

You will see the screen similar to shown on Figure 4. All available *mf* console commands are listed in Table 2, the command reference is given in Chapter 3: Command Reference, Section: *mf* Console Command Reference. These commands will be described in more detail later in this Guide.



```

root@ZILBER:~
root@192.168.1.5's password:
Last login: Thu Mar 25 16:06:16 2010 from 82.207.96.33
[root@ZILBER ~]# mf
WYDE.MF version 2.1.51
WYDE.MF>help
WYDE.MF>
call-associate      - Set bundle for the call.
call-custom-name    - Set custom name for the call.
call-drop           - Drop call or all calls in the conference.
call-hold           - Hold call.

```

Figure 4: *mf* Console, *help* Command OutputTable 2: *mf* Console Utility Available Commands

Commands	Description
call-associate	Set bundle for the call.
call-custom-name	Set custom name for the call.
call-drop	Drop call in the conference.
call-hold	Hold call.
call-move	Move call to other conference.
call-mute	Mute call.
call-qa-request	Start/stop Q&A request for the call.
call-qa-talk	Enable/disable Q&A session for the call in the queue.
callflow-reload	Reload callflows.
cmdcount-show	Display values of command counters.
conf-broadcast	Start/stop broadcast mode.
conf-drop	Drop conference.
conf-hold-group	Hold group.
conf-jobcode	Set JobCode for the conference.
conf-mute-group	Mute group.
conf-play-file	Manage of playing file to the conference.
conf-polling	Conference polling.
conf-qa-mode	Manage Q&A sessions.
conf-qa-mute	Mute/Unmute active Q&A session.
conf-qa-talk	Enable Q&A session for the first call in the queue.
conf-recording	Start/stop conference recording.
conf-schedule-extend	Extend scheduled conference duration.
conf-schedule-incsize	Resize scheduled conference subscription.
conf-secure	Secure conference.
conf-shunt	Make/drop shunt between two conferences.
confcount-show	Display values of confcount counters.
dc-show-bridges	Show known DC bridges.
dc-show-links	Show DC links.
dialout	Do dialout.
dialout-attr	Show dialout attributes for specified conference.
did-reload	Reload DID entries.
errcount-show	Display values of error counters.
freenumbers-show	Show free number leases.

Commands	Description
help	Show help for console commands.
moh-reload	Reload customer's MOHs.
node-reload	Reload MF cluster nodes list.
node-show	Show MF cluster nodes list.
op-call-move	Move user that currently talking with operator to other conference.
op-dialout	Initiate dialout from operator's console.
op-listen	Listen conference.
op-queue	Display operator calls queue.
op-scan	Scan conferences.
op-show	Display operators.
op-talk	Operator talk to user.
partcount-show	Display values of partcount counters.
peer-reload	Reload peers.
quit	Quit console.
set-log-level	Set logger level.
settings-reload	Reload system settings.
show	Show conferences and calls.
transfer	Transfer calls.
welcomeprompt-reload	Reload welcome prompts.

If you need detail help about any of these commands you should use the command:

```
help command
```

where

- *command* – the specific *mf* command on which you would like to get help.

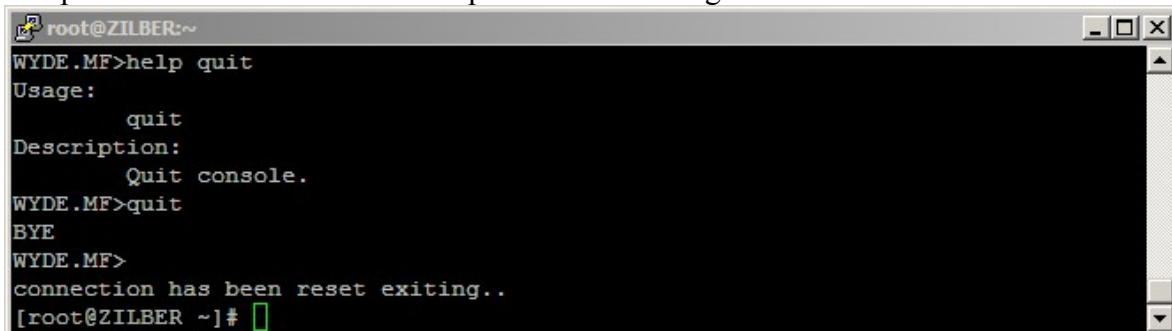
For instance, if you would like to get help on *quit* command you should use the following command:

```
help quit
```

If you would like to exit from *mf* console you should use the command:

```
quit
```

Sample of both these commands output is shown on Figure 5.



```

root@ZILBER:~
WYDE.MF>help quit
Usage:
    quit
Description:
    Quit console.
WYDE.MF>quit
BYE
WYDE.MF>
connection has been reset exiting..
[root@ZILBER ~]#

```

Figure 5: *mf* Console *help quit* and *quit* Commands Output Sample

Subscriber User Management

One of the administration tasks that you will be called upon frequently to do is to add unique subscribers to the system and allow them to use the system. For terminology's sake we should clarify one main terminology item. A subscriber is a real person – he has a name, phone number, e-mail address, etc. A user is a person in a conference with a particular role (e.g. host, participant, listener, etc). So a subscriber could be a host user in one conference and a listener in another and a subscriber can have different roles in the same the conferences.

Create a Subscriber

Whenever a new person needs access to the system, you must create a new subscriber for them so that they can log in to the system. To add subscribers to the system using the command line, you would use the *wyde* command line utility with the *subscriber-add* option. The syntax is as follows:

```
wyde subscriber-add <arguments>
```

Each of the arguments is followed by a space and a value. In *subscriber-add* you can specify the following arguments:

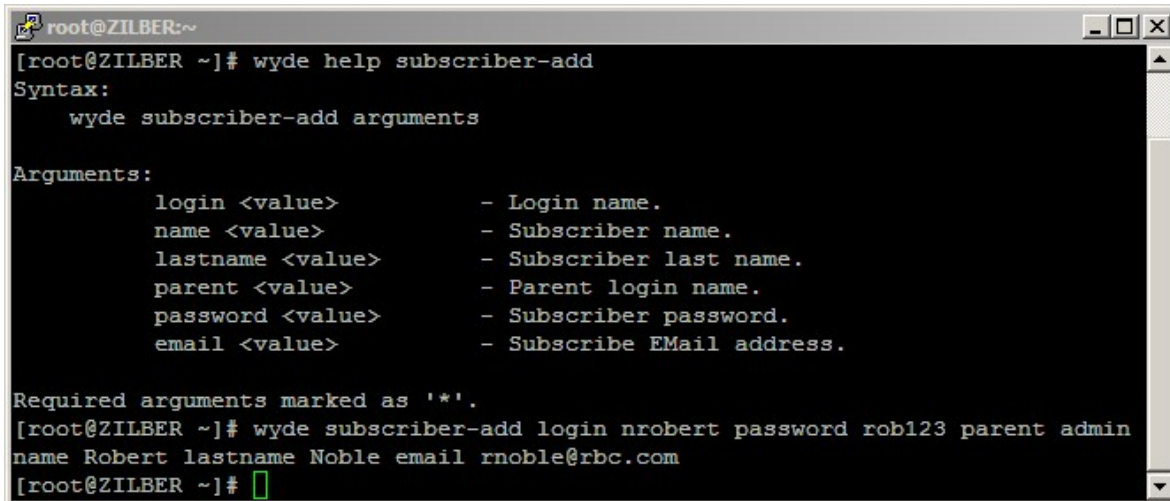
- `login <value>` – PIN for new subscriber, essentially the login ID for the subscriber. It is usually a number so this PIN can be used as a login id by telephone for additional conference features to identify the subscriber, if this is determined by call flow. This login also can be used as login into this Web Administration Interface. If the login is planning to be used as web login only, the PIN can be alpha-numeric. If you don't want to come up with your own numbering scheme for logins, you can omit this argument to generate a random unique ID for the new subscriber.
- `name <value>` – The subscriber's first name.
- `lastname <value>` – The subscriber's last name.
- `parent <value>` – Parent login name, i.e. parent subscriber PIN. A parent is the user account that created this account.
- `password <value>` – This is a password for the subscriber, for instance it can be used to login into Web Administration Interface.
- `email <value>` – The email address of the subscriber.

The arguments can be transferred to this command in any order.

For example you can add new subscriber using the following command (new subscriber attributes are shown in *italic*):

```
wyde subscriber-add login nrobert password rob123  
    parent admin name Robert lastname Noble  
    email rnoble@rbc.com
```

If the command is successful, the system will not return any errors or messages, in typical Linux fashion, it will just drop you back to the command prompt (#). The sample of the *subscriber-add* command output and the help on this command is shown on Figure 6.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help subscriber-add
Syntax:
  wyde subscriber-add arguments

Arguments:
  login <value>          - Login name.
  name <value>           - Subscriber name.
  lastname <value>       - Subscriber last name.
  parent <value>         - Parent login name.
  password <value>       - Subscriber password.
  email <value>          - Subscribe EMail address.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde subscriber-add login nrobert password rob123 parent admin
name Robert lastname Noble email rnoble@rbc.com
[root@ZILBER ~]#

```

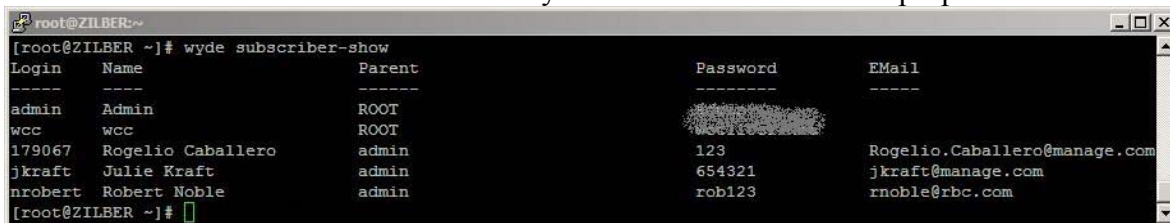
Figure 6: *wyde help subscriber-add* and *wyde subscriber-add* Commands Output Sample

View/Modify a Subscriber

To show a list of the subscribers in the system using the command line, you should use the *wyde* command line utility with the *subscriber-show* option. The syntax is as follows:

```
wyde subscriber-show
```

This command will output a list of the all existed subscribers on the system, similar to shown on Figure 7. As you can see, the *wyde subscriber-show* command shows the subscribers that have been created in the system as well as their basic properties.



```

root@ZILBER:~
[root@ZILBER ~]# wyde subscriber-show
Login      Name      Parent      Password      EMail
-----
admin      Admin     ROOT        [REDACTED]
wcc        wcc       ROOT
179067     Rogelio Caballero admin      123           Rogelio.Caballero@manage.com
jkraft     Julie Kraft admin      654321        jkraft@manage.com
nrobert    Robert Noble admin      rob123        rnoble@rbc.com
[root@ZILBER ~]#

```

Figure 7: *wyde subscriber-show* Command Output Sample

The *wyde* command line utility does not allow changing of subscribers. You can use Web Administration Interface for this purpose (please read “Web Administration Interface – User Guide” if you need assistance in subscriber modification).

Delete a Subscriber

From time to time, you will need to delete a subscriber.

To delete a subscriber using the *wyde* command line utility you should use *subscriber-del* option. The syntax is as follows:

```
wyde subscriber-del login <login>
```

where

- *<login>* – login name, i.e. PIN of the subscriber you wish to delete.

For example to delete subscriber *nrobert* (created in previous sample) you should run the command:

```
wyde subscriber-del login nrobert
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Create a Conference Account

A conference account (conference user) is essentially a set of configuration settings for a particular subscriber for a particular DNIS/call flow. Within a conference account configuration, each given a unique ID#, you specify how a particular subscriber will interact – whether they will be host, participant, or listener, which DNIS/call flow will be used, and what the access code will be used for that conference account.

To add conference accounts (users) to a conference using the command line interface you should use the *wyde* command line utility with the *confuser-add* option. The syntax is as follows:

```
wyde confuser-add <arguments>
```

Each of the arguments is followed by a space and a value (if present). In *confuser-add* you can specify the following arguments:

- `accesscode <value>` – Access code; the value can be any number. If this argument omitted unique access code will be generated.
- `did <value>` – DID (DNIS) number.
- `conference <value>` – Conference number; the value can be any number. If this argument omitted unique access code will be generated.
- `conf-accesscode` – Set the conference number the same as access code. If the conference number argument was transferred together with this argument, the conference number will be ignored, and the conference number will be set equal to the access code.
- `role <value>` – Role in the conference: *Host* or *Participant* or *Listener*. Note: this argument is case-sensitive.
- `subscriber <value>` – Subscriber login name, i.e. subscriber PIN.

Arguments *did*, *role*, and *subscriber* are required. The arguments can be transferred to this command in any order.

For example if you would like to create the conference account for subscriber *nrobert* with DID (DNIS) number *(866) 508-0012*, conference number *444012*, access code *304050* with *Host* role you should run the following command (new conference account attributes are shown in *italic*):

```
wyde confuser-add subscriber nrobert did 8665080012  
conference 444012 accesscode 304050 role Host
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *confuser-add* command output and the help on this command is shown on Figure 8.

```

root@ZILBER:~
[root@ZILBER ~]# wyde help confuser-add
Syntax:
    wyde confuser-add arguments

Arguments:
    accesscode <value>      - Access code.
    * did <value>           - DID number.
    conference <value>      - Conference number.
    conf-accesscode         - Set conference number same as access code.
    * role <value>          - Role in conference {Host|Participant}.
    * subscriber <value>    - Subscriber login name.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde confuser-add subscriber nrobert did 8665080012 conference
444012 accesscode 304050 role Host
[root@ZILBER ~]#

```

Figure 8: `wyde help confuser-add` and `wyde confuser-add` Commands Output Sample

Configuration of the Conferences

Note that it could be different approaches with conferences configuration and setting access codes and roles for the conferences. If you would like to configure the single conference, the conference number must be the same in all conference accounts records that describe this single conference. However the DNIS number and/or subscriber for them could be different; the way that the system determines that it is the single conference is the same conference number.

The first approach to configure the conference is the following: the subscriber creates for himself the conference account records with the same conference number (this determines that it is the single conference configuration) and multiple records with different access codes that are used for different roles. Depending on what access code is used when the user has entered in the conference, the role of this user is different, and the user can be either the host, or the participant or the listener. For example if you need to create for the subscriber *nrobert* the second conference account with participant role and the third conference account with listener role you should execute the following commands:

```

wyde confuser-add subscriber nrobert did 8665080012
    conference 444012 accesscode 304060 role Participant
wyde confuser-add subscriber nrobert did 8665080012
    conference 444012 accesscode 304070 role Listener

```

In this sample for the conference *444012* the host access code is *304050*, the participant access code is *304060*, the listener access code is *304070*; DNIS number in all cases is the same: *8665080012*.

The second approach to configure the conference is creating conference accounts for each subscriber that should participate in the conference. The conference number for all these conference accounts must be the same, and these records will show what access code should use the subscriber and subscriber role in the conference call.

View/Modify a Conference Account

To show a list of the conference accounts (users) in the system using the command line, you should use the *wyde* command line utility with the *confuser-show* option. The syntax is as follows:

```
wyde confuser-show <arguments>
```

Each of the arguments is followed by a space and a value. In *confuser-show* you can specify the following arguments:

- *subscriber* <value> – Subscriber login name, i.e. subscriber PIN; if this argument is transferred this command will return only conference accounts (users) that belong to the specified subscriber.
- *accesscode* <value> – Access code; if this argument is transferred this command will return only conference accounts (users) with the specified access code.

If no arguments were transferred this command lists all conference accounts that exist on the bridge.

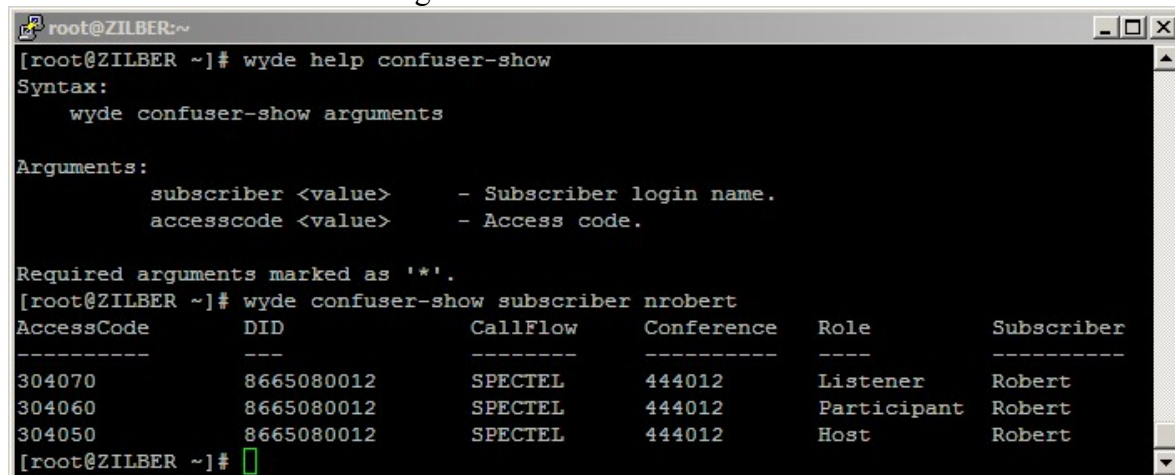
For example the command:

```
wyde confuser-show
```

returns all existed conference accounts, but the command:

```
wyde confuser-show subscriber nrobert
```

returns conference accounts that belong to subscriber *nrobert*, see Figure 9 for details. As you can see, the *wyde confuser-show* command shows the conference accounts that have been created in the system as well as their basic properties: the access codes, DNIS (DID) numbers, call flow names, conference numbers, subscriber's role in that conference, and the first name of the subscriber assigned for each conference account.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help confuser-show
Syntax:
    wyde confuser-show arguments

Arguments:
    subscriber <value>      - Subscriber login name.
    accesscode <value>     - Access code.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde confuser-show subscriber nrobert
AccessCode    DID          CallFlow    Conference    Role          Subscriber
-----
304070        8665080012    SPECTEL    444012        Listener     Robert
304060        8665080012    SPECTEL    444012        Participant  Robert
304050        8665080012    SPECTEL    444012        Host         Robert
[root@ZILBER ~]#

```

Figure 9: *wyde help confuser-show* and *wyde confuser-show* Commands Output Sample

The *wyde* command line utility does not allow changing of conference accounts properties. You can use Web Administration Interface for this purpose (please read “Web Administration Interface – User Guide” if you need assistance in conference accounts modification).

Delete a Conference Account

If you wish to delete the specific conference account, you can use the *wyde* command line utility with *confuser-del* option. The syntax is as follows:

```
wyde confuser-del accesscode <access code>
```

where

- *<access code>* – the access code of the conference account you wish to delete.

For example to delete the conference account with access code: *304070* (the *listener* role conference account created in previous sample) you should run the command:

```
wyde confuser-del accesscode 304070
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Override a Call Flow Attribute Value for a Conference

Call flow attributes can be overridden for a particular subscriber conference definition (conference account). That means that some call flow attributes could be redefined for the specific conference.

To override conference call flow attributes using the command line interface you should use the *wyde* command line utility with the *conference-attr-set* option. The syntax is as follows:

```
wyde conference-attr-set <arguments>
```

Each of the arguments is followed by a space and a value. In *conference-attr-set* you can specify the following arguments:

- *number <value>* – The conference number.
- *name <value>* – This conference call flow attribute name you would like to override.
- *value <value>* – This conference call flow attribute new value.

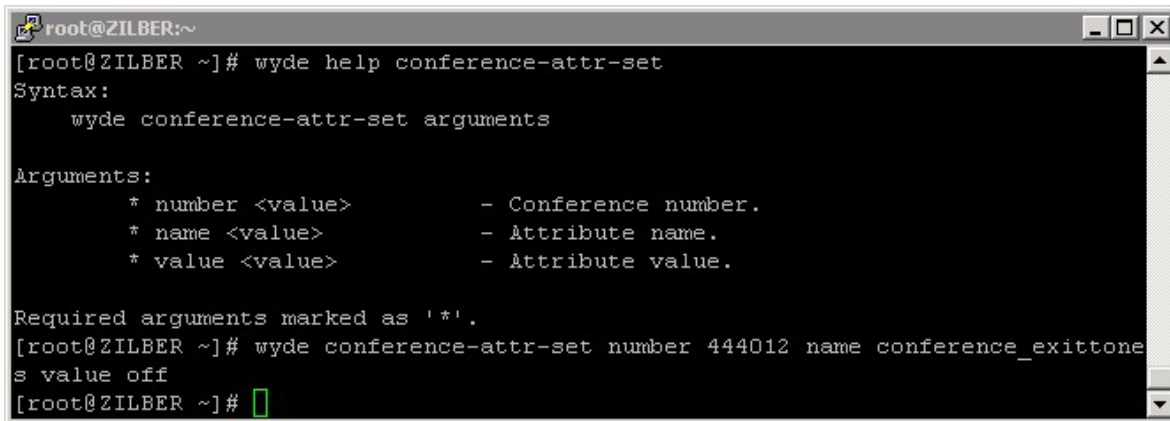
All these arguments are required. The arguments can be transferred to this command in any order.

The default *conference_exittones* (exit tones) call flow attribute value is “on”.

Let’s assume that we need to override its value to “off” for the conference *444012* that we used in previous samples. In this case you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde conference-attr-set number 444012  
    name conference_exittones value off
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *conference-attr-set* command output and the help on this command is shown on Figure 10.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help conference-attr-set
Syntax:
    wyde conference-attr-set arguments

Arguments:
    * number <value>          - Conference number.
    * name <value>            - Attribute name.
    * value <value>           - Attribute value.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde conference-attr-set number 444012 name conference_exittone
s value off
[root@ZILBER ~]#

```

Figure 10: *wyde help conference-attr-set* and *wyde conference-attr-set* Commands Output Sample

If you would like to set new value for the conference call flow attribute that was already overridden, you can just perform another *conference-attr-set* command with new attribute value.

View Call Flow Attributes Values for a Conference

To show a list of all call flow attributes for the specific conference (including the attributes defined on call flow level and the attributes overridden on DNIS/conference level) using the command line, you should use the *wyde* command line utility with the *conference-attr-show* option. The syntax is as follows:

```
wyde conference-attr-show number <conference number>
```

where

- <conference number> – The number of the conference which attributes you would like to view.

For example if you would like to see the list all call flow attributes for the conference *444012* you should use the command:

```
wyde conference-attr-show number 444012
```

This command outputs all call flow attributes for this conference regardless were they defined on call flow level or on DNIS level or on conference level, see Figure 11 for details. As you can see, the *wyde conference-attr-show* command shows the conference call flow attributes names and values; the last column “*Override*” contains information showing was the attribute overridden either on DNIS (DID) level – in this case this column contains “*DID” or on conference level – in this case this column contains asterisk “*”.

```

root@ZILBER:~# wyde conference-attr-show number 444012
Name                                     Value                                     Override
----                                     -
call_announceparticipantcount          hp1
call_exit_dtmf
call_instructions_dtmf                  hp
call_jobcodeonenter
call_mute_dtmf                          hp
call_operator_dtmf
call_participantsnumber_dtmf            hp
conference_callerdb                     on                                       *DID
conference_dialout_dtmf                 h
conference_distributed                   off
conference_entryexittones_dtmf
conference_entrvtones                   on
conference_exittones                     off                                    *
conference_extended_dtmf                hp1

```

Figure 11: *wyde conference-attr-show* Command Output Sample

Delete a Conference Call Flow Attribute Redefinition

If you wish to delete the specific conference call flow attribute that was previously overridden, you can use the *wyde* command line utility with *conference-attr-del* option. The syntax is as follows:

```
wyde conference-attr-del <arguments>
```

Each of the arguments is followed by a space and a value. In *conference-attr-show* you can specify the following arguments:

- *number* <value> – The conference number which attribute you wish to delete.
- *name* <value> – This conference call flow attribute name you wish to delete.

Both arguments are required. The arguments can be transferred to this command in any order.

For example to delete the conference *444012* call flow attribute

conference_exittones (overridden in previous sample) you should run the command:

```
wyde conference-attr-del number 444012
      name conference_exittones
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Call Flow and DNIS Management

One piece of terminology unique to conference call setup is the idea of a call flow. In other words, the call flow is unique conference call setup, the logic that is used to process the 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. The various system settings and usage parameters are different depending on the call flow used.

The following basic call flows are available in the system:

- CONF
- OPERATOR
- PLAYBACK
- SPECTEL

One of the benefits of the WYDE architecture is the ability to build a complete custom call flow for your organization. So additional custom call flows could be created as customization of the existing call flows. For more information on a custom call flow, please contact your WYDE representative.

A DNIS is the 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 less than 10 digits). As DNIS number you can also use the mask of the number, in such mask '*' – denotes any sequence of digits and '?' – denotes one single digit; in this case for such DNISes the bridge will process all numbers that comply with the DNIS number mask. For the purposes of this guide, a DNIS Association is a combination of a call flow and an actual inbound DNIS (DID) number or number mask.

You can read detail information about call flows and DNISes in “Web Administration Interface – User Guide”, Chapter 3: Call Flows and Section: DNIS Management.

Using the command line administration interface you will be able to change parameters of the call flows and DNISes, as it will be described later in this chapter.



Changing any of the call flow attributes, unless instructed to do so by WYDE technical support, can render your system inoperable.

Add a Call Flow

Before you register new call flow, you should create the folder for this call flow. This folder should contain the files *callflow.spec* (Asterisk dialplan on AEL language file) and *script.ael* (specification of the scenario file) and optionally subfolder *sounds* for audio (voice) files. The default root folder for call flows is */usr/local/DNCA/callflows/*. You can read “Call Flow Development – Programmer’s Guide” for additional information about call flow folder structure and contents.

To add new call flow registration using the command line interface you should use the *wyde* command line utility with the *callflow-add* option. The syntax is as follows:

```
wyde callflow-add <arguments>
```

Each of the arguments is followed by a space and a value. In *callflow-add* you can specify the following arguments:

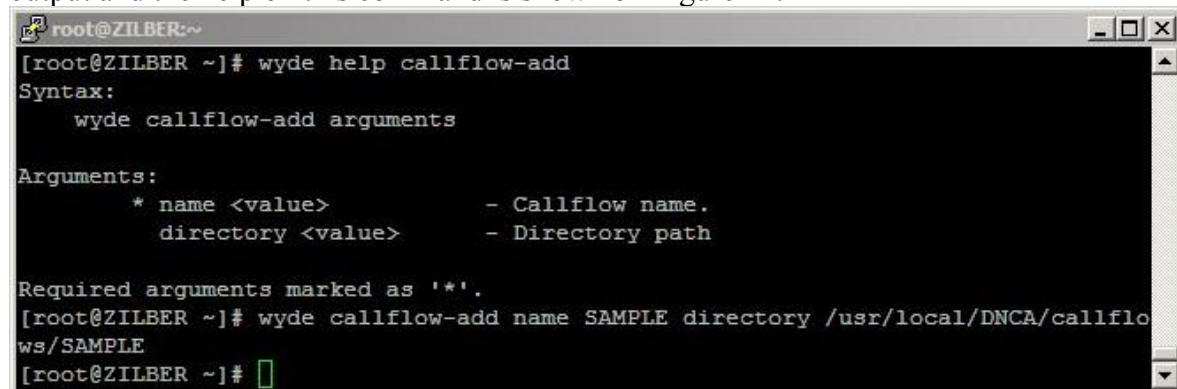
- *name* <value> – The name of new call flow that should be registered. This is required argument. This name should be unique, i.e. there should no be any other call flow with the same name on the bridge.
- *directory* <value> – Call flow directory path. If this parameter omitted the current folder will be used as this parameter value.

The arguments can be transferred to this command in any order.

Let's assume that we have created the folder */usr/local/DNCA/callflows/SAMPLE/* for new call flow *SAMPLE* that contains required files *callflow.spec*, *script.ael* and *sound* subfolder. To add this call flow to the bridge you should use the command:

```
wyde callflow-add name SAMPLE
    directory /usr/local/DNCA/callflows/SAMPLE
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *callflow-add* command output and the help on this command is shown on Figure 12.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help callflow-add
Syntax:
    wyde callflow-add arguments

Arguments:
    * name <value>          - Callflow name.
    directory <value>       - Directory path

Required arguments marked as '*'.
[root@ZILBER ~]# wyde callflow-add name SAMPLE directory /usr/local/DNCA/callflows/SAMPLE
[root@ZILBER ~]#
  
```

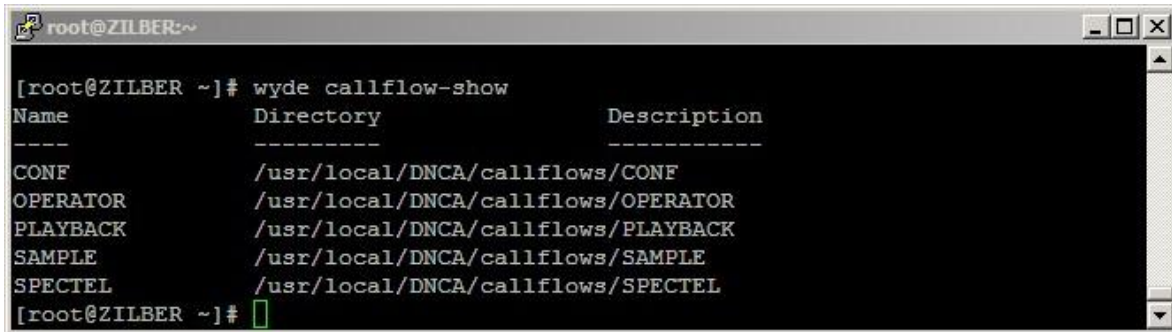
Figure 12: *wyde help callflow-add* and *wyde callflow-add* Commands Output Sample

View a Call Flow

To show a list of all call flows in the system using the command line, you should use the *wyde* command line utility with the *callflow-show* option. The syntax is as follows:

```
wyde callflow-show
```

This command will output a list of the all existed call flows on the system, similar to shown on Figure 13. As you can see, the *wyde callflow-show* command shows the call flows that have been created in the system as well as their basic properties: call flow name and directory path.



```

root@ZILBER:~# wyde callflow-show
Name          Directory          Description
-----
CONF          /usr/local/DNCA/callflows/CONF
OPERATOR      /usr/local/DNCA/callflows/OPERATOR
PLAYBACK      /usr/local/DNCA/callflows/PLAYBACK
SAMPLE        /usr/local/DNCA/callflows/SAMPLE
SPECTEL       /usr/local/DNCA/callflows/SPECTEL
root@ZILBER:~#

```

Figure 13: *wyde callflow-show* Command Output Sample

Delete a Call Flow

To delete a call flow using the *wyde* command line utility you should use *callflow-del* option. The syntax is as follows:

```
wyde callflow-del name <call flow name>
```

where

- *<call flow name>* – the name of the call flow you wish to delete.

Note: when you delete the call flow all DNISes associated with this call flow and all this call flow conference accounts also will be deleted.

For example to delete call flow *SAMPLE* (created in previous sample) you should run the command:

```
wyde callflow-del name SAMPLE
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Set a Call Flow Attribute Value

To set a call flow attribute value using the command line interface you should use the *wyde* command line utility with the *callflow-attr-set* option. The syntax is as follows:

```
wyde callflow-attr-set <arguments>
```

Each of the arguments is followed by a space and a value. In *callflow-attr-set* you can specify the following arguments:

- *callflow <value>* – The name of the call flow which attribute value you would like to set.
- *name <value>* – This call flow attribute name you would like to update.
- *value <value>* – This call flow attribute new value.

The arguments *callflow* and *name* are required; if the argument *value* is omitted the empty value will be set to this attribute. The arguments can be transferred to this command in any order.

For example if you would like to change *call_instructions_dtmf* (instructions DTMF policy) call flow attribute value to “h”, i.e. make the instructions available for hosts only, for the call flow *SPECTEL*, you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde callflow-attr-set callflow SPECTEL
    name call_instructions_dtmf value h
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *callflow-attr-set* command output and the help on this command is shown on Figure 14.

```
root@ZILBER:~
[root@ZILBER ~]# wyde help callflow-attr-set
Syntax:
    wyde callflow-attr-set arguments

Arguments:
    * callflow <value>          - Callflow name.
    * name <value>              - Attribute name.
    value <value>              - Attribute value.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde callflow-attr-set callflow SPECTEL name call_instructions_
dtmf value h
[root@ZILBER ~]#
```

Figure 14: *wyde help callflow-attr-set* and *wyde callflow-attr-set* Commands Output Sample

View Call Flow Attributes Values

To show a list of all call flow attributes for the specific call flow using the command line, you should use the *wyde* command line utility with the *callflow-attr-show* option. The syntax is as follows:

```
wyde callflow-attr-show callflow <call flow name>
```

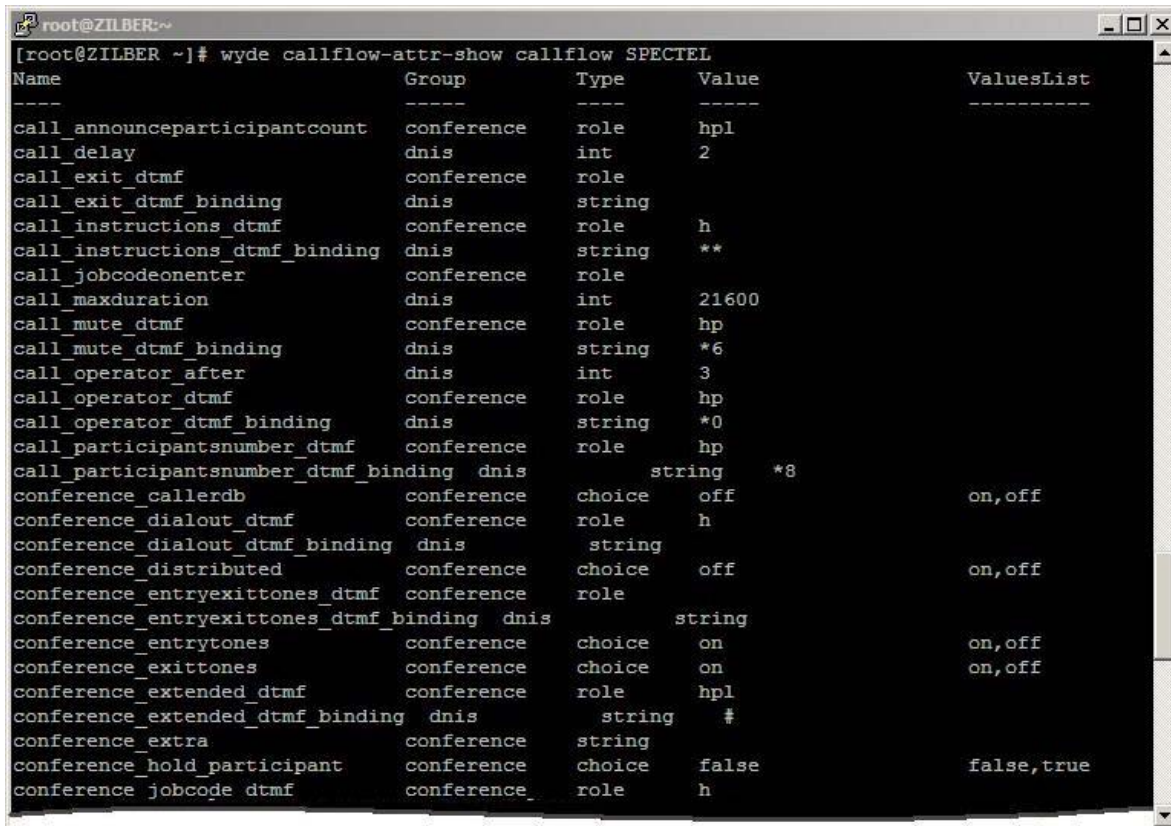
where

- *<call flow name>* – The name of the call flow which attributes you would like to view.

For example if you would like to see the call flow *SPECTEL* attributes you should use the command:

```
wyde callflow-attr-show callflow SPECTEL
```

This command outputs all specific call flow attributes, see Figure 15 for details. As you can see, the *wyde callflow-attr-show* command shows the call flow attributes names, groups, types, values, and allowed values list (if applicable). The group of an attribute can be either “conference” or “dnis”; when the attribute group is “dnis”, the attribute can be overridden in DNIS Associations only; when the attribute role is “conference”, the attribute can be overridden either in DNIS Associations or in Subscriber Conferences – see “Web Administration Interface – User Guide” for details. The type of an attribute can have one of the following values: “role” (the attribute determines the policy, i.e. the person to whom this option should be available – hosts (h) or participants (p) or listeners (l)), “choice” (only values specified in values list are permitted for this attribute), “string”, or “int”.



```

root@ZILBER:~# wyde callflow-attr-show callflow SPECTEL
Name                               Group      Type      Value      ValuesList
-----
call_announceparticipantcount     conference role      hpl
call_delay                         dn timer   int      2
call_exit_dtmf                    conference role
call_exit_dtmf_binding            dn timer   string
call_instructions_dtmf            conference role      h
call_instructions_dtmf_binding     dn timer   string   **
call_jobcodeonenter               conference role
call_maxduration                  dn timer   int      21600
call_mute_dtmf                    conference role      hp
call_mute_dtmf_binding            dn timer   string   *6
call_operator_after               dn timer   int      3
call_operator_dtmf                conference role      hp
call_operator_dtmf_binding         dn timer   string   *0
call_participantsnumber_dtmf       conference role      hp
call_participantsnumber_dtmf_binding dn timer   string   *8
conference_callerdb               conference choice    off      on,off
conference_dialout_dtmf            conference role      h
conference_dialout_dtmf_binding     dn timer   string
conference_distributed             conference choice    off      on,off
conference_entryexittones_dtmf     conference role
conference_entryexittones_dtmf_binding dn timer   string
conference_entrytones              conference choice    on      on,off
conference_exittones               conference choice    on      on,off
conference_extended_dtmf           conference role      hpl
conference_extended_dtmf_binding     dn timer   string   #
conference_extra                   conference string
conference_hold_participant         conference choice    false   false,true
conference_jobcode_dtmf             conference role      h

```

Figure 15: *wyde callflow-attr-show* Command Output Sample

Reload All Call Flows

If there were changes in the *script.ael* file or sound files were updated in *sounds* subfolder for any of the call flows you should reload all call flows.

To send the signal on the WYDE bridge to the call flow engine to reload the scripts the following command should be executed:

```
wyde callflow-reload
```

If this command is successful, you will be returned to the command line with no additional prompts.

As alternative of this *wyde* command you can use *mf* console command *callflow-reload* for the same purposes. This *wyde* command is basically used in automated scripts and for convenient call flow modification.

Update Call Flow Attributes Definition in a Database

If there were changes in call flow attributes definition and *callflow.spec* file was updated you should update call flow attributes definition in the database.

To send the signal on the WYDE bridge to update the call flow attributes for the specific call flow the following command should be executed:

```
wyde callflow-attr-update-db callflow <call flow name>
```

where

- <call flow name> – the name of the call flow which attributes you wish to update.

For example if the definition of call attributes for call flow SAMPLE (described in samples) were updated, i.e. the *callflow.spec* file in the folder */usr/local/DNCA/callflows/SAMPLE/* was changed, you should implement the following command:

```
wyde callflow-attr-update-db callflow SAMPLE
```

Create a DNIS Association

To create new DNIS association of a call flow and an actual inbound DNIS (DID) number using the command line interface you should use the *wyde* command line utility with the *did-add* option. The syntax is as follows:

```
wyde did-add <arguments>
```

Each of the arguments is followed by a space and a value. In *did-add* you can specify the following arguments:

- number <value> – Dial-in number, i.e. DNIS (DID) number, you want to be associated with this DNIS Association.
- callflow <value> – The name of the call flow to be used with this DNIS Association.
- description <value> – DNIS (DID) description.

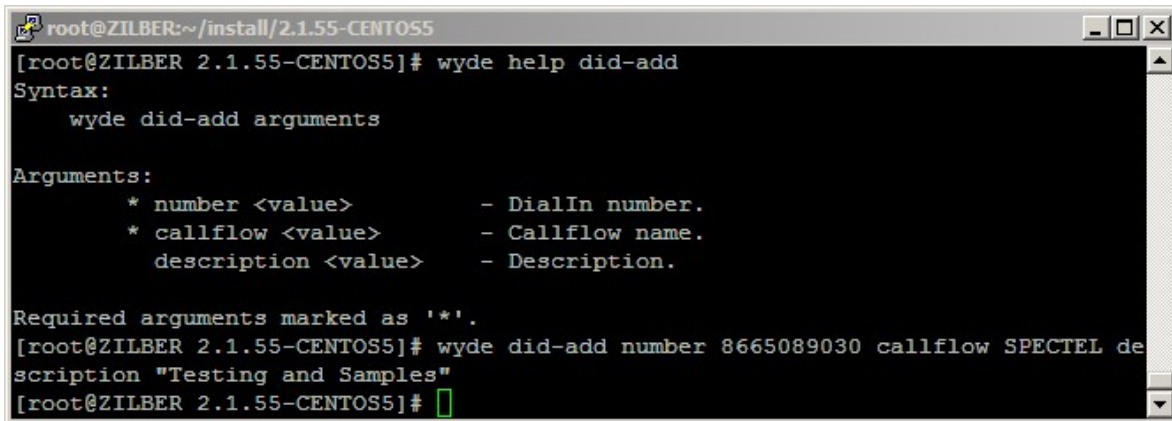
Arguments number and callflow are required. The arguments can be transferred to this command in any order.

For example if you would like to create the DNIS Association of DID (DNIS) number (866) 508-9030 and call flow *SPECTEL* you should run the following command (new DNIS association properties are shown in *italic*):

```
wyde did-add number 8665089030 callflow SPECTEL  
description "Testing and Samples"
```

Note that to set the description that contains spaces you should use double quotes (").

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *did-add* command output and the help on this command is shown on Figure 16.



```

root@ZILBER:~/install/2.1.55-CENTOS5
[root@ZILBER 2.1.55-CENTOS5]# wyde help did-add
Syntax:
    wyde did-add arguments

Arguments:
    * number <value>          - DialIn number.
    * callflow <value>        - Callflow name.
    description <value>      - Description.

Required arguments marked as '*'.
[root@ZILBER 2.1.55-CENTOS5]# wyde did-add number 8665089030 callflow SPECTEL de
scription "Testing and Samples"
[root@ZILBER 2.1.55-CENTOS5]#

```

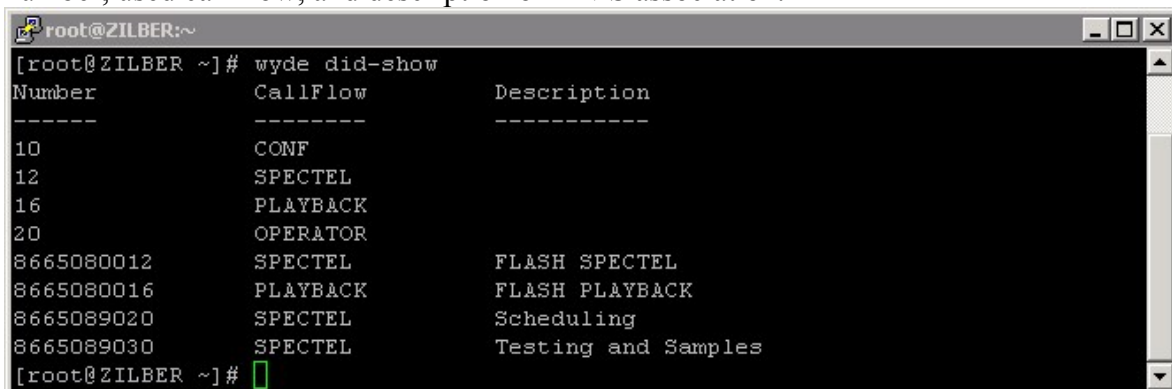
Figure 16: *wyde help did-add* and *wyde did-add* Commands Output Sample

View/Modify a DNIS Association

To show a list of the DNIS associations in the system using the command line, you should use the *wyde* command line utility with the *did-show* option. The syntax is as follows:

```
wyde did-show
```

This command will output a list of the all existed DNIS associations on the system, similar to shown on Figure 17. As you can see, the *wyde did-show* command shows the DNIS associations that have been created in the system as well as their basic properties: dial-in number, used call flow, and description of DNIS association.



```

root@ZILBER:~
[root@ZILBER ~]# wyde did-show
Number      CallFlow      Description
-----
10          CONF
12          SPECTEL
16          PLAYBACK
20          OPERATOR
8665080012  SPECTEL      FLASH SPECTEL
8665080016  PLAYBACK     FLASH PLAYBACK
8665089020  SPECTEL      Scheduling
8665089030  SPECTEL      Testing and Samples
[root@ZILBER ~]#

```

Figure 17: *wyde did-show* Command Output Sample

The *wyde* command line utility does not allow changing of DNIS associations. You can use Web Administration Interface for this purpose (please read “Web Administration Interface – User Guide” if you need assistance in DNIS associations modification).

Create a DNIS Number Alias

The WYDE conferencing bridge software allows you to create aliases for DNIS (DID) numbers. That means that if you need to have multiple dial-in numbers that should operate the same way and have the same properties, you can create one DNIS association for so called “main” DNIS (DID) number and define other numbers as aliases for that one. In this case the DNIS configuration is being made for this main DNIS number only, and you should not configure additional DNISes for additional DNIS numbers.

To create new DNIS number alias using the command line interface you should use the *wyde* command line utility with the *did-alias-add* option. The syntax is as follows:

```
wyde did-alias-add <arguments>
```

Each of the arguments is followed by a space and a value. In *did-alias-add* you can specify the following arguments:

- *number* <value> – Main dial-in number, i.e. DNIS (DID) number for which you have created DNIS Association.
- *alias* <value> – The alias of the main dial-in number that you wish to add.
- *description* <value> – The description of DNIS (DID) alias.

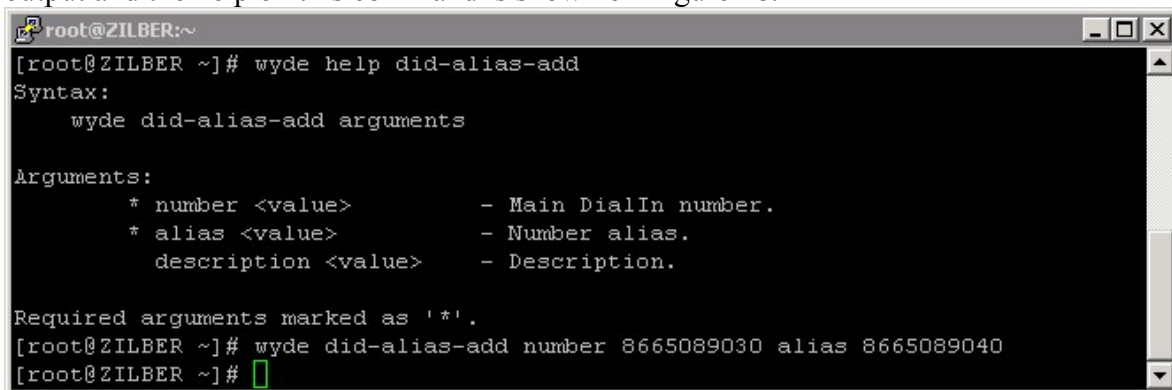
Arguments *number* and *alias* are required. The arguments can be transferred to this command in any order.

For example if you would like to create the alias number *(866) 508-9040* for the DNIS number *(866) 508-9030* (that was used in previous samples) you should run the following command (both dial-in numbers are shown in *italic*):

```
wyde did-alias-add number 8665089030 alias 8665089040
```

Note that in this sample we do not create the description for the alias.

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *did-alias-add* command output and the help on this command is shown on Figure 18.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help did-alias-add
Syntax:
    wyde did-alias-add arguments

Arguments:
    * number <value>          - Main DialIn number.
    * alias <value>           - Number alias.
    description <value>      - Description.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde did-alias-add number 8665089030 alias 8665089040
[root@ZILBER ~]#

```

Figure 18: *wyde help did-alias-add* and *wyde did-alias-add* Commands Output Sample

View DNIS Number Aliases

To show a list of all DNIS number aliases in the system using the command line, you should use the *wyde* command line utility with the *did-alias-show-all* option. The syntax is as follows:

```
wyde did-alias-show-all
```

To show a list of DNIS number aliases for the specific DNIS association only using the command line, you should use the *wyde* command line utility with the *did-alias-show* option. The syntax is as follows:

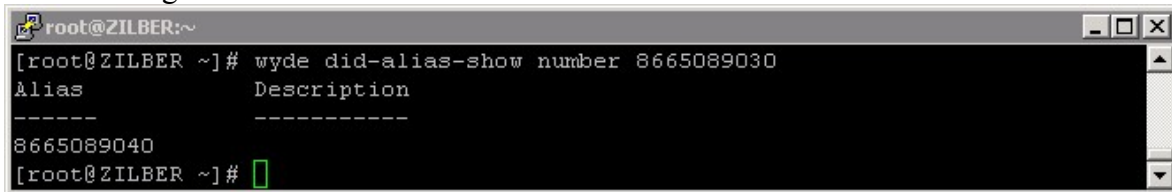
```
wyde did-alias-show number <DNIS Number>
```

where

- `<DNIS Number>` – Main dial-in number, i.e. DNIS (DID) number whose aliases you would like to show. This argument is required.

For example if you would like to see the list of aliases for DNIS number *(866) 508-9030* (that was used in previous samples) you should run the following command:
`wyde did-alias-show number 8665089030`

This command outputs all aliases with their descriptions for the specific DNIS number as shown on Figure 19.



```

root@ZILBER:~
[root@ZILBER ~]# wyde did-alias-show number 8665089030
Alias      Description
-----
8665089040
[root@ZILBER ~]#

```

Figure 19: *wyde did-alias-show* Command Output Sample

Delete a DNIS Number Alias

If you wish to delete the specific DNIS number alias, you can use the *wyde* command line utility with *did-alias-del* option. The syntax is as follows:

```
wyde did-alias-del <arguments>
```

Each of the arguments is followed by a space and a value. In *did-alias-del* you can specify the following arguments:

- `number <value>` – Main dial-in number, i.e. DNIS (DID) number which alias you wish to delete.
- `alias <value>` – The alias of the main dial-in number that you wish to delete.

Both arguments are required. The arguments can be transferred to this command in any order.

For example to delete the alias *(866) 508-9040* of DNIS number association *(866) 508-9030* (created in previous sample) you should run the command:

```
wyde did-alias-del number 8665089030 alias 8665089040
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Delete a DNIS Association

If you wish to delete the specific DNIS number association, you can use the *wyde* command line utility with *did-del* option. The syntax is as follows:

```
wyde did-del number <DNIS Number>
```

where

- `<DNIS Number>` – DNIS (DID) number you wish to delete. This argument is required.

For example to delete the DNIS number association (866) 508-9030 (created in previous sample) you should run the command:

```
wyde did-del number 8665089030
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Override a Call Flow Attribute Value for a DNIS

Call flow attributes can be overridden for a particular DNIS association. That means that some call flow attributes could be redefined on DNIS level.

To override call flow attributes for a DNIS using the command line interface you should use the *wyde* command line utility with the *did-attr-set* option. The syntax is as follows:

```
wyde did-attr-set <arguments>
```

Each of the arguments is followed by a space and a value. In *did-attr-set* you can specify the following arguments:

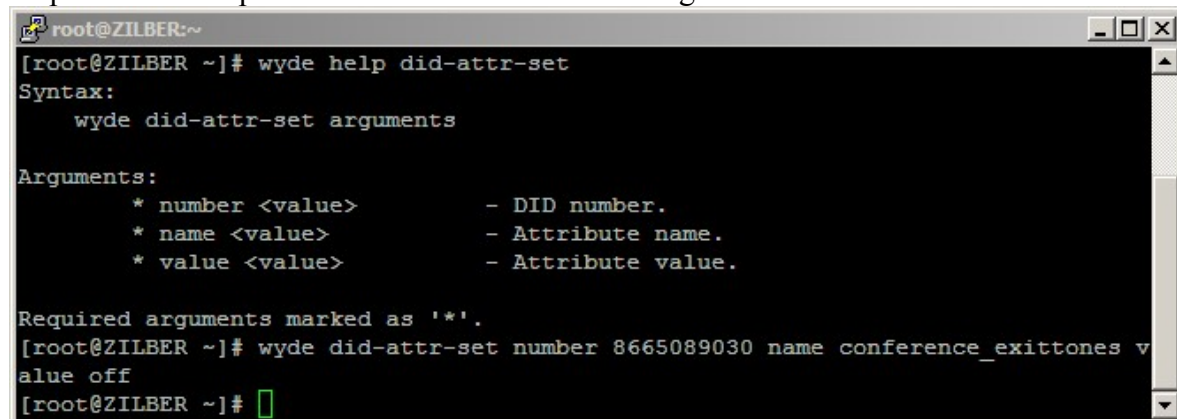
- `number <value>` – DNIS (DID) number.
- `name <value>` – This DNIS call flow attribute name you would like to override.
- `value <value>` – This DNIS call flow attribute new value that you would like to override.

All these arguments are required. The arguments can be transferred to this command in any order.

The default *conference_exittones* (exit tones) call flow attribute value for *SPECTEL* call flow is “on”. Let’s assume that we need to override its value to “off” for the DNIS (866) 508-9030 that we used in previous samples. In this case you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde did-attr-set number 8665089030 name conference_exittones  
value off
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *did-attr-set* command output and the help on this command is shown on Figure 20.



```
root@ZILBER:~  
[root@ZILBER ~]# wyde help did-attr-set  
Syntax:  
    wyde did-attr-set arguments  
  
Arguments:  
    * number <value>          - DID number.  
    * name <value>            - Attribute name.  
    * value <value>           - Attribute value.  
  
Required arguments marked as '*'.  
[root@ZILBER ~]# wyde did-attr-set number 8665089030 name conference_exittones v  
alue off  
[root@ZILBER ~]#
```

Figure 20: *wyde help did-attr-set* and *wyde did-attr-set* Commands Output Sample

If you would like to set new value for the DNIS call flow attribute that was already overridden, you can just perform another *did-attr-set* command with new attribute value.

View Call Flow Attributes Values for a DNIS

To show a list of all call flow attributes for the DNIS association (including the attributes defined on call flow level and the attributes overridden on DNIS level) using the command line, you should use the *wyde* command line utility with the *did-attr-show* option. The syntax is as follows:

```
wyde did-attr-show number <DNIS Number>
```

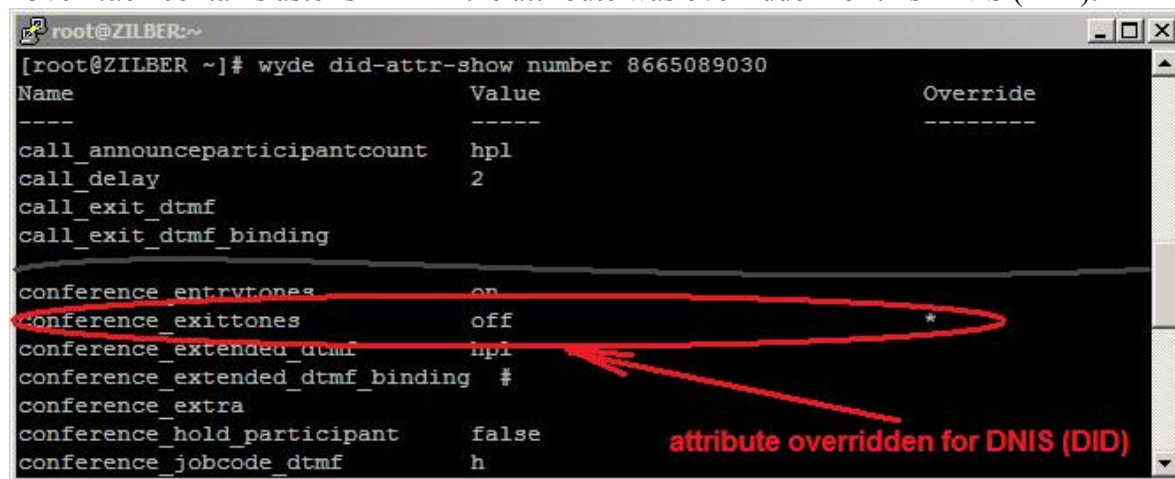
where

- *<DNIS Number>* – DNIS (DID) number which attributes you would like to view. This argument is required.

For example if you would like to see the list all call flow attributes for the DNIS (866) 508-9030 you should use the command:

```
wyde did-attr-show number 8665089030
```

This command outputs all call flow attributes for this DNIS regardless were they defined on call flow level or on DNIS level, see Figure 21 for details. As you can see, the *wyde did-attr-show* command shows the DNIS call flow attributes names and values; the last column “Override” contains asterisk “*” if the attribute was overridden for this DNIS (DID).



```
[root@ZILBER ~]# wyde did-attr-show number 8665089030
Name                                Value                                Override
-----
call_announceparticipantcount      hpl
call_delay                          2
call_exit_dtmf
call_exit_dtmf_binding
conference_entrytones               on
conference_exittones                off
conference_extended_dtmf            hpl
conference_extended_dtmf_binding    #
conference_extra
conference_hold_participant         false
conference_jobcode_dtmf             h
```

Figure 21: *wyde did-attr-show* Command Output Sample

Delete a DNIS Call Flow Attribute Redefinition

If you wish to delete the specific DNIS call flow attribute that was previously overridden, you can use the *wyde* command line utility with *did-attr-del* option. The syntax is as follows:

```
wyde did-attr-del <arguments>
```

Each of the arguments is followed by a space and a value. In *did-attr-show* you can specify the following arguments:

- *number <value>* – DNIS (DID) number which attribute you wish to delete.
- *name <value>* – This DNIS call flow attribute name you wish to delete.

Both arguments are required. The arguments can be transferred to this command in any order.

For example to delete the DNIS (866) 508-9030 call flow attribute

conference_exittones (overridden in previous sample) you should run the command:

```
wyde did-attr-del number 8665089030 name conference_exittones
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Reload All DNISes Caches

MF service (Multi Frontend Dispatcher) caches all DNIS information. As the result any DNIS changes (such as changes in call flow attributes of DNISes or changes in DNIS aliases) take effect only in about one minute after the changes have been made. If you would like that these changes take effect immediately you should reload all DNISes caches.

To send the signal on the WYDE bridge *MF* engine to reload all DNISes caches the following command should be executed:

```
wyde did-reload
```

If this command is successful, you will be returned to the command line with no additional prompts.

As alternative of this *wyde* command you can use *mf* console command *did-reload* for the same purposes. This *wyde* command is basically used in automated scripts and for convenient DNIS attributes configuration.

Conferences and Calls Management

Once a conference has begun, you can manage it using the command line interface. You can manage conferences and calls on-the-fly using the *mf* console. The *wyde* command line utility can be used only for basic tasks, such as view conferences, drop conferences and calls, and it does not provide the possibility to control the conferences and calls. Some of the information can be viewed through *Asterisk* built-in tools, as discussed later.

View Conferences and Calls in Progress

View Conferences in Progress

Any time when there are conferences in progress, you can view them in the command line interface – either via *wyde* command line utility or using *mf* console.

To show a list of all conferences that currently are in process on the bridge using the command line, you should use the *wyde* command line utility with the *show-conf* option:

```
wyde show-conf
```

You will see the screen similar to shown on Figure 22.

```

root@ZILBER:~
[root@ZILBER ~]# wyde help show-conf
Syntax:
    wyde show-conf arguments

Arguments:
    number <value>          - Conference number. Use 'confless' to show calls which hasn't placed to any conference

Required arguments marked as '*'.
[root@ZILBER ~]# wyde show-conf
Number      Members  On IVR  On MP  Flags
-----
390008      1          1       0      OB
667788      3          0       3      B
4 calls in 2 conferences.
[root@ZILBER ~]#

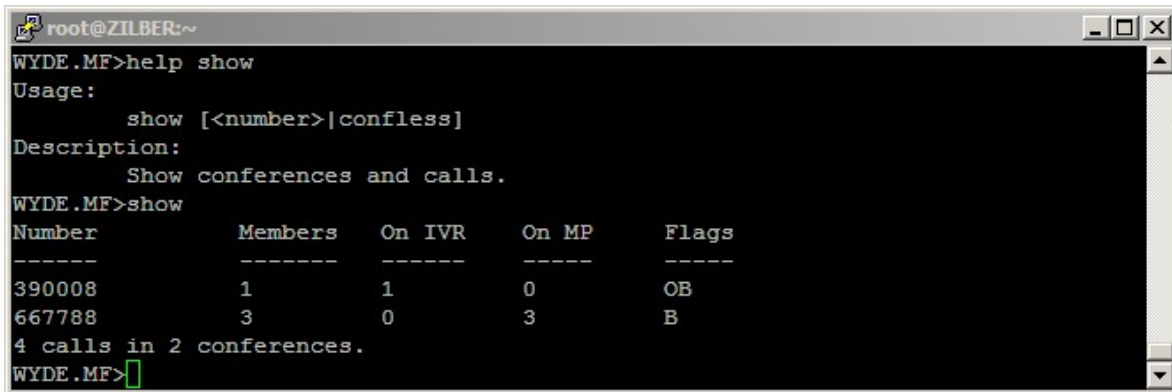
```

Figure 22: *wyde help show-conf* and *wyde show-conf* Commands Output Sample

As alternative you can use *mf* console to show all started conferences. To do so you should run the command *show* from the console:

```
show
```

You will see the screen similar to shown on Figure 23.



```

root@ZILBER:~
WYDE.MF>help show
Usage:
    show [<number>|confless]
Description:
    Show conferences and calls.
WYDE.MF>show
Number      Members  On IVR  On MP  Flags
-----
390008      1         1       0      OB
667788      3         0       3      B
4 calls in 2 conferences.
WYDE.MF>

```

Figure 23: *mf* Console *help show* and *show* Commands Output Sample

As you can see both commands have similar syntax and similar output. This *wyde* command is basically used in automated scripts, *mf* console command can be considered as primary interactive command to show started conferences. All active conferences are listed sorted ascending by conference number. There are several columns of information about each conference. Table 3 details what each column indicates.

Table 3: Show Conferences Columns

Column	Description
Number	The unique number assigned to each conference. You can use this conference number to view all calls joined to this conference
Members	Indicates how many participants are currently joined to this conference
On IVR	Indicates how many conference calls are currently processed by the frontend (IVR), for instance these calls could hear music on hold, hear any message from some service functions, etc., i.e. these calls are not participating in the conference – they could not talk and hear the conference
On MP	Indicates how many conference calls are currently processed by the Media Processor (MP), i.e. these calls are participating in the conversation – they could talk and hear each other
Flags	Indicates the mode of the conference; there could be the following conference modes: <ul style="list-style-type: none"> • B – broadcast mode (listeners can hear the conference); • O – operator mode; • Q – Q&A mode (Q&A mode is switched on for the conference); • R – the conference is being recorded; • S – secured mode (the conference is secured); • T – the conference maintains the real time (RT) protocol; Note that the conference could be in multiple modes in the same time

Number of calls on IVR plus number of calls on MP makes total number of conference calls shown in *Members* column.

View Conference Calls in Progress

Any time when there are conference calls in progress, you can view them in the command line interface – either via *wyde* command line utility or using *mf* console.

To show a list of all calls that are joined to the specific conference using the command line, you should use the *wyde* command line utility with the *show-conf* option:

```
wyde show-conf number <conference number>
```

where

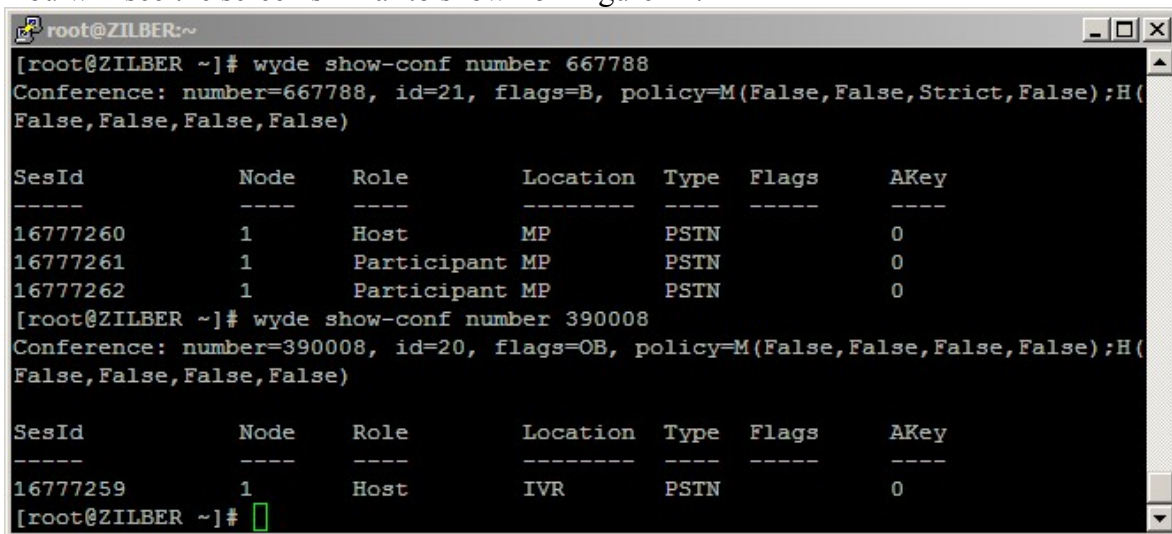
- `<conference number>` – The number of the conference which calls you would like to view.

The previous sample shows that there are two conferences on the bridge: *667788* and *390008*. If you would like to view the calls joined to these conferences you should run the commands:

```
wyde show-conf number 667788
```

```
wyde show-conf number 390008
```

You will see the screen similar to shown on Figure 24.



```

root@ZILBER:~
[root@ZILBER ~]# wyde show-conf number 667788
Conference: number=667788, id=21, flags=B, policy=M(False,False,Strict,False);H(
False,False,False,False)

SesId      Node   Role      Location  Type   Flags   AKey
-----
16777260   1      Host      MP         PSTN   0       0
16777261   1      Participant MP         PSTN   0       0
16777262   1      Participant MP         PSTN   0       0
[root@ZILBER ~]# wyde show-conf number 390008
Conference: number=390008, id=20, flags=OB, policy=M(False,False,False,False);H(
False,False,False,False)

SesId      Node   Role      Location  Type   Flags   AKey
-----
16777259   1      Host      IVR        PSTN   0       0
[root@ZILBER ~]#

```

Figure 24: *wyde show-conf* Commands Output Sample for the Specific Conferences

As alternative you can use *mf* console to show all calls that are joined to the specific conference. To do so you should run the command *show* from the console:

```
show <number>
```

where

- `<number>` – The number of the conference which calls you would like to view.

To view the calls for the same conferences *667788* and *390008* using *mf* console you should run the following commands from the console:

```
show 667788
```

```
show 390008
```

You will see the screen similar to shown on Figure 25.

```

root@ZILBER:~
WYDE.MF>show 667788
Conference: number=667788, id=21, flags=B, policy=M(False,False,Strict,False);H(
False,False,False,False)

SesId      Node   Role      Location  Type  Flags  AKey
-----
16777260   1      Host      MP        PSTN   0
16777261   1      Participant MP        PSTN   0
16777262   1      Participant MP        PSTN   0
WYDE.MF>show 390008
Conference: number=390008, id=20, flags=OB, policy=M(False,False,False,False);H(
False,False,False,False)

SesId      Node   Role      Location  Type  Flags  AKey
-----
16777259   1      Host      IVR       PSTN   0
WYDE.MF>

```

Figure 25: *mf* Console *show* Commands Output Sample for the Specific Conferences

As you can see both commands have similar output for conference calls as well. This *wyde* command is basically used in automated scripts, *mf* console command can be considered as primary interactive command to show started conference calls. The information about the conference is shown in the header of the output; it includes number of the conference (*number* field), the conference identifier (*id* field), the conference flags (*flags* field), default policy for muting (*M*) and holding (*H*) of new participants on connection to the conference (i.e. initial value of mute and hold flags for each role when the call with this role connects to the conference). All active conference calls are listed sorted ascending by session identifier. There are several columns of information about each conference call. Table 4 details what each column indicates.

Table 4: Show Conference Calls Columns

Column	Description
SesId	The session identifier of the call
Node	The frontend node identifier for the call; you can see the list of all available MF cluster nodes using <i>mf</i> console command <i>node-show</i> (Show <i>MF</i> Cluster Nodes List), see <i>mf</i> Console Command Reference for details
Role	The subscriber's (call's) role for this conference: Host, Participant or Listener
Location	Either IVR (the call is being processed by the frontend) or MP (the call is being processed by the Media Processor)
Type	Either PSTN (phone call) or VoIP (software call that supports real time protocol) or Ctrl (control call)

Column	Description
Flags	<p>Indicates the mode and state of the call; there could be the following call modes:</p> <ul style="list-style-type: none"> • H – the call is placed on hold, possible states: s – self, m – moderator; • M – the call is muted, possible states: s – self, m – moderator, q – Q&A mode is switched on; • O – operator request, possible states: w – waiting, t – talking; • Q – Q&A request, possible state: w – waiting, t – talking; • T – real time (RT) mode; <p>Note that T (real time) mode does not have states, all other modes are being characterized by mode state as well;</p> <p>Also note that the call could be in multiple modes in the same time, flags (modes) H and M could have multiple states at the same time (for example M (sm) value means that the call has muted itself and the moderator has muted the call as well, M (m) H (m) value means that the call is muted and placed on hold by moderator)</p>
AKey	The audio key of the call, 0 – if the audio key is not defined for the call

If you try to show conference calls for the conference that does not started or does not exist the error message “*Error : Conference not found*” will be returned back to you.

Show Calls that have not placed to Conferences

To show a list of all calls that are not joined to any of the conferences using the command line, you should use the *wyde* command line utility with the *show-conf* option:

```
wyde show-conf number confless
```

The keyword *confless* specifies that the command should return the list of calls that have not places to conferences. If you run this command you will see the screen similar to shown on Figure 26.

```

root@ZILBER:~
[ root@ZILBER ~]# wyde show-conf number confless
SesId      Node    Location  Type   Flags  AKey
-----
16777263    1       IVR       PSTN   ----- 0
16777264    1       IVR       PSTN   ----- 0
16777265    1       IVR       PSTN   ----- 0
16777266    1       IVR       PSTN   ----- 0
[ root@ZILBER ~]#

```

Figure 26: *wyde show-conf* Command Output Sample for the Calls that have not placed to Conferences

As alternative you can use *mf* console to show all calls that are not joined to the conferences. To do so you should run the command *show* from the console:

```
show confless
```

The command also uses keyword *confless* to specify that the command should return the list of calls that have not places to conferences. If you run this command you will see the screen similar to shown on Figure 27.

```

root@ZILBER:~
WYDE.MF>show confless
SesId      Node    Location  Type  Flags  AKey
-----
16777263   1       IVR       PSTN   -       0
16777264   1       IVR       PSTN   -       0
16777265   1       IVR       PSTN   -       0
16777266   1       IVR       PSTN   -       0
WYDE.MF>

```

Figure 27: *mf* Console *show* Command Output Sample for the Calls that have not placed to Conferences

Once again both commands have similar output for calls that have not placed to conferences. Because these calls have not connected to the conferences this view does not have *Role* column (the call has role only after it is connected to the conference).

Calls Management using *mf* Console

The basic tool to manage the conference calls is *mf* console; *wyde* command line utility provides only few possibilities to control the calls. This section explains how to manage the calls using *mf* console; the alternative *wyde* command will be noted if applicable.

Dropping Call Participants

If during a call, you wish to cancel a conference call for specific participants, you may use the *call-drop* command of *mf* console to kick someone off the call. To drop someone from a conference call you should run the following *mf* console command:

```
call-drop <conf_number> <ses_id>
```

where

- *<conf_number>* – The number of the conference which calls you would like to drop. You can use *confless* keyword instead of conference number to indicate calls which have not placed to any conference.
- *<ses_id>* – The session (i.e. call) identifier you wish to drop from the conference.

Both arguments are required.

For example if you would like to drop the session *16777261* from the conference *667788* you should run *mf* console command:

```
call-drop 667788 16777261
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”; if the session not found you will receive the message “*Error : Session not found*”.

As the alternative to this command to drop call participants you can use the *wyde* command *drop-call*; see *wyde* Command Reference for details. This *wyde* command is basically used in automated scripts only.

Mute Call Participants

While a conference call is in progress, you can mute and un-mute the specific participants and listeners using the *call-mute* command of *mf* console. That means they will be muted, but they will be able to hear the conversation. To mute someone in a conference call you should run the following *mf* console command:

```
call-mute {false|strict|relaxed} <conf_number> <ses_id>
```

where

- {false|strict|relaxed} – The argument *false* indicates that the call should be un-muted, *strict* indicates that the call should be muted and participants can not un-mute themselves, *relaxed* indicates that the call should be muted but participants can un-mute themselves.
- <conf_number> – The number of the conference which call you would like to mute or un-mute.
- <ses_id> – The session (i.e. call) identifier you wish to mute or un-mute.

All arguments are required.

For example if you would like to mute the session *16777262* in the conference *667788* without the possibility to un-mute itself you should run *mf* console command:

```
call-mute strict 667788 16777262
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”; if the session not found you will receive the message “*Error : Session not found*”.

Placing Call Participants on Hold

While a conference call is in progress, you can place the specific participants and listeners on hold or take them of hold using the *call-hold* command of *mf* console. That means they will hear music and do not hear the conversation. To place someone on hold or to take someone of hold in the conference call you should run the following *mf* console command:

```
call-hold {true|false} <conf_number> <ses_id>
```

where

- {true|false} – The argument *true* indicates that the call should be placed on hold, *false* indicates that the call should be taken of hold.
- <conf_number> – The number of the conference which call you would like to place on hold or to take of hold.
- <ses_id> – The session (i.e. call) identifier you wish to place on hold or to take of hold.

All arguments are required.

For example if you would like to place the session *16777262* in the conference *667788* on hold you should run *mf* console command:

```
call-hold true 667788 16777262
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”; if the session not found you will receive the message “*Error : Session not found*”.

Set Custom Name for a Call

While a conference call is in progress, you can define the custom name for any conference participants using the *call-custom-name* command of *mf* console. To change the custom name for someone in the conference call you should run the following *mf* console command:

```
call-custom-name <conf_number> <ses_id> [<name>]
```

where

- *<conf_number>* – The number of the conference for which call you wish to define the custom name.
- *<ses_id>* – The session (i.e. call) identifier for which you wish to define the custom name.
- *<name>* – new custom name for the call, if this argument is omitted the empty custom name will be set to this call.

Arguments *<conf_number>* and *<ses_id>* are required.

For example if you would like to define the custom name *Guest* for the session *16777262* in the conference *667788* you should run *mf* console command:

```
call-custom-name 667788 16777262 Guest
```

If you would like to remove the custom name for this call, i.e. to define the empty custom name for it you should run *mf* console command:

```
call-custom-name 667788 16777262
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”; if the session not found you will receive the message “*Error : Session not found*”.

Set Audio Key for a Call

While a conference call is in progress, you can define the audio key for any conference call using the *call-associate* command of *mf* console. The calls with the same audio key are being grouped into the same bundle. To change the audio key for someone in the conference call you should run the following *mf* console command:

```
call-associate <conf_number> <ses_id> [<audiokey>]
```

where

- *<conf_number>* – The number of the conference for which call you wish to define the audio key, i.e. the bundle.
- *<ses_id>* – The session (i.e. call) identifier for which you wish to define the audio key (bundle).
- *<audiokey>* – new audio key for the call, if this argument is omitted the audio key for this call will be set to 0, i.e. the call in this case will not belong to any bundle.

Arguments *<conf_number>* and *<ses_id>* are required.



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.

For example if you would like to define the audio key 886 for the session 16777287 in the conference 758288 you should run *mf* console command:

```
call-associate 758288 16777287 886
```

If you would like to remove the audio key for this call, i.e. to exclude the call from any bundle you should run *mf* console command:

```
call-associate 758288 16777287
```

As you can see on Figure 28 after assigning the audio key to the voice call, both calls – the control and the PSTN – have the same audio key; *AKey* column after the command execution has 886 value for both records; that means both these calls belong to the same bundle. After removing of the audio key from the voice call the *AKey* column shows 0 value for the voice call, the control call still have the audio key 886.

```

root@ZILBER:~
WYDE.MF>call-associate 758288 16777287 886
Success
WYDE.MF>show 758288
Conference: number=758288, id=32, flags=TB, policy=M(False,False,Strict,False);H
(False,False,False,False)

SesId      Node   Role   Location  Type  Flags  AKey
-----
16777286   1      Host   IVR        Ctrl  T      886
16777287   1      Host   IVR        PSTN  886
WYDE.MF>call-associate 758288 16777287
Success
WYDE.MF>show 758288
Conference: number=758288, id=32, flags=TB, policy=M(False,False,Strict,False);H
(False,False,False,False)

SesId      Node   Role   Location  Type  Flags  AKey
-----
16777286   1      Host   IVR        Ctrl  T      886
16777287   1      Host   IVR        PSTN  0
WYDE.MF>

```

Figure 28: *mf* Console *call-associate* Commands Output Sample

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”; if the session not found you will receive the message “*Error : Session not found*”.

Conferences Management using *mf* Console

The basic tool to manage the conferences is *mf* console; *wyde* command line utility provides only few possibilities to control them. This section explains how to manage the conferences using *mf* console; the alternative *wyde* command will be noted if applicable.

Dropping a Conference

If during a call, you wish to cancel the entire conference for all users, you may use the *conf-drop* command of *mf* console to end the conference. To drop a conference you should run the following *mf* console command:

```
conf-drop <conf_number>|all  
where
```

- *<conf_number>* – The number of the conference you would like to drop. You can use *all* keyword instead of conference number to indicate that all conferences started on the bridge should be dropped. This argument is required.

For example if you would like to drop the conference 667788 you should run *mf* console command:

```
conf-drop 667788
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”.

As the alternative to this command to drop call participants you can use the *wyde* command *drop-conf*; see *wyde* Command Reference for details. This *wyde* command is basically used in automated scripts only.

Moreover you can drop the individual conference participants as it was previously described in section: Dropping Call Participants.

Conference Mute and Q&A Modes

The conference can be in one of the following muting modes that determine how the conference is muted. The conference modes are:

- *open* – Anyone can talk and un-mute themselves.
- *relaxed* – All participants are muted by default, but they can un-mute themselves, using default *6 on their DTMF keypad. Hosts are not muted.
- *strict* – None of the participants can un-mute themselves. The only way to un-mute a caller is through the command line interface or by the host. In this mode, when participant tries to un-mute using default *6 – the system tells “conference host has muted the conference; this line can not be unmuted”.
- *question* – The conference is in Q&A mode. In this mode, if participant presses *6 the system tells “If you’d like to ask a question – press 1”. Q&A Sessions will be described later in the next section of this guide.

If during a call, you wish to mute or un-mute all hosts or all participants in the conference you may use the *conf-mute-group* command of *mf* console to mute them. To mute/un-mute the group of callers you should run the following *mf* console command:

```
conf-mute-group {false|strict|relaxed} {host|participant}  
    <conf_number>
```

where

- {*false|strict|relaxed*} –
 - *false* indicates that the group (hosts or participants) should be un-muted – the conference mode will be *open*;
 - if the next argument is *host*, *strict* or *relaxed* indicates that the hosts should be muted, but they can un-mute themselves;
 - if the next argument is *participant*, *strict* indicates that all participants should be muted and they can not un-mute themselves – the conference mode will be *strict*;
 - if the next argument is *participant*, *relaxed* indicates that all participants should be muted but they can un-mute themselves – the conference mode will be *relaxed*.
- {*host|participant*} – Denotes who (all hosts or all participants) should be muted or un-muted.
- <conf_number> – The number of the conference which calls you wish to mute or un-mute.

All arguments are required.

For example if you would like to mute all *participants* in the conference 667788 without the possibility to un-mute themselves you should run *mf* console command:

```
conf-mute-group strict participant 667788
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”, if you transferred wrong argument (not *false*, *strict*, *relaxed*, *host*, *participant*) you will receive the message “*Error : Wrong argument*”.

Moreover you can mute/unmute the individual conference participants as it was previously described in section: Mute Call Participants.

Q&A Sessions

The WYDE conferencing software has a feature built in known as Q&A (questions and answers). You can set up and manage a Q&A session using the command line interface *mf* console that allows conference participants to, in turn, ask questions and receive answers. When Q&A session is started all participants will be muted and they will be unable to unmute their self, but they will be able to request to speak (to request question). You as the host can allow them to speak using *mf* console. After they have finished speaking you can drop them out of the Q&A queue or you can drop the user from the Q&A queue without allowing them to speak using *mf* console commands.

The Q&A also can be managed using DTMF keypad and the Web Administration Interface as it is described in “Web Administration Interface – User Guide”.

To start or stop Q&A mode for the conference and to clear Q&A queue you should run the following *mf* console command:

```
conf-qa-mode {start|stop|clear} <conf_number>
```

where

- `{start|stop|clear}` – *start* indicates that Q&A session should be started for the conference, i.e. the conference mode should be *question*; *stop* indicates that Q&A session should be stopped for the conference; *clear* indicates that Q&A queue should be cleared for the conference. i.e. all participants should be removed from Q&A queue.
- `<conf_number>` – The number of the conference for which you would like to manage the Q&A session.

All arguments are required.

To request to speak and to cancel this request within Q&A session you should run the following *mf* console command:

```
call-qa-request {start|stop} <conf_number> <ses_id>
```

where

- `{start|stop}` – *start* indicates that the call should start the request to speak (the request to ask the question), i.e. placed into Q&A queue; *stop* indicates that the call should stop (cancel) the request to speak, i.e. removed from Q&A queue.
- `<conf_number>` – The number of the conference where Q&A session is started.
- `<ses_id>` – The call session identifier that should be placed into Q&A queue or removed from it.

All arguments are required.

To enable Q&A session for the specific call from Q&A queue, i.e. to unmute this call and to disable Q&A session for the specific call, i.e. to mute this call you should run the following *mf* console command:

```
call-qa-talk {enable|disable} <conf_number> <ses_id>
```

where

- `{enable|disable}` – *enable* indicates that the call from Q&A queue should be unmuted and the participants should be able to speak (to ask his question); *disable* indicates that the unmuted call should be muted and removed from Q&A queue.
- `<conf_number>` – The number of the conference where Q&A session is started.
- `<ses_id>` – The call session identifier that should be unmuted to ask his question or should be muted again and/or removed from Q&A queue.

All arguments are required.

To move to the next questioner from Q&A queue, i.e. to allow speaking (enable Q&A session) for the first call in the queue you should run the following *mf* console command:

```
conf-qa-talk <conf_number>
```

where

- `<conf_number>` – The number of the conference where Q&A session is started and where you would like to enable (unmute) the first call in the Q&A queue. The active questioner (if exists) will be removed from Q&A queue, so this command ends Q&A session for the current questioner and starts it for the next one. This argument is required.

To mute and unmute the active Q&A session you should run the following *mf* console command:

```
conf-qa-mute {true|false} <conf_number>
```

where

- {true|false} – *true* indicates that the active Q&A session should be muted; *false* indicates that the active Q&A session should be unmuted (*). Both these options do not remove the questioner from Q&A queue; the command just temporary allows/disallows the active questioner to speak.
- <conf_number> – The number of the conference where Q&A session is started and where you would like to mute/unmute the active Q&A session (*).

All arguments are required.

Let's assume that we are maintaining Q&A session in the conference 667788 and there are two participants' sessions in this conference: 16777302 and 16777303. Table 5 shows samples how to manage this conference Q&A sessions using *mf* console commands.

Table 5: Q&A Sessions Management Samples Using *mf* Console Commands

Description	<i>mf</i> Console Command	DTMF
Start Q&A mode for the conference	conf-qa-mode start 667788	*1 1
Request to speak (question) for specific calls, i.e. place the calls to Q&A queue	call-qa-request start 667788 16777302 call-qa-request start 667788 16777303	*6
Cancel the request to speak for specific calls	call-qa-request stop 667788 16777303	
Clear Q&A queue in the conference	conf-qa-mode clear 667788	*1 5
Enable Q&A session for the specific call from Q&A queue, i.e. to allow him to speak	call-qa-talk enable 667788 16777302	
Disable Q&A session for the specific call, mute the call, and remove participant from Q&A queue	call-qa-talk disable 667788 16777302	
Enable Q&A session for the first call in Q&A queue, i.e. move to the next questioner	conf-qa-talk 667788	*1 2
Mute the active questioner	conf-qa-mute true 667788	*1 4
Unmute the active questioner	conf-qa-mute false 667788	*1 4
Stop Q&A mode for the conference	conf-qa-mode stop 667788	*1 3

Placing a Conference on Hold

While a conference is in progress, you can place all participants and listeners on hold or take them of hold (i.e. place them on line) using the *conf-hold-group* command of *mf* console. That means they will hear music and do not hear the conversation while hosts have a private discussion. To place all participants or all listeners on hold or to take them of hold you should run the following *mf* console command:

```
conf-hold-group {true|false} {participant|listener}
<conf_number>
```

where

- `{true|false}` – The argument *true* that the group (participants or listeners) should be placed on hold, *false* indicates that the group (participants or listeners) should be taken of hold.
- `{participant|listener}` – Denotes who (all participants or all listeners) should be placed on hold or taken of hold.
- `<conf_number>` – The number of the conference which calls you would like to place on hold or to take of hold.

All arguments are required.

For example if you would like to place on hold *participants* in the conference *667788* you should run *mf* console command:

```
conf-hold-group true participant 667788
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”, if you transferred wrong argument (not *true*, *false*, *participant*, *listener*) you will receive the message “*Error : Wrong argument*”.

Moreover you can place on hold and take of hold the individual conference participants as it was previously described in section: Placing Call Participants on Hold.

Broadcast a Conference for Listeners

While a conference is in progress you may need to start or stop conference broadcasting for listeners. When the broadcasting is stopped all listeners are on hold, i.e. hear music on hold, and do not hear the conference. To start/stop the conference broadcast mode for listeners you should use the *conf-broadcast* command of *mf* console:

```
conf-broadcast {start|stop} <conf_number>
```

where

- `{start|stop}` – Denotes should the conference be broadcasted to listeners or not: *start* indicates that the listeners should hear the conference; *stop* indicates that all listeners should be on hold and do not hear the conference.
- `<conf_number>` – The number of the conference you wish to broadcast.

All arguments are required.

For example if for the conference *667788* you would like to *stop* broadcasting for listeners you should run *mf* console command:

```
conf-broadcast stop 667788
```

All listeners will be placed on hold as the result of this command execution. If you would like to *start* the conference broadcasting for them again you should run *mf* console command:

```
conf-broadcast start 667788
```

All listeners will be online as the result of this command execution, i.e. they will hear the conference.

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference*”.

not found”, if you transferred wrong argument (neither *start*, nor *stop*) you will receive the message “*Error : Wrong argument*”.

Making a Conference Secure

While a conference is in progress, from time to time you may want to make a call secure. That is, make it so that no other participants or listeners can join to the conference (hosts still can join to secured conferences). To do this you should use the *conf-secure* command of *mf* console:

```
conf-secure {secure|unsecure} <conf_number>  
where
```

- {secure|unsecure} – Denotes should the conference be made secured or unsecured.
- <conf_number> – The number of the conference you wish to make secured or unsecured.

All arguments are required.

For example if you would like to make the conference 667788 *secured* you should run *mf* console command:

```
conf-secure secure 667788
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”, if you transferred wrong argument (neither *secure*, nor *unsecure*) you will receive the message “*Error : Wrong argument*”.

Setting a Conference Job Code

While a conference is in progress, you can define the conference job code using the *conf-jobcode* command of *mf* console. When any of the calls ends the current conference job code will be stored in the CDR record of the call; for example this job code can be used in the calls reporting. To set the conference job code you should run the following *mf* console command:

```
conf-jobcode <conf_number> [<code>]  
where
```

- <conf_number> – The number of the conference which job code you would like to set. This argument is required.
- <code> – New job code for the conference. If this argument is omitted there will be no job code stored in CDR records of the calls.

For example if you would like to set job code 8899 for the conference 667788 you should run *mf* console command:

```
conf-jobcode 667788 8899
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”.

Recording a Conference

While a conference is in progress, you can record the conference using the *conf-recording* command of *mf* console. The recorded conferences can be playback as it will be described later in this guide. To start/stop the conference recording you should run the following *mf* console command:

```
conf-recording {start|stop} <conf_number> [<accesscode>]
```

where

- {start|stop} – Denotes should the conference recording be started or stopped.
- <conf_number> – The number of the conference you wish to record.
- <accesscode> – Denotes pin, i.e. password to the recording server if “*Recording method*” call flow attribute value is “*remote*” (this value can be either defined on call flow level or overridden on DNIS level).

Arguments {start|stop} and <conf_number> are required. The argument <accesscode> should be used only if “*Recording method*” call flow attribute value is “*remote*”.

For example if you would like to *start* the recording for the conference 667788 you should run *mf* console command:

```
conf-recording start 667788
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”, if you transferred wrong argument (neither *start*, nor *stop*) you will receive the message “*Error : Wrong argument*”.

Playing an Audio File to a Conference

While a conference is in progress, you can playback the audio file to the conference using the *conf-play-file* command of *mf* console. Using this command you can either playback previous this conference recordings or playback the files that were previously uploaded via web interface. To playback the file you usually need to use the control call to this conference and play this file via such control call. To manage of playing the audio file to the conference you should run the following *mf* console command:

```
conf-play-file <conf_number> <ses_id> {assign <dir>  
    <filename>|start|stop|seek <offset> <whence>}
```

where

- <conf_number> – The number of the conference where you would like to play the audio file.
- <ses_id> – The session identifier (usually the identifier of the control call) which should be used to play the audio file.
- {assign <dir> <filename>|start|stop|seek <offset> <whence>} – One of the following arguments should be specified here:

- 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.
- *start* – Start the playback from the current position.
- *stop* – Stop the playback.
- *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.

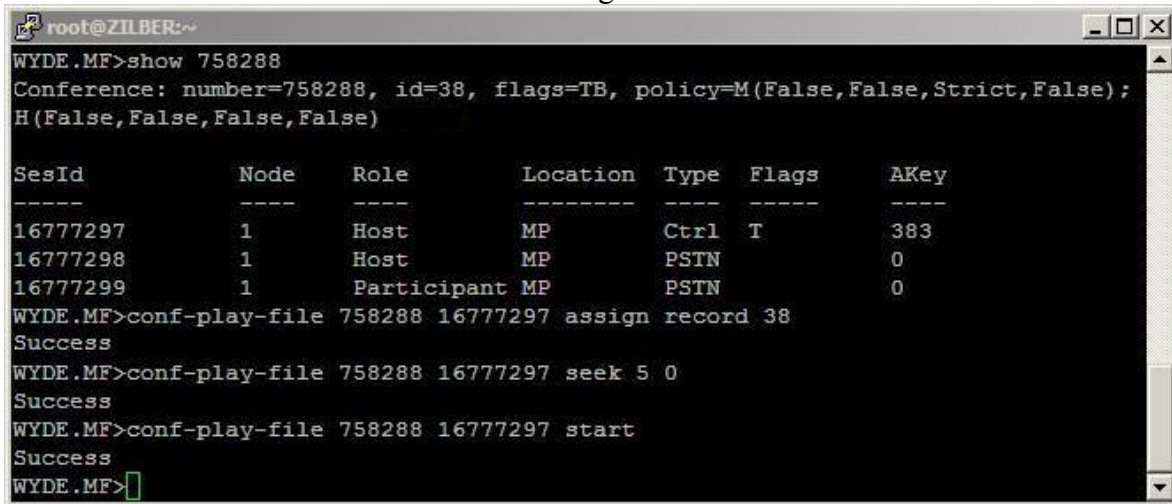
All arguments are required.

Let's assume that you should implement the following scenario: for the conference 758288 you would like to play the conference recording file 38 (actually this is the conference identifier that is used as conference recording file name), i.e. you need to *assign* this file to the control call; after that you need to *seek* the audio file on 5 seconds starting from the *beginning* of the file; and finally you need to *start* the playback from the current position.

To implement this scenario you should run the following *mf* console commands:

```
conf-play-file 758288 16777297 assign record 38
conf-play-file 758288 16777297 seek 5 0
conf-play-file 758288 16777297 start
```

You will see the screen similar to shown on Figure 29.



```

root@ZILBER:~
WYDE.MF>show 758288
Conference: number=758288, id=38, flags=TB, policy=M(False,False,Strict,False);
H(False,False,False,False)

SesId      Node   Role    Location  Type  Flags  AKey
-----
16777297    1     Host    MP         Ctrl  T      383
16777298    1     Host    MP         PSTN   0
16777299    1     Participant MP         PSTN   0
WYDE.MF>conf-play-file 758288 16777297 assign record 38
Success
WYDE.MF>conf-play-file 758288 16777297 seek 5 0
Success
WYDE.MF>conf-play-file 758288 16777297 start
Success
WYDE.MF>
  
```

Figure 29: *mf* Console *conf-play-file* Commands Output Sample

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”; if the session not found you will receive the message “*Error : Session not found*”.

Dialing another User

Often, there will be a conference call in progress and you will need to call another participant. That is where the dialout feature comes in. If your system supports dialing outside users, you can dial-out another user using the *dialout* command of *mf* console. To dial to another user from the started conference you should run the following *mf* console command:

```
dialout <peer_number> <timeout> <originator_conf_number>
      <did_number> <accesscode> [<role>]
```

where

- `<peer_number>` – The phone number you wish to dial.
- `<timeout>` – Allowed maximal timeout in seconds, i.e. how many seconds the call can wait the answer.
- `<originator_conf_number>` – The original conference number, i.e. the conference from which you would like to make dial-out, this conference must be started prior to dial-out.
- `<did_number>` – The target conference DNIS (DID) number where the call should be joined after the dial-out is complete.
- `<accesscode>` – The access code that should be used to join to the target conference.
- `<role>` – the role that will be granted to the call when it joins to the target conference (applicable only for call flows without authorization, for example *CONF* call flow).

The argument `<role>` is optional and should be used for call flows without authorization only. All other arguments are required.

Note that this command allows making dial-out and connecting the call to another conference; the target conference is being defined by the pair: access code and role. While dialout is in progress the conference defined in the argument `<originator_conf_number>` contains the call that performing dialout; when dialout is complete the call is being joined to the conference defined by access code and role.

Nevertheless while the conference is in progress if you need to see its dialout attributes, i.e. DNIS numbers and access codes that could be used in *dialout* command and that were used by the calls that already connected to this conference, you can use the following *mf* console command:

```
dialout-attr <originator_conf_number>
```

where

- `<originator_conf_number>` – the number of the started conference which dialout attributes (DNISes and access codes) you would like to show. This argument is required.

This command returns the list of all used DNIS numbers and access codes with roles for the requested started conference:

```
DID: <did1>[,<did2>[,<did3>[,...]]]
```

```
AccessCode: <role1>:<access code1>
```

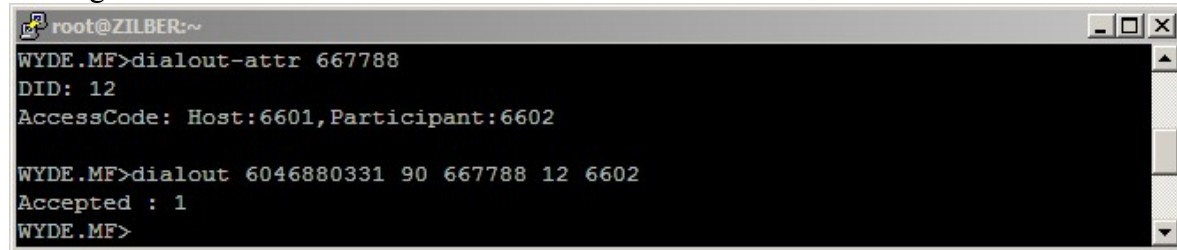
```
    [,<role2>:<access code2>[,<role3>:<access code3>[,...]]]
```

For example if you would like

- to show DNIS numbers and access codes for the conference 667788;
 - to dialout from this conference to phone number 6046880331, timeout 90 seconds, and connect this call to the same conference using DNIS number 12 and access code 6602
- you should run *mf* console commands:

```
dialout-attr 667788
dialout 6046880331 90 667788 12 6602
```

See Figure 30 for details.



```
root@ZILBER:~
WYDE.MF>dialout-attr 667788
DID: 12
AccessCode: Host:6601, Participant:6602

WYDE.MF>dialout 6046880331 90 667788 12 6602
Accepted : 1
WYDE.MF>
```

Figure 30: *mf* Console *dialout-attr* and *dialout* Commands Output Sample

The command *dialout* is asynchronous command; because of that *mf* console is unable to return its result immediately after the command was started. When you run this command, the system returns the message: “*Accepted : <taskId>*”, where *<taskId>* – is identifier for the dialout task. If the command was implemented successfully you will receive the message: “*Notify : <taskId> : Success*”, where *<taskId>* – the same task identifier that was assigned to dialout task and was shown in “*Accepted*” message; if the was any error during the command execution you will receive the message: “*Notify : <taskId> : Error : <error message>*”, where *<taskId>* – once again the same task identifier and *<error message>* – the specific error message (for example “*Conference not found*”, etc.).

Note that the WYDE bridge should be configured to perform dialout; this dialout configuration is described later in this Guide in section: Dialout Settings Configuration.

Move a Call to another Conference

From time to time you may need to move the caller from one started conference to another conference. That means that the call is being removed from his current conference and moved to another conference. The target conference could be either started or it could be not started, but it becomes started during this move. To make such move you should run the following *mf* console command:

```
call-move <conf_number> <ses_id> <new_did_number>
         <new_accesscode> [<new_role>]
```

where

- *<conf_number>* – The source conference number whose call you would like to move to another conference.
- *<ses_id>* – The call session identifier that you wish to move to another conference.
- *<new_did_number>* – New (i.e. target) conference DNIS (DID) number where the call should be moved.

- `<new_accesscode>` – The access code that should be used to join to new (i.e. target) conference.
- `<new_role>` – The role that will be granted to the call when it joins to new conference. This argument is applicable only for call flows without authorization, for example *CONF* call flow.

The argument `<new_role>` is optional and should be used for call flows without authorization only. All other arguments are required.

Let's review the following scenario: there are two conferences on the bridge – the conference *667788* with the call session *16777325* and the conference *758288* with the call session *16777324*; we need to move the call *16777324* from the second conference to the first one; DNIS (DID) number to connect to the conference *667788* is *12* and the access code is *6601* (for hosts). To implement this scenario the following command should be run:

```
call-move 758288 16777324 12 6601
```

See Figure 31 for details. In addition this figure shows what conferences and calls were on the bridge prior to this command call and after this command call.

```

root@ZILBER:~
WYDE.MF>show
Number          Members    On IVR    On MP    Flags
-----
667788          1          1         0        B
758288          1          1         0        TB
2 calls in 2 conferences.
WYDE.MF>show 667788
Conference: number=667788, id=49, flags=B, policy=M(False,False,Strict,False);H(
False,False,False,False)

SesId          Node    Role        Location    Type    Flags    AKey
-----
16777325       1      Participant  IVR         PSTN     0
1 calls
WYDE.MF>show 758288
Conference: number=758288, id=48, flags=TB, policy=M(False,False,Strict,False);
H(False,False,False,False)

SesId          Node    Role        Location    Type    Flags    AKey
-----
16777324       1      Host        IVR         PSTN     0
1 calls
WYDE.MF>call-move 758288 16777324 12 6601
Accepted : 6
Notify : 6 : Success
WYDE.MF>show
Number          Members    On IVR    On MP    Flags
-----
667788          2          0         2        B
2 calls in 1 conferences.
WYDE.MF>show 667788
Conference: number=667788, id=49, flags=B, policy=M(False,False,Strict,False);H(
False,False,False,False)

SesId          Node    Role        Location    Type    Flags    AKey
-----
16777324       1      Host        MP          PSTN     0
16777325       1      Participant  MP          PSTN     0
WYDE.MF>

```

Figure 31: *mf* Console *call-move* Command Output Sample

The command *call-move* is asynchronous command; because of that *mf* console is unable to return its result immediately after the command was started. When you run this command, the system returns the message: “Accepted : <taskId>”, where <taskId> – is identifier for the call move task. If the command was implemented successfully you will receive the message: “Notify : <taskId> : Success”, where <taskId> – the same task identifier that was assigned to the move task and was shown in “Accepted” message; if the was any error during the command execution you will receive the message: “Notify : <taskId> : Error : <error message>”, where <taskId> – once again the same task identifier and <error message> – the specific error message (for example “Conference not found”, “Session not found”, “DID” – if DNIS number specified in <new_did_number> argument does not exist or incorrect, “AccessCode” – if the access code in <new_accesscode> argument does not exist or incorrect, etc.).

Making a Shunt between Two Conferences

While two conferences are in progress, you may allow to the callers from one of them hearing the callers from another and vice versa using the *conf-shunt* command of *mf* console. The shunt (cross join bridge) is being created between these two conferences and all callers from these conferences can hear each other, but both these conferences still remain. To start/stop the shunt between two conferences you should run the following *mf* console command:

```
conf-shunt {start <conf_number> [<peer_conf_number>|
                                stop <conf_number>}
```

where

- {start|stop} – *start* denotes that the shunt between two conferences should be made (started). *stop* denotes that the shunt between two conferences should be dropped (stopped). This argument is required.
 - <conf_number> – The number of the conference you wish to shunt or unshunt. This argument is required.
 - <peer_conf_number> – The number of the peer conference you wish to shunt with the first one. This argument should be specified for *start* option only.

For example if you would like to *start* shunt between two conferences: 667788 and 758288, i.e. to allow these conferences hearing each other, you should run *mf* console command:

```
conf-shunt start 667788 758288
```

If you would like to *stop* shunt between these two conferences you should run *mf* console command:

```
conf-shunt stop 667788
```

If the command was implemented successfully you will receive the message: “*Success*”; if any of the conferences not found or not started you will receive the message “*Error : Conference not found*”, if you transferred wrong argument (neither *start*, nor *stop*) you will receive the message “*Error : Wrong argument*”, if the shunt already exist for the conference you will receive the message “*Error : Shunt is already exists for this conference.*”.

Polling

While a conference is in progress, you can start polling (voting) for the conference using the *conf-polling* command of *mf* console. While the polling is in progress all conference participants can vote, i.e. select one of the available options defined when the polling was started. If you need detail information about the polling process, including information how to get polling results please read “Web Administration Interface – User Guide”, section: Polling. To start/stop the conference polling you should run the following *mf* console command:

```
conf-polling <conf_number> {start <keys>|stop}
```

where

- <conf_number> – the number of the conference for which you would like to start or to stop the polling;

- {start <keys>|stop} – *start* indicates that the polling should be started for the conference, *stop* indicates that the polling should be stopped for the conference;
 - <keys> – available polling options (i.e. digits 1, 2, ..., 9, 0) that should be specified when the polling is started.

All arguments are required.

For example if you would like to *start* the polling for the conference 667788 and options 1, 2, and 3 should be available for voting, you should run *mf* console command:

```
conf-polling 667788 start 123
```

If you would like to *stop* the polling for this conference you should run *mf* console command:

```
conf-polling 667788 stop
```

If the command was implemented successfully you will receive the message: “*Success*”; if the conference not found or not started you will receive the message “*Error : Conference not found*”, if you transferred wrong argument (neither *start*, nor *stop*) or if you do not specify the polling options when you starting the polling you will receive the message “*Error : Wrong argument*”.

Scheduling

Any conference could be scheduled to take place at specific date (date range), time, week day (DOW), with specified number of the conference participants. The following call flow attributes are responsible for the conference scheduling:

- *conference_scheduled* (Conference scheduled) – denotes is the conference scheduled or not, this attribute value should be equal “on” for the scheduled conferences, if this attribute value is “off” all scheduling attributes are being ignored;
- *conference_schedule* (Conference schedule) – denotes date (date range), time, day of the week when the conference is scheduled and how many participants can be joined to this scheduled conference, this attribute has the following format:

```
<scheduling 1>[;<scheduling 2>[;<scheduling 3>[;...]]]
```

i.e. this is the list of the conference scheduling records separated with semicolon (;); each scheduling record has the following format:

```
<part_count>,<days>,<tm_begin>,<duration>,<date_begin>,<date_end>
```

where

- *part_count* – number of allowed the scheduled conference participants;
- *days* – day of weeks when the scheduled conference is allowed: 1 (Sunday), 2 (Monday), ..., 7 (Saturday);
- *tm_begin* – time when the scheduled conference begins in *hhmm* format;
- *duration* – the scheduled conference duration in minutes;
- *date_begin* – the beginning date in *YYMMDD* format when the conference scheduling is active (i.e. the first date when the scheduling is on);
- *date_end* – the ending date in *YYMMDD* format when the conference scheduling is active (i.e. the last date when the scheduling is on)

For example if you need to schedule the conference for 20 participants on Monday, Wednesday, Thursday, from 10:00 a.m. for 90 minutes (i.e. till 11:30 a.m.), for the date range from November 1st, 2009 till March 1st, 2010 the attribute value should be:

20,245,1000,90,091101,100301

If in addition to previous scheduling you need to schedule the conference for 30 participants on November 11th, 2009 (Wednesday) the attribute value should be:

20,245,1000,90,091101,091110;30,4,1000,90,091111,091111;20,245,1000,90,091112,100301

- `conference_schedule_extend_dtmf` (Schedule conference extend DTMF policy) – denotes who (h/p/l, default: h – host) can extend duration of the scheduled conference;
- `conference_schedule_extend_dtmf_binding` (Schedule conference extend DTMF binding) – denotes what keys should be used to extend duration of the scheduled conference, when these keys are pressed the following message will be played: *“please press 1 to extend conference duration for 5 minutes, press 2 – for 10 minutes, press * – to return to the conference”*, the total maximal extension time is defined in the attribute `conference_schedule_extend`;
- `conference_schedule_extend` (Schedule conference extend maximal time in seconds) – denotes to what maximal time in seconds the duration of the scheduled conference can be extended;
- `conference_schedule_remind` (Schedule conference end remind time in seconds) – denotes remind time in seconds before the scheduled conference ends when the reminder (warning) message should be played to moderator, for instance if this attribute value is 600 before 10 minutes (600 seconds) till the scheduled conference ending time the remind message will be played to the host;
- `conference_schedule_hold` (Wait before scheduled conference begin) – denotes wait time in seconds before the scheduled conference begins when the participants can join; if the participant connected prior to this time his call will be dropped, if he connected after this time but prior the conference begin time music-on-hold will be played to him until the conference starts, for instance if the conference is scheduled to be started at 11:00 and this parameter values is 600 (i.e. 10 minutes) all participants’ calls earlier than 10:50 will be dropped with the message that the conference is scheduled to be started at 11:00, the participants’ calls from 10:50 till 11:00 will be placed on hold (music on hold will be played) until the conference begins.

The following conference scheduling logic is being implemented:

- if `conference_scheduled` attribute is “off”, the conference is not scheduled (the processing logic is usual, as for non-scheduled conferences);
- if `conference_schedule` attribute is empty, the conference is not scheduled;
- otherwise, if `conference_scheduled` attribute is “on” and `conference_schedule` attribute describes conference scheduling, the conferences are being considered as scheduled and the following logic will be applied:
 - if there are no any scheduled conferences for the nearest 30 days, the call will be dropped with the message *“This conference has no schedule. Please check with the organizer. Good bye.”*;

- if for the current day of week all scheduled conferences are already completed, i.e. the current time is more than ending time for the last scheduled conference, the call will be dropped with the message *“Your conference is not running or scheduled. Please check its schedule with the organizer. Good bye.”*;
- if till the scheduled conference beginning time (<tm_begin>) more than hour the message *“There is no upcoming conference scheduled within an hour. Please check the schedule with the organizer. Good bye.”* will be played and the call will be dropped;
- if till scheduled conference beginning time (<tm_begin>) less than hour, but more than period defined in `conference_schedule_hold` attribute value the message *“Your conference will begin in <NNN> minutes. Please call again later. Good bye.”* (where <NNN> – minutes till the conference beginning time) will be played and the call will be dropped;
- if till scheduled conference beginning time (<tm_begin>) less than `conference_schedule_hold` attribute value the message *“Please wait. Your conference will begin shortly. This line is now on hold.”* will be played and the call will be placed on hold (music on hold will be played); at conference beginning time (<tm_begin>) the message *“Your conference is started. Please wait while we place you into the conference.”* will be played to inform that the conference is ongoing;
- if the new call exceeds <part_count> amount the call will be dropped with the message *“This conference is already full. Please call again later. Good bye.”*, the host also receives the warning about that;
- at `conference_schedule_remind` seconds before the conference ending time of the scheduled conference, the host receives announcement *“This conference has <MMM> more minutes to complete. Press <KEY> to request an extension.”* (where <MMM> – minutes till the conference ending time, <KEY> – keys defined in `conference_schedule_extend_dtmf_binding` call flow attribute value that should be used on DTMF keypad by the host if he wishes to extend the conference duration); the host can extend the conference duration, the maximal extension time is defined in `conference_schedule_extend` attribute value;
- when the scheduled conference time is over the message *“This conference has run out of time. All calls will now end.”* will be played and all this conference calls will be dropped.

While the scheduled conference is in progress you can use *mf* console to extend conference duration using the *conf-schedule-extend* command and to increase number of allowed participants using the *conf-schedule-incsize* command. To extend the started scheduled conference duration you should run the following *mf* console command with two required arguments:

```
conf-schedule-extend <conf_number> <seconds>
```

where

- <conf_number> – The number of the started scheduled conference which duration you would like to extend.
- <seconds> – The increment in seconds that should be added to the allowed duration for this scheduled conference. Note that the call flow attribute

`conference_schedule_extend` (Schedule conference extend maximal time in seconds) denotes to what maximal time in seconds the duration of the scheduled conference can be extended and you are unable to extend duration more than specified in this attribute.

To increase the number of allowed scheduled conference participants that was previously defined in `conference_schedule` (Conference schedule) call flow attribute, `part_count` value for the started conference you should run the following *mf* console command with two required arguments:

```
conf-schedule-incsize <conf_number> <count>
```

where

- `<conf_number>` – The number of the started scheduled conference where you would like to increase the number of the allowed participants.
- `<count>` – The increment in the count of the conference participants that should be added to the number of maximal allowed the scheduled conference participants.

For example if you would like to extend scheduled conference *999020* duration on *600* seconds (i.e. 10 minutes) you should run *mf* console command:

```
conf-schedule-extend 999020 600
```

If you would like to increase the maximal number of allowed scheduled conference *999020* participants on *2* callers you should run *mf* console command:

```
conf-schedule-incsize 999020 2
```

Note that the scheduled conference must be started prior to these commands execution.

If the command was implemented successfully you will receive the message: *“Success”*; if the conference not found or not started you will receive the message: *“Error : Conference not found”*; if the specified conference is not scheduled you will receive the message: *“Schedule doesn't defined for this conference”*; if you are trying to extend the conference duration more than allowed by `conference_schedule_extend` call flow attribute you will receive the message: *“Error : Conference cannot be extended”*.

Conferences and Calls Management using *asterisk* Console

Calls and conferences can be also managed not for the entire bridge but for specific *asterisk* node only, i.e. the specific computer where *Frontend* components were installed. See section Nodes Administration for the information how to install and configure additional nodes.

If you are using the single bridge installation where all the WYDE software components were installed *asterisk* console commands will return the same conferences and calls information that analogous *mf* console commands return.

Using the IVR/*asterisk* Console

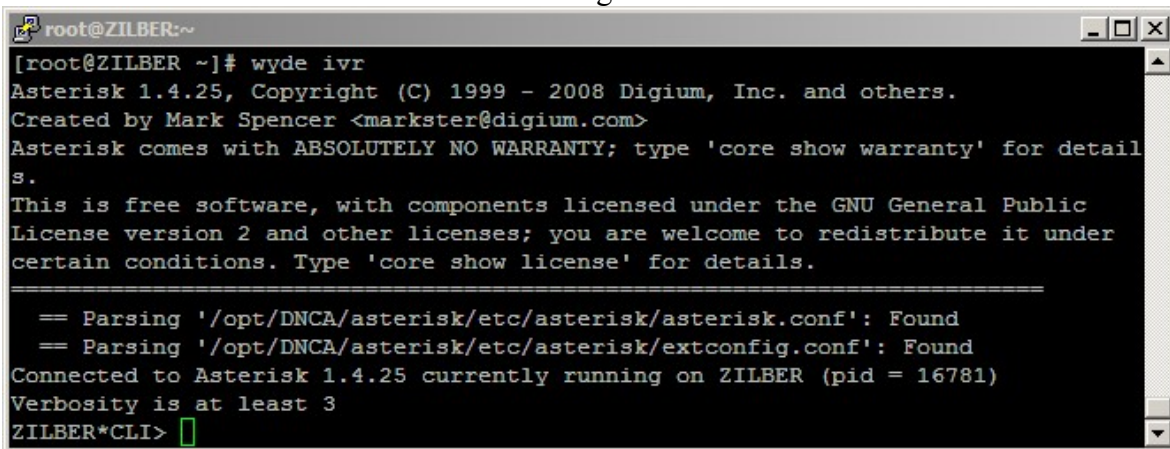
To start IVR, i.e. *asterisk* console you should either use the *wyde* command line utility with the *ivr* option:

```
wyde ivr
```

or you can use the following command to run the *asterisk* console:

```
/opt/DNCA/asterisk/usr/sbin/asterisk -rvvv -C  
/opt/DNCA/asterisk/etc/asterisk/asterisk.conf
```

You will see the screen similar to shown on Figure 32.



```
root@ZILBER:~  
[root@ZILBER ~]# wyde ivr  
Asterisk 1.4.25, Copyright (C) 1999 - 2008 Digium, Inc. and others.  
Created by Mark Spencer <markster@digium.com>  
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.  
This is free software, with components licensed under the GNU General Public  
License version 2 and other licenses; you are welcome to redistribute it under  
certain conditions. Type 'core show license' for details.  
=====
```

```
== Parsing '/opt/DNCA/asterisk/etc/asterisk/asterisk.conf': Found  
== Parsing '/opt/DNCA/asterisk/etc/asterisk/extconfig.conf': Found  
Connected to Asterisk 1.4.25 currently running on ZILBER (pid = 16781)  
Verbosity is at least 3  
ZILBER*CLI> 
```

Figure 32: Starting *asterisk* Console

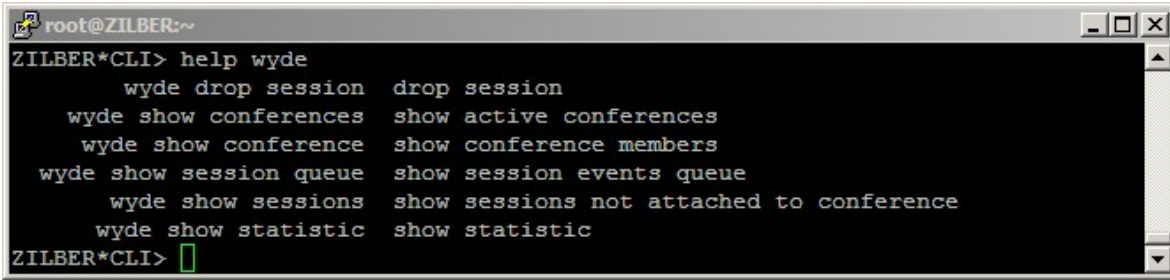
This console is the standard *asterisk* console where you can execute standard *asterisk* commands. But additionally this console has *wyde* commands extension, i.e. additional *wyde* commands can be executed using this console. This section describes *wyde* commands that were added to the *asterisk* console only and does not describe standard *asterisk* commands.

To see the list of all available *asterisk* console *wyde* commands in the console you should type the command:

```
help wyde
```

You will see the screen similar to shown on Figure 33. All *wyde* commands available in *asterisk* console are listed in Table 6, the command reference is given in Chapter 3:

Command Reference, Section: *asterisk* Console Command Reference. These commands will be described in more detail later in this Guide.



```

root@ZILBER:~
ZILBER*CLI> help wyde
    wyde drop session      drop session
    wyde show conferences  show active conferences
    wyde show conference   show conference members
    wyde show session queue show session events queue
    wyde show sessions     show sessions not attached to conference
    wyde show statistic    show statistic
ZILBER*CLI>

```

Figure 33: *asterisk* Console, *help wyde* Command Output

Table 6: *asterisk* Console Utility Available *wyde* Commands

Commands	Description
wyde drop session	drop session
wyde show conferences	show active conferences
wyde show conference	show conference members
wyde show session queue	show session events queue
wyde show sessions	show sessions not attached to conference
wyde show statistic	show statistic

If you need detail help about any of these commands you should use the command:

`help command`

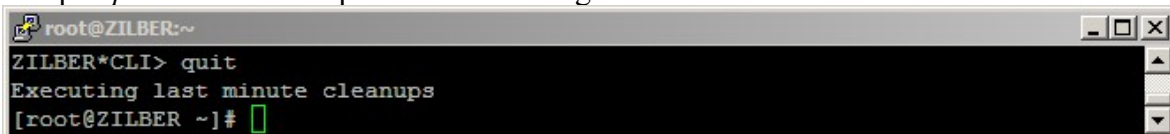
where

- *command* – the specific *asterisk* command on which you would like to get help.

If you would like to exit from *asterisk* console you should use the command:

`quit`

Sample *quit* command output is shown on Figure 34.



```

root@ZILBER:~
ZILBER*CLI> quit
Executing last minute cleanups
[root@ZILBER ~]#

```

Figure 34: *asterisk* Console *quit* Command Output Sample

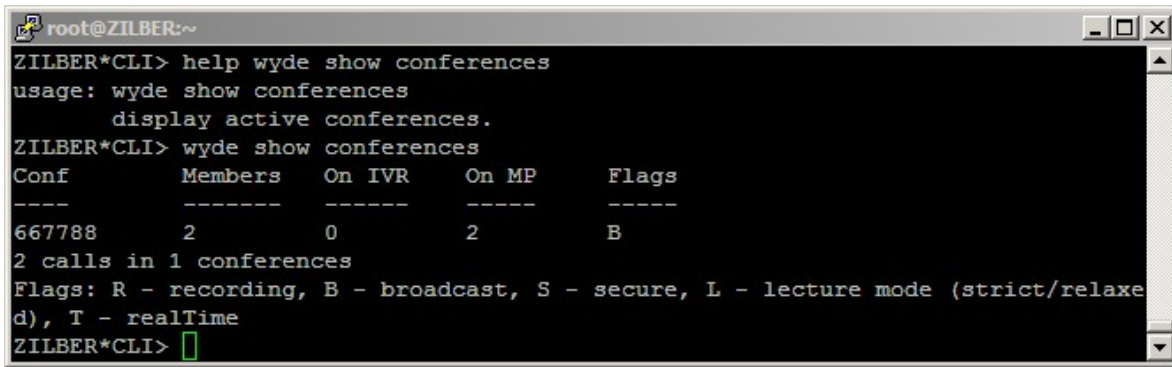
View Conferences in Progress

Any time when there are conferences in progress, you can view them for the specific node using *asterisk* console.

To show a list of all conferences that currently are in process on the bridge node using the command line, you should use *asterisk* console and run the following command from it:

`wyde show conferences`

You will see the screen similar to shown on Figure 35.



```

root@ZILBER:~
ZILBER*CLI> help wyde show conferences
usage: wyde show conferences
        display active conferences.
ZILBER*CLI> wyde show conferences
Conf      Members  On IVR   On MP    Flags
-----
667788     2           0        2        B
2 calls in 1 conferences
Flags: R - recording, B - broadcast, S - secure, L - lecture mode (strict/relaxed), T - realTime
ZILBER*CLI>

```

Figure 35: asterisk Console *wyde show conferences* and its *help* Commands Output Sample

All active conferences are listed sorted ascending by conference number. There are several columns of information about each conference. Table 3 details what each column indicates. See section View Conferences in Progress for *mf* console for additional information.

View Conference Calls in Progress

Any time when there are conference calls in progress, you can view them in the command line interface using *asterisk* console.

To show a list of all calls that are joined to the specific conference using *asterisk* console you can use the following command:

```
wyde show conference <number>
```

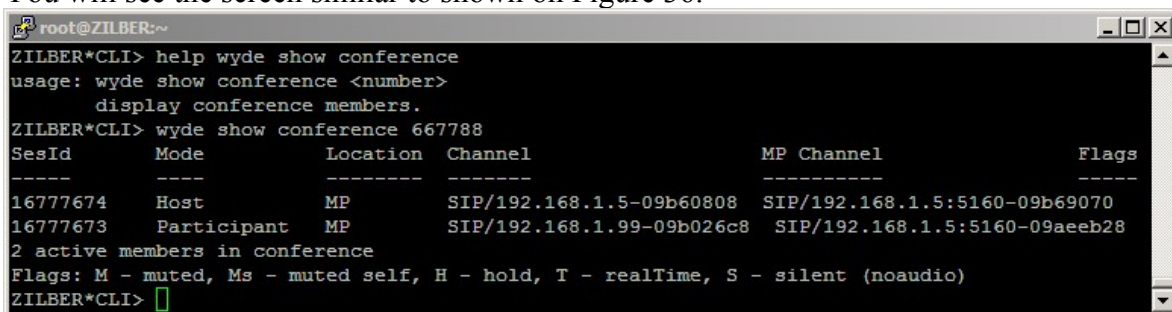
where

- <number> – The number of the conference which calls you would like to view.

For example if you would like to view the calls that were joined to the conference 667788 you should run the command:

```
wyde show conference 667788
```

You will see the screen similar to shown on Figure 36.



```

root@ZILBER:~
ZILBER*CLI> help wyde show conference
usage: wyde show conference <number>
        display conference members.
ZILBER*CLI> wyde show conference 667788
SesId      Mode      Location  Channel      MP Channel      Flags
-----
16777674    Host      MP        SIP/192.168.1.5-09b60808  SIP/192.168.1.5:5160-09b69070
16777673    Participant MP        SIP/192.168.1.99-09b026c8  SIP/192.168.1.5:5160-09aeeb28
2 active members in conference
Flags: M - muted, Ms - muted self, H - hold, T - realTime, S - silent (noaudio)
ZILBER*CLI>

```

Figure 36: asterisk Console *wyde show conference* and its *help* Commands Output Sample

All active conference calls will be shown. There are several columns of information about each conference call. Table 4 details what each column indicates. In addition to *mf* console *show* command output this command returns two extra columns:

- *Channel* – name/address of the channel from asterisk to the user, for instance it could be the IP address where the call came from;
- *MP Channel* – name/address of the channel from asterisk to the *MP*.

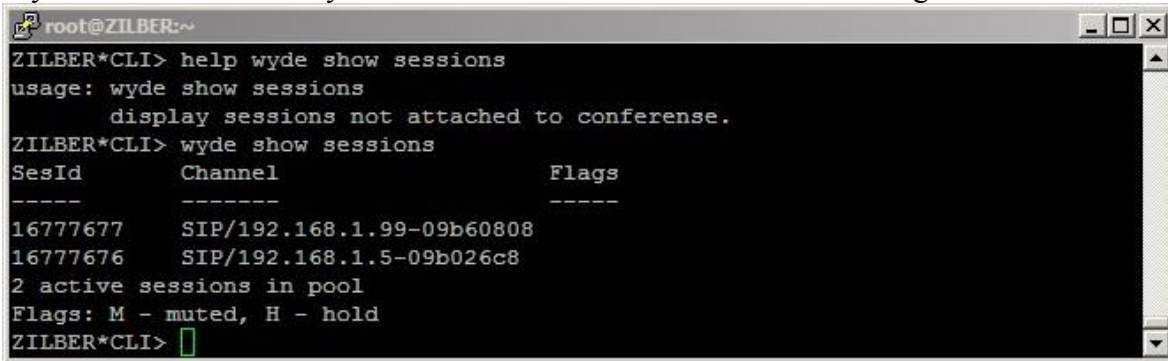
If you try to show conference calls for the conference that does not started or does not exist the error message “*Conference '<number>' not found*” will be returned back to you.

Show Calls that have not placed to Conferences

To show a list of all calls that are not joined to any of the conferences using *asterisk* console you can use the following command:

```
wyde show sessions
```

If you run this command you will see the screen similar to shown on Figure 37.



```

root@ZILBER:~
ZILBER*CLI> help wyde show sessions
usage: wyde show sessions
        display sessions not attached to conference.
ZILBER*CLI> wyde show sessions
SesId      Channel      Flags
-----
16777677   SIP/192.168.1.99-09b60808
16777676   SIP/192.168.1.5-09b026c8
2 active sessions in pool
Flags: M - muted, H - hold
ZILBER*CLI>

```

Figure 37: *asterisk* Console *wyde show sessions* and its *help* Commands Output Sample

Because these calls are not connected to any of the conferences this view contains only session identifier, channel and flags columns and it does not contains other columns that are shown in *wyde show conference* command output.

Dropping Conference Call Participants

If during a call, you wish to cancel a conference call for specific participants, you may use the *wyde drop session* command of *asterisk* console to kick someone off the call. To drop someone from a conference call you should run the following *asterisk* console command:

```
wyde drop session <conf_number> <session_id>
```

where

- *<conf_number>* – The number of the conference which call you wish to drop (disconnect).
- *<session_id>* – The call session identifier you wish to drop from the conference.

Both arguments are required.

For example if you would like to drop the session *16777673* from the conference *667788* you should run *asterisk* console command:

```
wyde drop session 667788 16777673
```

If the conference not found or not started you will receive the message “*Conference '<conf_number>' not found*”; if the session not found you will not receive any error message, but the call will not be dropped; otherwise if all parameters are correct the conference call will be dropped and you may receive *asterisk* messages in the console about the call hang-up processing.

Show Session Events Queue

Each call has its own events (commands) queue to *Frontend*. For example when the caller tries to execute any DTMF command this command is being placed into command queue. For debugging purposes you may wish to show the session events queue. This can be made using the following *asterisk* console command:

```
wyde show session queue <conf_number> <session_id>
```

where

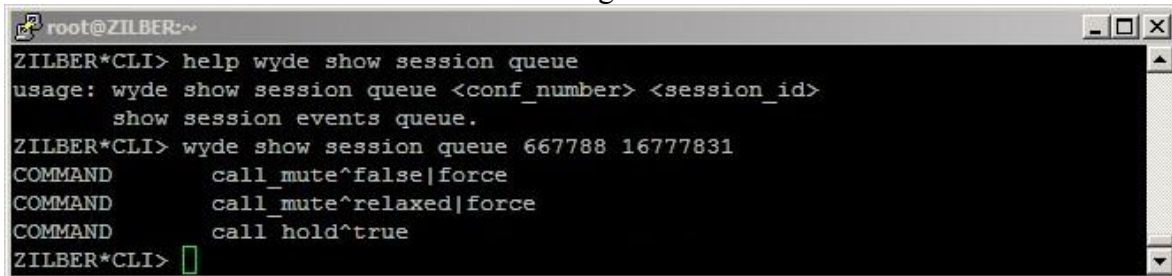
- <conf_number> – the call conference number which events queue you would like to show.
- <session_id> – the call session identifier which events queue you would like to show.

Both arguments are required.

For example if you would like to show the events queue for the session *16777831* of the conference *667788*, you should run *asterisk* console command:

```
wyde show session queue 667788 16777831
```

You will see the screen similar to shown on Figure 38.



```
root@ZILBER:~  
ZILBER*CLI> help wyde show session queue  
usage: wyde show session queue <conf_number> <session_id>  
       show session events queue.  
ZILBER*CLI> wyde show session queue 667788 16777831  
COMMAND      call_mute^false|force  
COMMAND      call_mute^relaxed|force  
COMMAND      call_hold^true  
ZILBER*CLI> 
```

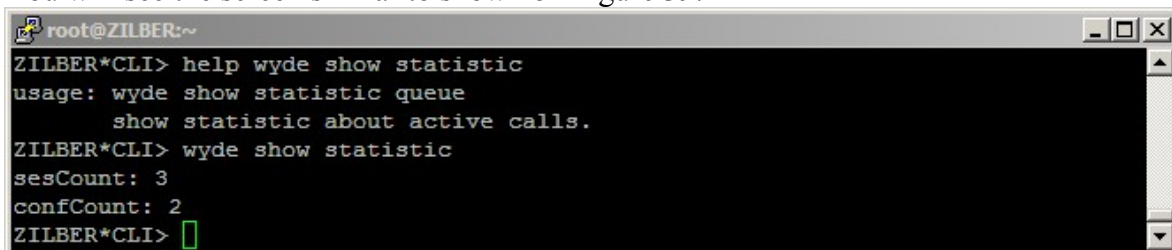
Figure 38: *asterisk* Console *wyde show session queue* and its *help* Commands Output Sample

Show Node Statistics

If you would like to see statistics, i.e. number of conferences and number of calls, for specific *asterisk* node it can be made using the following *asterisk* console command:

```
wyde show statistic
```

You will see the screen similar to shown on Figure 39.



```
root@ZILBER:~  
ZILBER*CLI> help wyde show statistic  
usage: wyde show statistic queue  
       show statistic about active calls.  
ZILBER*CLI> wyde show statistic  
sesCount: 3  
confCount: 2  
ZILBER*CLI> 
```

Figure 39: *asterisk* Console *wyde show statistic* and its *help* Commands Output Sample

Note that information returned by this command displays statistics not for entire bridge, but for selected *asterisk* node only.

Conferences and Calls Management using *mp* Console

The *mp* console is the tool that could be used to manage *Backend* components (media processor, *mpw* service) of the WYDE bridge. Calls and conferences that are being managed by this console are being controlled for specific *Backend* only.

Using the *mp* Console

The *mp* console is being installed in *Backend* installation of the WYDE bridge software.

If you are using the single bridge installation where all the WYDE software components were installed to start *mp* console you should run the command:

```
telnet <Your Bridge IP> 4545
```

or, for example if you are running this command from your bridge computer:

```
telnet localhost 4545
```

If you use distributed installation to start *mp* console you should point to the computer where *Backend* components were installed:

```
telnet <Your Backend Computer IP> 4545
```

If you enter any unknown command the *mp* console returns you the list of all available commands, see Figure 40 for details. Additionally if you try using any *mp* command incorrectly, the console returns the format of the command you are trying to use. All *mp* console commands are listed in Table 7. These commands will be described in more detail later in this Guide.

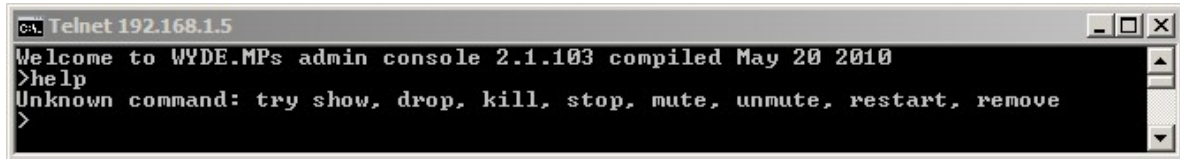


Figure 40: Starting *mp* console, Available Commands

Table 7: *mp* Console Utility Available Commands

Commands	Description
show	Show different <i>mp</i> statistics
timer	Start timer running <i>show</i> command
kill	Stop specific timer or all timers
drop	Drop boards and calls
stop	Stop <i>mp</i> components
restart	Restart <i>mp</i> boards and logs

Show Different *mp* Information

The *mp* console *show* command displays information about conferences, calls, boards, and other statistics. The syntax is as follows:

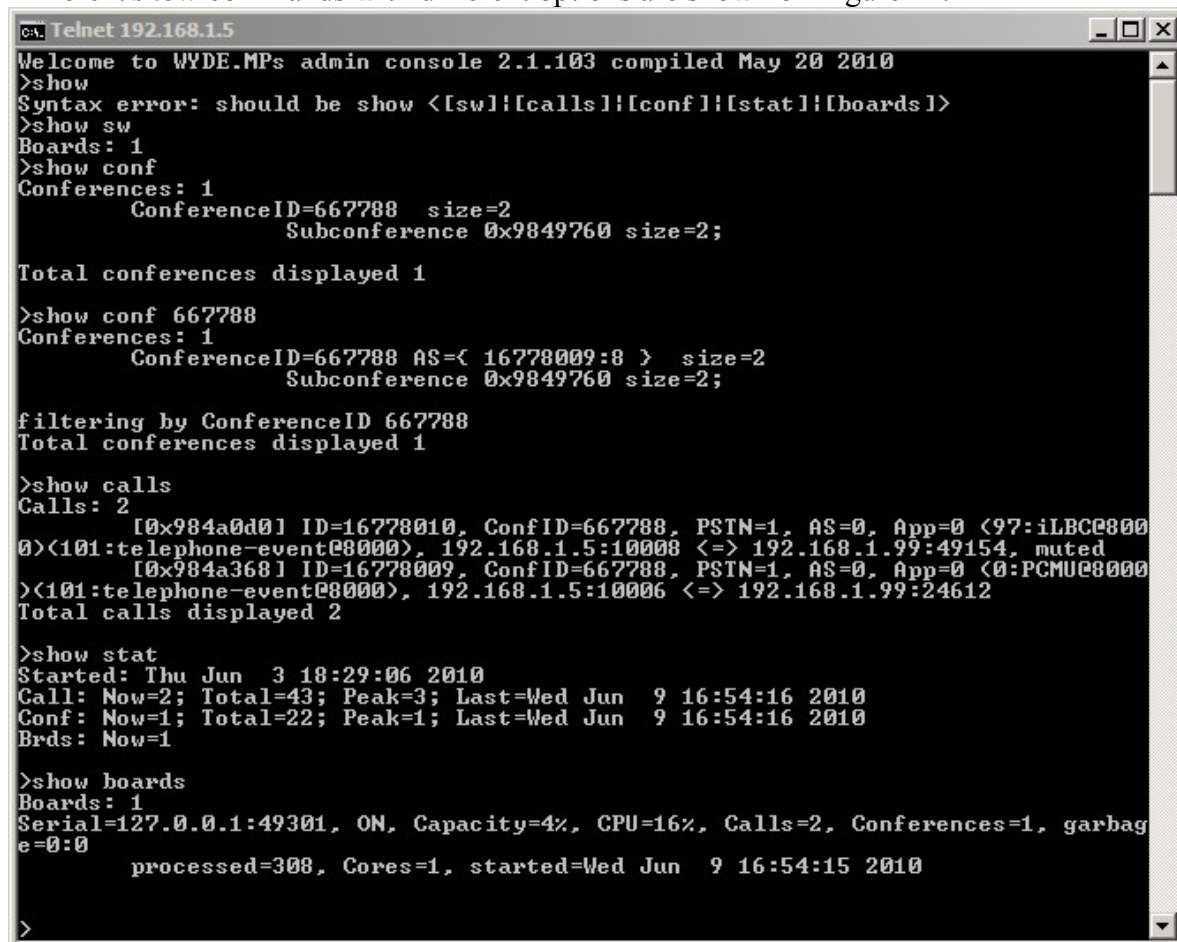
```
show {sw|conf|conf <conf_number>|calls|stat|boards}
```

where one of the following arguments should be specified:

- *sw* – show the number of available *MPw* boards, i.e. backend components count;

- `conf` – show all conferences currently running on the *mp*;
- `conf <conf_number>` – show statistics for the specific conference only, including active speaker notifications;
- `calls [<id1>-<id2>]` – show all calls currently running on the *mp* (the command returns calls identifiers, conference numbers, IP addresses, ports, codecs, status, etc.); if you specify the calls sessions identifiers range, only the calls within this range will be returned by the command;
- `stat` – show overall *mp* statistics about calls, conferences and *MPs* boards (*now* column shows current data, *total* column shows data from *mp* start, *peak* column shows acme/height value);
- `boards` – show detail *MPs* boards statistics, including the board current state, capacity, CPU load, number of conferences and calls, etc.

Different *show* commands with different options are shown on Figure 41.



```

CA: Telnet 192.168.1.5
Welcome to WYDE.MPs admin console 2.1.103 compiled May 20 2010
>show
Syntax error: should be show [<sw>|<calls>|<conf>|<stat>|<boards>]
>show sw
Boards: 1
>show conf
Conferences: 1
      ConferenceID=667788 size=2
      Subconference 0x9849760 size=2;

Total conferences displayed 1
>show conf 667788
Conferences: 1
      ConferenceID=667788 AS=< 16778009:8 > size=2
      Subconference 0x9849760 size=2;

filtering by ConferenceID 667788
Total conferences displayed 1
>show calls
Calls: 2
      [0x984a0d0] ID=16778010, ConfID=667788, PSTN=1, AS=0, App=0 <97:iLBC08000>
      <101:telephone-event08000>, 192.168.1.5:10008 <=> 192.168.1.99:49154, muted
      [0x984a368] ID=16778009, ConfID=667788, PSTN=1, AS=0, App=0 <0:PCMU08000>
      <101:telephone-event08000>, 192.168.1.5:10006 <=> 192.168.1.99:24612
Total calls displayed 2
>show stat
Started: Thu Jun 3 18:29:06 2010
Call: Now=2; Total=43; Peak=3; Last=Wed Jun 9 16:54:16 2010
Conf: Now=1; Total=22; Peak=1; Last=Wed Jun 9 16:54:16 2010
Brds: Now=1
>show boards
Boards: 1
Serial=127.0.0.1:49301, ON, Capacity=4%, CPU=16%, Calls=2, Conferences=1, garbage=0:0
      processed=308, Cores=1, started=Wed Jun 9 16:54:15 2010
>

```

Figure 41: *mp* Console *show* Commands Output Sample

Start and Stop *mp* Console Timers

If it is necessary you can run any *mp* console *show* command automatically in a given interval using *timer* command. The syntax is as follows:

```
timer <interval> show <options>
```

where

- <interval> – denotes interval in seconds in which the *show* command should be re-run;
- show <options> – denotes specific *show* command with its options that should be repeated in the given *interval*.

These commands will be repeated until the timer will be stopped. The *mp* console *kill* command should be used to stop the specific timer of all started timers. The syntax is as follows:

```
kill {<id> [<id>] ...|all}
```

where

- <id> – the specific timer identifier that should be stopped;
- all – denotes that all started timers should be stopped.

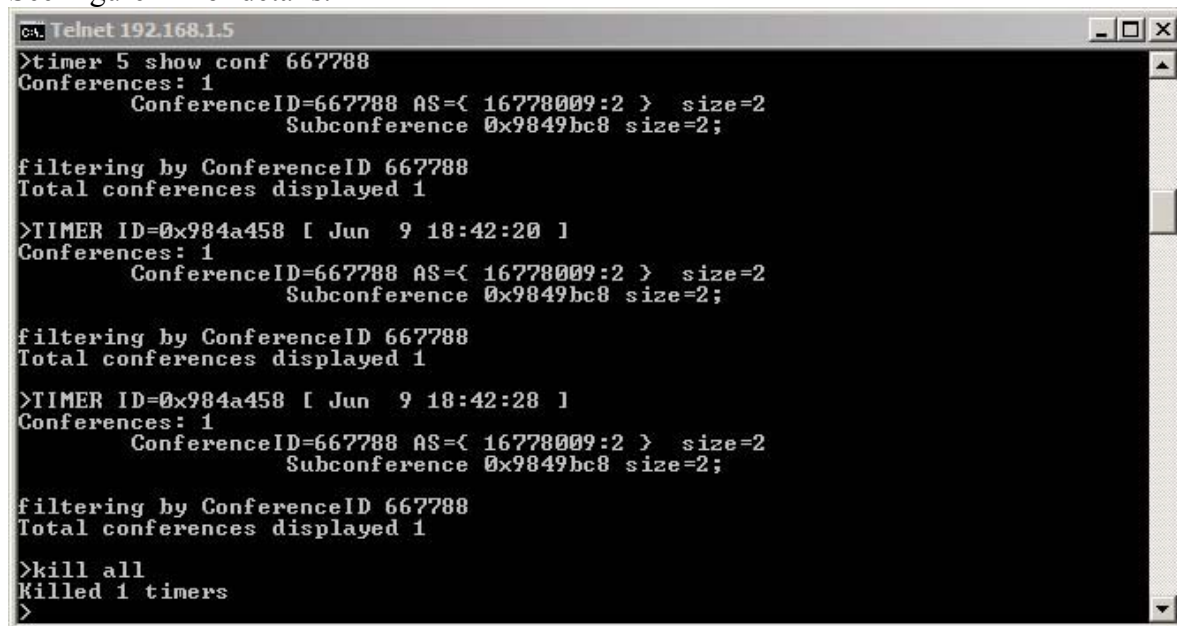
For example if you would like to show conference 667788 status each 5 seconds you should run the command:

```
timer 5 show conf 667788
```

To stop this timer you could use the command:

```
kill all
```

See Figure 42 for details.



```

C:\>Telnet 192.168.1.5
>timer 5 show conf 667788
Conferences: 1
      ConferenceID=667788 AS={ 16778009:2 } size=2
      Subconference 0x9849bc8 size=2;

filtering by ConferenceID 667788
Total conferences displayed 1

>TIMER ID=0x984a458 [ Jun  9 18:42:20 ]
Conferences: 1
      ConferenceID=667788 AS={ 16778009:2 } size=2
      Subconference 0x9849bc8 size=2;

filtering by ConferenceID 667788
Total conferences displayed 1

>TIMER ID=0x984a458 [ Jun  9 18:42:28 ]
Conferences: 1
      ConferenceID=667788 AS={ 16778009:2 } size=2
      Subconference 0x9849bc8 size=2;

filtering by ConferenceID 667788
Total conferences displayed 1

>kill all
Killed 1 timers
>

```

Figure 42: *mp* Console *timer* and *kill* Commands Output Sample

Dropping Boards and Calls

To drop *MPw* boards and calls from *mp* you can use *mp* console *drop* command. The syntax is as follows:

```
drop {<boards <serial> [<serial>] ...>|<calls <<id> [<id>] ...>|all>}
```

where one of the following arguments should be specified:

- `<boards <serial> [<serial>] ...>` – denotes *MPw* boards that should be dropped (placed on hold); use `<serial>` to specify the boards that should be dropped;
 - ✓ this option works as toggle switch – if you repeat this command the board will be switched on;
- `<calls <<id> [<id>] ...>|all>` – denotes that the calls should be dropped from *mp*; use `<id>` to specify the call sessions identifiers that should be dropped or use `all` keyword to drop all calls.

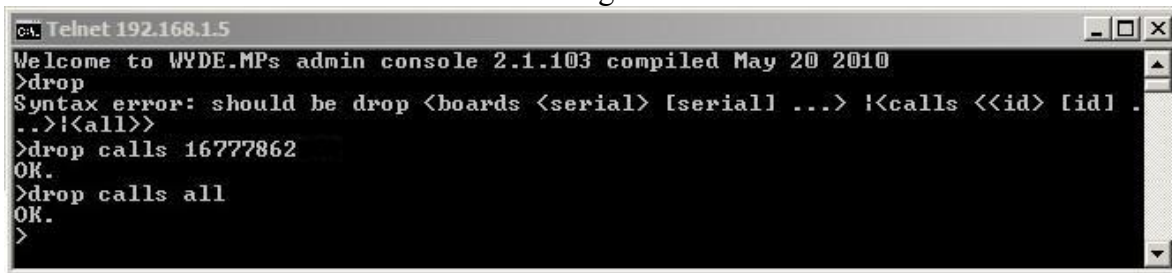
For example if you would like to drop the call *16777862* from *mp* you show run the command:

```
drop calls 16777862
```

To drop all calls from *mp* use the command:

```
drop calls all
```

You will see the screen similar to shown on Figure 43.



```

C:\Telnet 192.168.1.5
Welcome to WYDE.MPs admin console 2.1.103 compiled May 20 2010
>drop
Syntax error: should be drop <boards <serial> [serial] ...> !<calls <<id> [id] .
..>|all>>
>drop calls 16777862
OK.
>drop calls all
OK.
>
```

Figure 43: *mp* Console *drop* Commands Output Sample

Restarting and Stopping *mp* Boards and Logs

If it is necessary to restart *MPw* boards or *mp* logs you can use *mp* console *restart* command. The syntax is as follows:

```
restart {<log [<level>]>|<board <serial>|all>}
```

where

- `<log [<level>]>` – denotes that the log should be restarted; `<level>` – specifies what messages should be stored in the log: `3` – denotes that only information and error messages should be stored in the log file (default); `9` – denotes that debugging mode is switched on and all possible log messages should be added to the log file.
- `<board <serial>|all>` – denotes the *MPw* boards that should be restarted; use `all` keyword to restart all *MPw* boards.

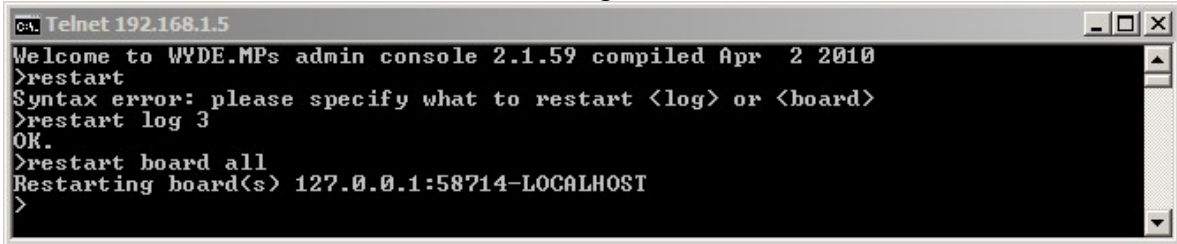
For example if you would like to restart the log and include into the log file only information and error messages you show run the following command from *mp* console:

```
restart log 3
```

For example if you would like to restart all *MPw* boards you should run *mp* console command:

```
restart board all
```

You will see the screen similar to shown on Figure 44.

A screenshot of a Telnet window titled 'C:\ Telnet 192.168.1.5'. The window shows a command-line interface for 'WYDE.MPs admin console 2.1.59 compiled Apr 2 2010'. The user enters '>restart', which results in a 'Syntax error: please specify what to restart <log> or <board>'. The user then enters '>restart log 3', which results in 'OK.'. Finally, the user enters '>restart board all', which results in 'Restarting board(s) 127.0.0.1:58714-LOCALHOST' followed by a prompt '>'.

```
C:\ Telnet 192.168.1.5
Welcome to WYDE.MPs admin console 2.1.59 compiled Apr 2 2010
>restart
Syntax error: please specify what to restart <log> or <board>
>restart log 3
OK.
>restart board all
Restarting board(s) 127.0.0.1:58714-LOCALHOST
>
```

Figure 44: *mp* Console *restart* Commands Output Sample

To stop *mp* processes (*mpw* and *mps* services) you can use *mp* console *stop* command, but this service is being restarted automatically because it is need for the WYDE bridge activity. The syntax is as follows:

```
stop
```

Operator Conferences

Operator conferences and call flows are designed to administrate the conferences and assist the users in case of any problems. In other words operator conferences can be used as quick technical support for users. The operator can monitor the conferences, connect to the different conferences, receive requests from the users, attach the users to the different conferences, dialout, etc. When operator connects to his OPERATOR conference he is being notified when new requests are assigned to his queue or he can initiate the conference surveillance and management.

During the conference any call participants can request a call to an operator. To do that they should press *0 (default, can be overridden); after that the users will be asked to press:

- 1 – to speak to an operator
- 2 – to pause the request and return to the conference

If the user presses 1 button he will be prompted: *“Please wait and you will connect to an operator”*. After that the user will be placed to the operator queue.

Any time while the user is waiting the operator response he can press *0 again and he will be asked to press:

- 1 – to cancel the request
- 2 – to return to the conference

In addition some call flows can provide that if the user is entering access code three times incorrectly (default, can be overridden) this user is being placed to an operator queue automatically. The operator can assist to such user connect to the requested conference.

Operator conference management can be implemented using *mf* console. In addition operator conferences can be managed using DTMF keypad on your phone and the Web Administration Interface as it is described in “Web Administration Interface – User Guide”.

Operator Conferences Management using *mf* Console

To show started operator conferences, i.e. the operator conferences that are in progress, you should run the following *mf* console command:

```
op-show [<operator_number>]
```

where

- `<operator_number>` – The operator conference number that you would like to show.
 - If this argument is omitted the brief information about all started operator conferences will be displayed; this information includes operator conference number, used DNIS (DID) number and current operator status (Free, Talk, Listen).
 - If this argument is specified the detail information about specific operator conference will be shown; this information includes data about this operator conference, such as current operator status (the same as above), flags, and information about users who are talking to the operator: their source conference number, DNIS, session identifier. Operator conference flag is two-digit number

MN, where M and N is either 0 or 1; M=1 means that the operator is currently connected to another conference, N=1 means that the operator is currently scanning and listening other conferences. For example status=Talk, flags=01 means that the operator is currently talking to a user and he has activated conference surveillance, when he finishes the conversation the conference monitoring will be automatically resumed.

To show the operator queue you should run the following *mf* console command:

```
op-queue
```

This command displays all users who are currently waiting the operator assistance in operator queue. This information includes source conference number (if applicable), DNIS and session identifier; if there are no any users in the queue the message “*there are 0 calls in the queue*” will be returned.

For example to show all started operator conferences you should run *mf* console command:

```
op-show
```

To show detail information about operator conference *390008* you should run *mf* console command:

```
op-show 390008
```

To show the operator queue you should run *mf* console command:

```
op-queue
```

Figure 45 displays both these commands samples when the operator queue is empty, non-empty, and when the operator is talking to the user from another conference.


```

root@ZILBER:~
WYDE.MF>op-show
Number      DIDMask      Status
-----
390008      *            Free
There are 1 operators.
WYDE.MF>op-queue
DIDNumber    ConfNumber    SesId
-----
There are 0 calls in the queue.
WYDE.MF>op-queue
DIDNumber    ConfNumber    SesId
-----
12           667788       16777230
There are 1 calls in the queue.
WYDE.MF>op-show
Number      DIDMask      Status
-----
390008      *            Free
There are 1 operators.
WYDE.MF>op-show 390008
Operator: number=390008, status=Free, flags=00
WYDE.MF>op-talk 390008 start
Success
WYDE.MF>op-queue
DIDNumber    ConfNumber    SesId
-----
There are 0 calls in the queue.
WYDE.MF>op-show
Number      DIDMask      Status
-----
390008      *            Talk
There are 1 operators.
WYDE.MF>op-show 390008
Operator: number=390008, status=Talk, flags=00
Talking with: DIDNumber=12, ConfNumber=667788, SesId=16777230
WYDE.MF>

```

show started operator conference

operator queue is empty

operator queue has one call

the call from the queue is not processing yet

show operator conference brief and detail information

start talking to the user from the queue

operator queue is empty

operator is talking to the user from the queue

show operator conference brief and detail information

Figure 45: *mf* Console *op-show* and *op-queue* Commands Output Sample

To start or stop conference monitoring (scan), i.e. surveillance call, you should run the following *mf* console command:

```
op-scan <operator_number> {start|stop}
```

where

- *<operator_number>* – The operator conference number where you would like to start or stop conference monitoring (surveillance call).
- *{start|stop}* – *start* denotes that the monitoring should be started; *stop* denotes that the monitoring should be stopped.

All arguments are required. When monitoring is started the operator is being switched between all started conferences each 30 seconds until he stops the monitoring or connects to another specific conference or talking to the user from his queue.

To connect and talk to the next user, i.e. receive the call from the user from the operator queue, you should run the following *mf* console command:

```
op-talk <operator_number> {start|stop}
```

where

- *<operator_number>* – The operator conference number where you would like to start or stop talking to a user.
- *{start|stop}* – *start* denotes that the talking should be started; *stop* denotes that the talking should be stopped.

All arguments are required. When the talking is stopped the user who was talking to the operator is being returned to his conference or ivr.

To connect and listen another conference you should run the following *mf* console command:

```
op-listen <operator_number> {start <conf_number>
    [{directlink|shunt} [mute]]|stop}
```

where

- *<operator_number>* – the operator conference number that would like to listen other users conferences. This argument is required.
- *{start|stop}* – *start* denotes that the listening should be started; *stop* denotes that the listening should be stopped. This argument is required. If *start* is specified the following arguments could be specified here:
 - *<conf_number>* – the number of the conference you wish to start listening, must be indicated if *start* option is specified;
 - *{directlink|shunt}* – *directlink* 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), if this argument is omitted the *directlink* mode is being used by default;
 - *mute* – when specified denotes that the operator is muted, so he can only hear the specified conference and he is unable to talk.

Note that *op-show* command display information about connected conference, i.e. the connection mode (*directlink* or *shunt*) and the number of the conference you are listening.

To attach the user who currently is talking to the operator to a different conference, i.e. move user to a conference, you should run the following *mf* console command:

```
op-call-move <operator_number> <new_did_number>
    <new_accesscode> [<new_role>]
```

where

- *<operator_number>* – The operator conference number whose current call (i.e. the user that currently talking to the operator) you would like to move to another conference.
- *<new_did_number>* – New (i.e. target) conference DNIS (DID) number where the call should be moved.

- `<new_accesscode>` – The access code that should be used to join to new (i.e. target) conference.
- `<new_role>` – 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 argument `<new_role>` is optional and should be used for call flows without authorization only. All other arguments are required.

To initiate dial-out from the operator conference you should run the following *mf* console command:

```
op-dialout <operator_number> <peer_number>
```

where

- `<operator_number>` – The operator conference number where you are initiating the dialout.
- `<peer_number>` – Denotes the phone number you wish to dial.

All arguments are required. When dial-out is performed the user that was dialed is being connected to the operator conference. After that the operator is able to talk to the connected user, attach him to another conference, etc.

Let's assume that the operator conference *390008* is stated; another conference that we use to connect to is *667788*, its DNIS is *8665080012*, its participant access code is *6602*; phone number to dial-out *6046880331*. Table 8 shows samples what commands could be implemented to assist users via this operator conference using *mf* console commands.

Table 8: Operator Conference Management Samples Using *mf* Console Commands

Description	<i>mf</i> Console Command	DTMF
Start conference monitor (surveillance call)	<code>op-scan 390008 start</code>	*1
Stop conference monitor (surveillance call)	<code>op-scan 390008 stop</code>	*1 2
To connect and talk to the next user from the operator queue	<code>op-talk 390008 start</code>	*2
To stop talking to the current user and return him to his conference or ivr	<code>op-talk 390008 stop</code>	*3
To connect and listen another conference without current user (default <i>directlink</i> mode used)	<code>op-listen 390008 start 667788</code>	*4 2
To connect and listen another conference without current user and muting himself	<code>op-listen 390008 start 667788 directlink mute</code>	*4 2
To connect and listen another conference with current user (<i>shunt</i> mode used)	<code>op-listen 390008 start 667788 shunt</code>	*4 1
To stop listening another conference	<code>op-listen 390008 stop</code>	*3
Attach current user to a different conference	<code>op-call-move 390008 8665080012 6602</code>	*5
Dialing out to another user	<code>op-dialout 390008 6046880331</code>	*7

Samples and Use-Cases of Operator Conference Procedures

This section of the document will describe few samples about how the operator can assist users in their conference calls.

First of all the operator can monitor (hear) the conference to examine if there are any problems where he can assist.

Also for some call flows the operator will receive automatic requests from users who entered incorrect access codes three times (default, can be overridden). In this case the operator asks the user information about who he is and about the conference he wish to connect, connects to the requested conference, asks if this conference expecting this user, and if so, returns and attaches the user to this conference.

Moreover the users can request operator assistance in case of any problems, for instance bad audio quality or noises. In this case operator can connect to the user conference and use Web Administration Interface – Active Speaker notification mode (see “Web Administration Interface – User Guide”, section: Active Speaker Notification (ASN) mode) to examine where is the problem and after that inform the user about the reason of the problems.

In addition the operator can dialout to another user and attach the user to the conference. If call flows do not allow making dialout within the conference the operator assistance is needed.

WYDE Bridge Administration

In addition to the daily conferences and calls administration, subscriber user management, call flow and DNIS management, you will have to administer the WYDE bridge system itself. These system management tasks could be the following:

- bridge monitoring;
- bridge settings management, including their save and restore;
- nodes administration;
- distributed conferencing administration;
- peers management;
- calls transferring management;
- audio prompts management;
- other WYDE bridge administration tasks.



Note that you should be very careful with your bridge administration. It is possible to render your system inoperable if you are not familiar with the WYDE bridge administration. Changing any of the bridge preferences, unless instructed to do so by WYDE technical support, also can render your system inoperable.

Monitoring

The WYDE bridge exposes metrics to allow the monitoring system to measure health and activity on it. Some of these metrics are available via *mf* console commands; some of them are available via web.

To show the number of commands executed on bridge from *mf* starts till now you should run the following *mf* console command:

```
cmdcount-show
```

The counters that are being returned by this command are described in `cmdcount-show` (Display Values of *Command* Counters). You can also use URL:

<http://<Wyde bridge domain>/status/cmdcount> to expose these counters via web.

To show the break-up of conferences according to the participant size from *mf* starts till now you should run the following *mf* console command:

```
confcount-show
```

The counters that are being returned by this command are described in `confcount-show` (Display Values of *confcount* Counters). You can also use URL:

<http://<Wyde bridge domain>/status/confcount> to expose these counters via web.

To show the counters related to the participants from *mf* starts till now you should run the following *mf* console command:

```
partcount-show
```

The counters that are being returned by this command are described in `partcount-show` (Display Values of *partcount* Counters). You can also use URL:

<http://<Wyde bridge domain>/status/partcount> to expose these counters via web.

To show the number of terminated/incomplete calls from *mf* starts till now you should run the following *mf* console command:

```
errcount-show
```

The counters that are being returned by this command are described in `errcount-show` (Display Values of *Error* Counters). You can also use URL:

<http://<Wyde bridge domain>/status/errors> to expose these counters via web.

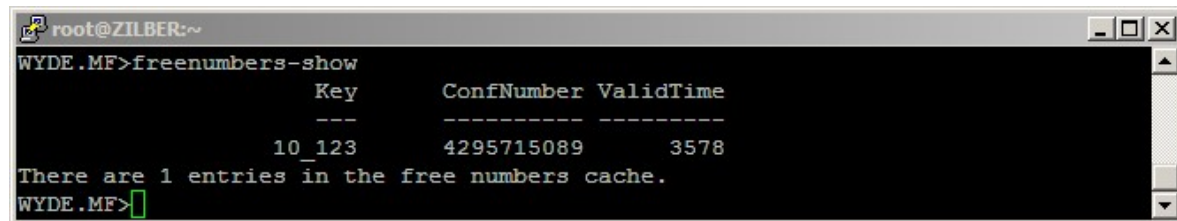
In addition you can use web to show the duration of the conferences that have ended in the last 2 minutes (<http://<Wyde bridge domain>/status/confduration>) and to show the duration of the participants calls that have ended in the last 2 minutes

(<http://<Wyde bridge domain>/status/partduration>).

Conference numbers for call flows without authorization (like CONF) are being issued automatically, once the conference begins. To show these conference numbers together with phone number called, access code used, and valid time, you should run the following *mf* console command:

```
freenumbers-show
```

You will see the screen similar to shown on Figure 46. *Key* column contains *<DNIS number>_<access code>*; *ValidTime* column contains the conference number expiration time in seconds till the end of conference number validity – the calls with the same DNIS number/access code pairs till end of the conference number valid time will get the same conference number.



```

root@ZILBER:~
WYDE.MF>freenumbers-show
      Key      ConfNumber ValidTime
      ---      -
      10_123    4295715089    3578
There are 1 entries in the free numbers cache.
WYDE.MF>

```

Figure 46: *mf* Console `freenumbers-show` Command Output Sample

WYDE bridge logs *mf* activity into `/usr/local/DNCA/log/mf.log` file. To set the logger level you should run the following *mf* console command:

```
set-log-level {debug|event|info}
```

where

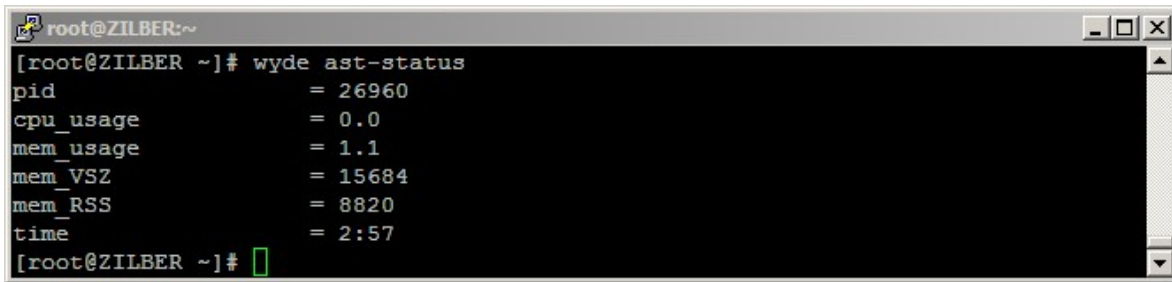
- `info` – denotes that only information and error messages should be stored in the log file (default); `event` – denotes that additionally the log should contain all events logging; `debug` – denotes that debugging mode is switched on, so additionally debug messages should be added to the log. This argument is required.

You can use the information from this *mf.log* file in case of any problems or if you need to monitor WYDE bridge activity.

To show the *asterisk* process status using the command line interface you should use the *wyde* command line utility with the *ast-status* option. The syntax is as follows:

```
wyde ast-status
```

You will see the screen similar to shown on Figure 47.



```

root@ZILBER:~
[ root@ZILBER ~ ]# wyde ast-status
pid                = 26960
cpu_usage          = 0.0
mem_usage          = 1.1
mem_VSZ            = 15684
mem_RSS            = 8820
time               = 2:57
[ root@ZILBER ~ ]#

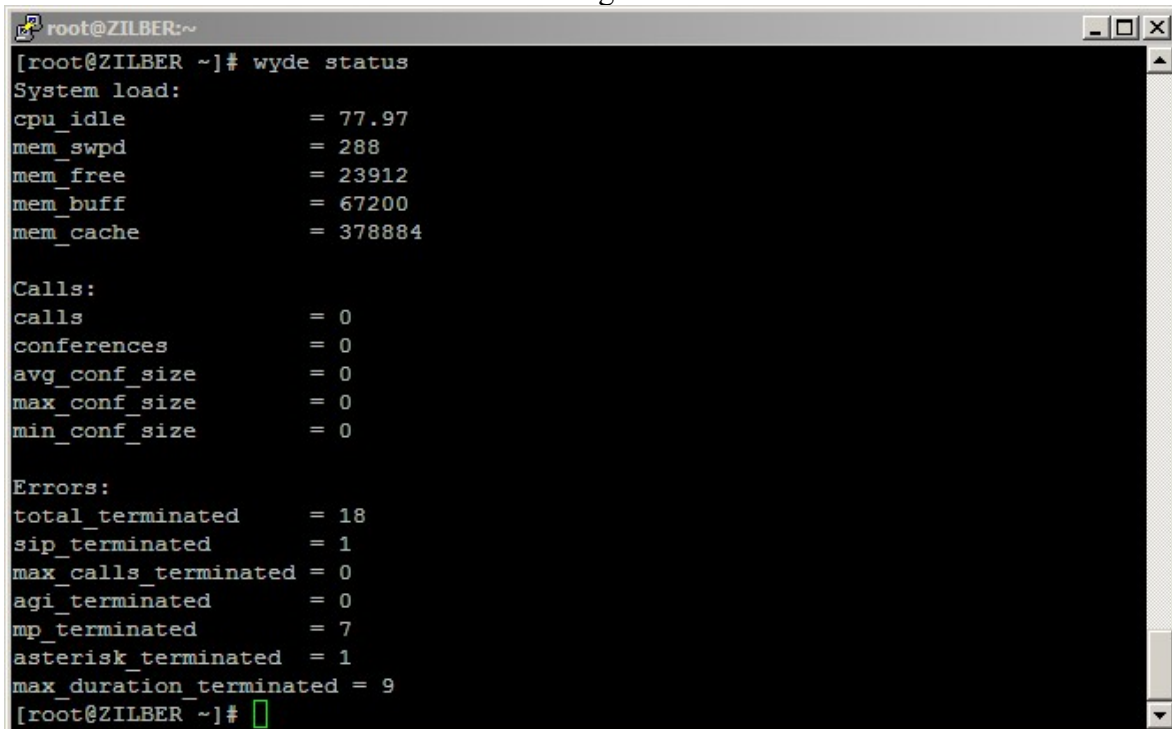
```

Figure 47: *wyde ast-status* Command Output Sample

To show the WYDE bridge status using the command line interface you should use the *wyde* command line utility with the *status* option. The syntax is as follows:

```
wyde status
```

You will see the screen similar to shown on Figure 48.



```

root@ZILBER:~
[ root@ZILBER ~ ]# wyde status
System load:
cpu_idle           = 77.97
mem_swpd           = 288
mem_free           = 23912
mem_buff           = 67200
mem_cache          = 378884

Calls:
calls              = 0
conferences         = 0
avg_conf_size      = 0
max_conf_size      = 0
min_conf_size      = 0

Errors:
total_terminated   = 18
sip_terminated     = 1
max_calls_terminated = 0
agi_terminated     = 0
mp_terminated      = 7
asterisk_terminated = 1
max_duration_terminated = 9
[ root@ZILBER ~ ]#

```

Figure 48: *wyde status* Command Output Sample

Additionally you can watch the same status permanently and refresh it automatically with the given interval. To do so you should use the *wyde* command line utility with the *watch* option. The syntax is as follows:

```
wyde watch [interval <interval>]
```

- *<interval>* – The interval in seconds that denotes the period of how often the bridge status should be refreshed. This argument is optional; if it is omitted the 5 seconds interval is used by default.

This command output is similar to *wyde status* command output (see Figure 48); the only difference is that this screen is being refreshed automatically with the given period. To

interrupt and exit from this command you should use CTRL+C keystroke. Note that each command execution takes 1-2 seconds, so the real interval is greater than you define.

WYDE Bridge Settings Management

If it is necessary you can change your WYDE bridge settings you can use the command line interface. To set WYDE bridge parameter value using the command line interface you should use the *wyde* command line utility with the *settings-edit* option. The syntax is as follows:

```
wyde settings-edit <arguments>
```

Each of the arguments is followed by a space and a value. In *settings-edit* you can specify the following arguments:

- `bridge <value>` – The name of the bridge which settings you would like to edit.
- `name <value>` – The parameter name that should be edited.
- `value <value>` – New parameter value that should be set.

The argument `name` is required, all other arguments are optional. If argument `bridge` is omitted the settings will be edited for the current bridge; the default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file:

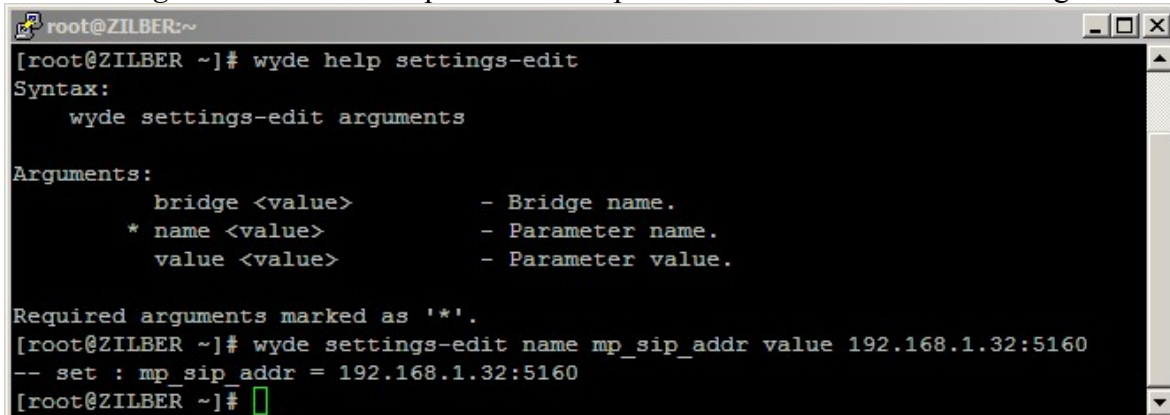
```
[general]
bridge_id = 1
```

If the argument `value` is omitted the empty value will be set to the parameter. The arguments can be transferred to this command in any order.

For example if you would like to change *mp_sip_addr* parameter value to *192.168.1.32:5160* (set new IP address) for the current bridge, you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde settings-edit name mp_sip_addr value 192.168.1.32:5160
```

If the command is successful, the system will return updated parameter name and new value that was set; after that it will return you back to the command prompt (#). The sample of the *settings-edit* command output and the help on this command is shown on Figure 49.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help settings-edit
Syntax:
    wyde settings-edit arguments

Arguments:
    bridge <value>          - Bridge name.
    * name <value>          - Parameter name.
    value <value>           - Parameter value.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde settings-edit name mp_sip_addr value 192.168.1.32:5160
-- set : mp_sip_addr = 192.168.1.32:5160
[root@ZILBER ~]#

```

Figure 49: *wyde help settings-edit* and *wyde settings-edit* Commands Output Sample

If any of WYDE bridge parameters has been changed you should reload them from the database, from *bridge_settings* table. To send the signal on the WYDE bridge *MF* engine to reload all system settings the following *mf* console command should be executed: `settings-reload`

If the command is successful, the system will not return any errors or messages.

If new system parameter was added it should be added into *bridge_settings* table and populated with the default values. Usually you do not need doing it, because it is being run in the WYDE software upgrade installation automatically. However you can update *bridge_settings* table in the database using the *wyde* command line utility with the *settings-update* option. The syntax is as follows:

```
wyde settings-update <arguments>
```

Each of the arguments is followed by a space and a value. In *settings-update* you can specify the following argument only:

- `bridge <value>` – The name of the bridge which settings should be updated. This argument is optional; if it is omitted the default bridge is being taken from `/usr/local/DNCA/etc/dnca.conf` file as it was previously described in this section.

If this command is successful, the system will return the message “*Update settings for <bridge name>*”; after that it will return you back to the command prompt (#).

To show a list of all or specific WYDE settings, i.e. system parameters and their values, using the command line, you should use the *wyde* command line utility with the *settings-show* option. The syntax is as follows:

```
wyde settings-show <arguments>
```

Each of the arguments is followed by a space and a value. In *settings-show* you can specify the following arguments:

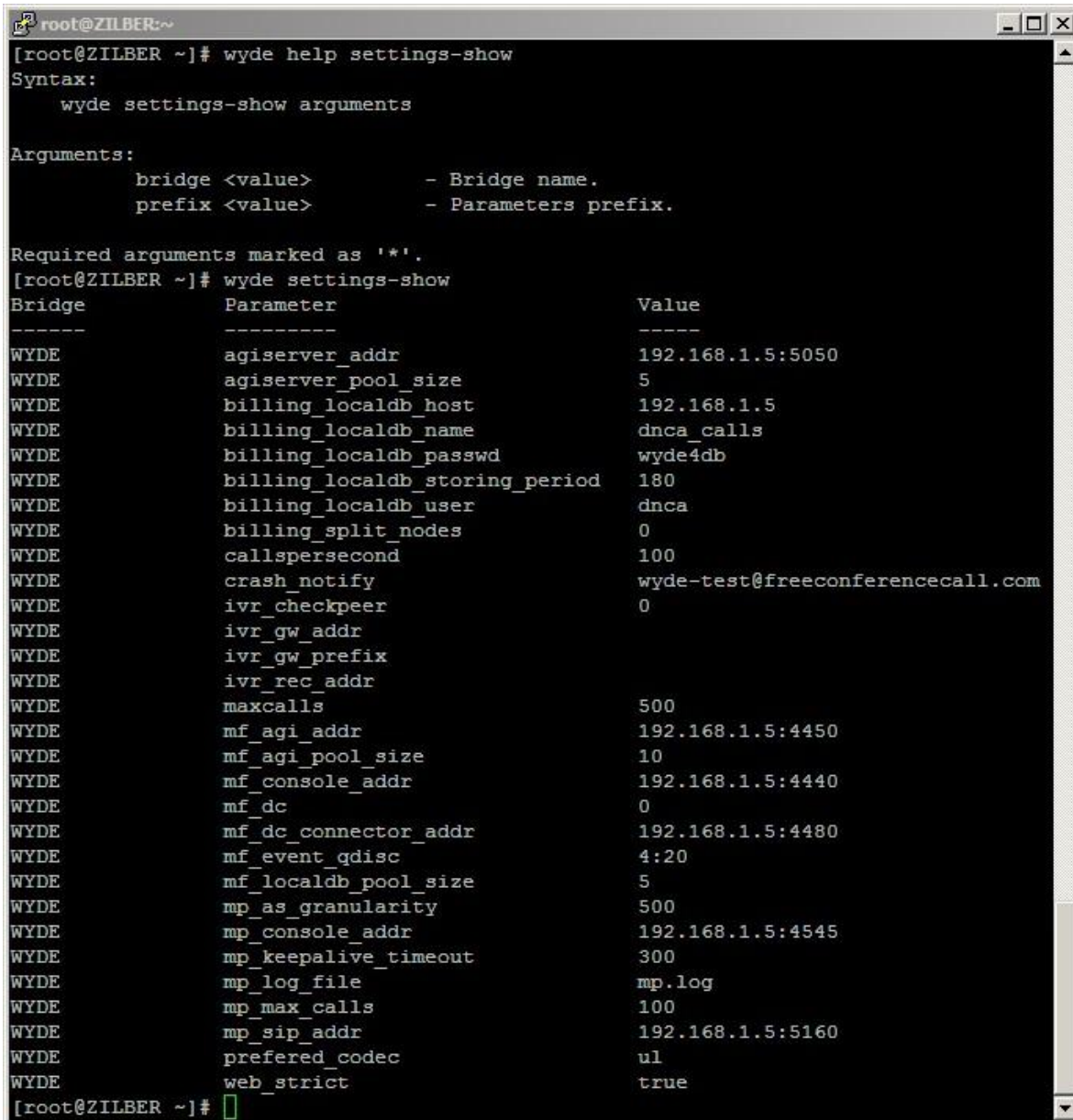
- `bridge <value>` – The name of the bridge which settings you would like to show.
- `prefix <value>` – The prefix of the parameters names that should be shown.

All arguments are optional. If argument `bridge` is omitted the settings will be shown for the current bridge; the default bridge is being taken from `/usr/local/DNCA/etc/dnca.conf` file as it was previously described in this section. If argument `prefix` is omitted all bridge parameters will be shown; if this argument is specified the command shows only parameters that starts with the specified prefix.

For example if you would like to see all current WYDE bridge settings you should use the command:

```
wyde settings-show
```

This command outputs all system parameters for the current bridge, see Figure 50 for details. As you can see, the *wyde settings-show* command shows the bridge name, parameters names and their current values.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help settings-show
Syntax:
    wyde settings-show arguments

Arguments:
    bridge <value>          - Bridge name.
    prefix <value>          - Parameters prefix.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde settings-show
Bridge      Parameter      Value
-----
WYDE        agiserver_addr    192.168.1.5:5050
WYDE        agiserver_pool_size 5
WYDE        billing_localdb_host 192.168.1.5
WYDE        billing_localdb_name dnca_calls
WYDE        billing_localdb_passwd wyde4db
WYDE        billing_localdb_storing_period 180
WYDE        billing_localdb_user dnca
WYDE        billing_split_nodes 0
WYDE        callspersecond    100
WYDE        crash_notify      wyde-test@freeconferencecall.com
WYDE        ivr_checkpeer      0
WYDE        ivr_gw_addr
WYDE        ivr_gw_prefix
WYDE        ivr_rec_addr
WYDE        maxcalls          500
WYDE        mf_agi_addr        192.168.1.5:4450
WYDE        mf_agi_pool_size   10
WYDE        mf_console_addr    192.168.1.5:4440
WYDE        mf_dc              0
WYDE        mf_dc_connector_addr 192.168.1.5:4480
WYDE        mf_event_qdisc     4:20
WYDE        mf_localdb_pool_size 5
WYDE        mp_as_granularity  500
WYDE        mp_console_addr    192.168.1.5:4545
WYDE        mp_keepalive_timeout 300
WYDE        mp_log_file        mp.log
WYDE        mp_max_calls       100
WYDE        mp_sip_addr        192.168.1.5:5160
WYDE        preferred_codec    ul
WYDE        web_strict         true
[root@ZILBER ~]#

```

Figure 50: *wyde help settings-show* and *wyde settings-show* Commands Output Sample

Bridge Configuration Changes

If all WYDE bridge software components are installed on one single computer and the IP address of this computer has been changed you can update this IP address in all configuration parameters using the single *wyde* command line utility run with the *set-ip* option. The syntax is as follows:

```
wyde set-ip <arguments>
```

Each of the arguments is followed by a space and a value. In *set-ip* you can specify the following arguments:

- *ip <value>* – New IP address of the bridge. This argument is required.

- `bridge <value>` – The name of the bridge which IP address should be changed. This argument is optional; if it is omitted the default bridge is being taken from `/usr/local/DNCA/etc/dnca.conf` file as it was previously described in this section. This command updates the following settings parameters (the parameters from `bridge_settings` table): `agiserver_addr`, `billing_localdb_host`, `mf_agi_addr`, `mf_console_addr`, `mf_dc_connector_addr`, `mp_console_addr`, `mp_sip_addr` with new IP address; it also updates `ip_addr` field value in `bridges` table and `ivr_addr` field value in `nodes` table. In addition this command updates *PostgreSQL* configuration file `pg_hba.conf`, *tomcat* configuration files and the IP information in the following files: `opt/tomcat/webapps/wydevoice/index.html`, `/opt/tomcat/webapps/wydevoice/playback.html`, `/opt/tomcat/webapps/supportmodule/WEB-INF/classes/supportmodule.properties`.

If you would like to change your notification email address using the command line, you should use the `wyde` command line utility with the `set-email` option. The syntax is as follows:

```
wyde set-email <arguments>
```

Each of the arguments is followed by a space and a value. In `set-email` you can specify the following arguments:

- `email <value>` – New notification email address for the bridge. This argument is required.
- `bridge <value>` – The name of the bridge which notification email address should be changed. This argument is optional; if it is omitted the default bridge is being taken from `/usr/local/DNCA/etc/dnca.conf` file as it was previously described in this section.

This command updates settings parameter `crash_notify` (the parameters from `bridge_settings` table) and `MAILTO` parameter in `/etc/crontab` file.

Dialout Settings Configuration

The following system parameters are responsible and should be configured to perform dial-out:

- `ivr_gw_addr` – denotes the gateway address (IP) where INVITE should be sent during the dial-out, if you are making dial-out to the real phone number the specified gateway should have PSTN output;
- `ivr_gw_prefix` – denotes the prefix that should be added to the beginning of the number during the dial-out or the template for replace.

For example if `ivr_gw_addr="192.168.1.88"` and `ivr_gw_prefix=""` (empty) the dialing out to the number `6046880331` actually will send INVITE to the specified gateway with *To* field equal `6046880331@192.168.1.88`. If, for instance, `ivr_gw_prefix="9"` this prefix will be added to the beginning of the number, i.e. *To: 96046880331@192.168.1.88*. If, for instance, `ivr_gw_prefix="00:11"` `00` in the number will be replaced with `11` and the dialing out to the number `006046880331` will be made as *To: 116046880331@192.168.1.88* (`00` will be replaced with `11`).

To set this gateway address for your bridge you can use the following `wyde` command:

```
wyde settings-edit name ivr_gw_addr value 192.168.1.88
```

In addition for testing purposes you can use the gateway *voice.freeconferencecallhd.com* and perform dialout to the number *9512624343*, for example using *wyde* command:

```
wyde settings-edit name ivr_gw_addr value  
voice.freeconferencecallhd.com
```

and using *MF* console command:

```
dialout 9512624343 90 667788 12 6602
```

WYDE Bridge Configuration Save and Restore

The WYDE bridge software allows you to save the currently installed and configured system. This includes:

- Main database *dnca*;
- Billing database *dnca_calls*;
- Contents of the folder */usr/local/DNCA/etc*;
- Contents of the folder */usr/local/DNCA/var*;
- Configuration files of *asterisk*.

If you would like to save the current bridge settings using the command line, you should use the *wyde* command line utility with the *config-save* option. The syntax is as follows:

```
wyde config-save file <target file name>
```

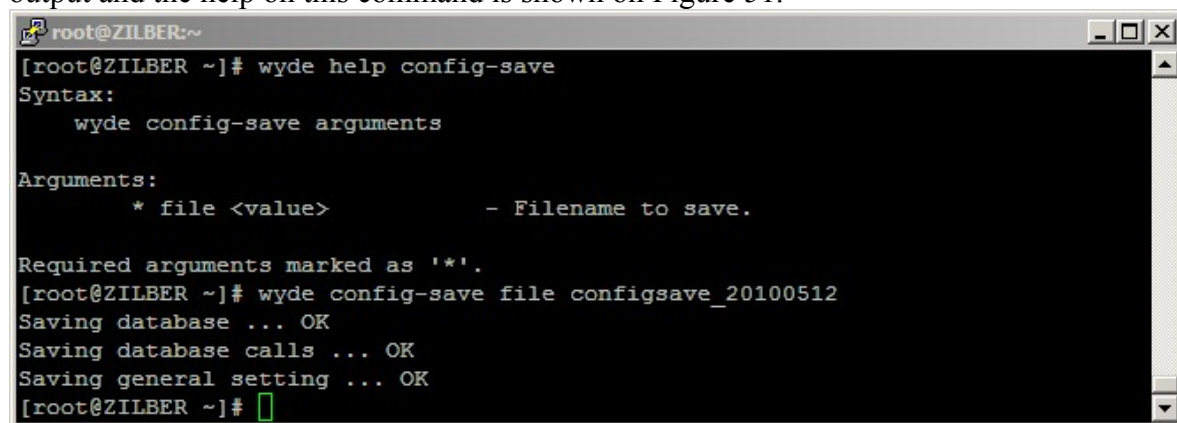
where

- *<target file name>* – The file name to save configuration settings. This argument is required.

For example to save the current WYDE bridge configuration into *configsave_20100512* file you should run the command:

```
wyde config-save file configsave_20100512
```

If the command is successful, you will be informed about each step that performs the command, i.e. you receive the messages when main database is saved, when billing database is saved and when general system settings are saved; after that the command will return you back to the command prompt (#). The sample of the *config-save* command output and the help on this command is shown on Figure 51.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help config-save
Syntax:
    wyde config-save arguments

Arguments:
    * file <value>          - Filename to save.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde config-save file configsave_20100512
Saving database ... OK
Saving database calls ... OK
Saving general setting ... OK
[root@ZILBER ~]#

```

Figure 51: *wyde help config-save* and *wyde config-save* Commands Output Sample

To restore previously saved configuration settings from the file using the command line, you should use the *wyde* command line utility with the *config-restore* option. The syntax is as follows:

```
wyde config-restore <arguments>
```

Each of the arguments is followed by a space and a value. In *config-restore* you can specify the following arguments:

- *file* <source file name> – The file name to restore configuration settings. This argument is required.
- *force* <value> – Force required {yes|no}:
 - *yes* denotes that the configuration file should be restored even if the versions of previously saved configuration data and currently installed WYDE software are different;
 - *no* denotes that the configuration file should be restored only if the versions of previously saved configuration data and currently installed WYDE software are the same.

This argument is optional, if the argument is omitted *force no* is used by default.

If the versions of previously saved data and currently installed WYDE software are the same this command will restore all saved settings, i.e. databases *dnca* and *dnca_calls*, folders */usr/local/DNCA/etc* and */usr/local/DNCA/var*, and *asterisk* configuration files. If these versions are different and parameter *force* either omitted or equal *no*, you will receive the error message about versions mismatch and restore will be canceled. If these versions are different and parameter *force* equal *yes*, you will receive the warning message about versions mismatch, but restore still will be processed.

Nodes Administration

In terms of WYDE bridge *MF* service, the *node* is the computer with the *asterisk* service installed and running. The *asterisk* is being installed in *Frontend* components installation. If you are performing cluster installation you can have multiple nodes, i.e. multiple *asterisk* computers in your WYDE bridge environment. Note that the version of the WYDE software must be the same on all nodes of you bridge.

If you are configuring the large scale bridges, i.e. the bridges that should support large amount of simultaneous calls, the single bridge resources may be insufficient. In this case you may consider using multiple nodes for you bridge. Later in this section we will describe how to install and configure additional node for you bridge.

To add new node definition on the bridge using the command line, you should use the *wyde* command line utility with the *node-add* option. The syntax is as follows:

```
wyde node-add <arguments>
```

Each of the arguments is followed by a space and a value. In *node-add* you can specify the following arguments:

- *bridge* <value> – The name of the bridge to which you would like to add the node. If this argument is omitted the node will be added for the current bridge.
- *name* <value> – The name of the node that should be added.

- `ivr_addr <value>` – The node IVR address, i.e. IP address and port of the node.
- `ivr_maxcalls <value>` – The maximum number of calls on the IVR for the node.
- `zone <value>` – The location zone for the node.

Arguments `name` and `ivr_addr` are required. The arguments can be transferred to this command in any order.

To update the properties of existing WYDE bridge node using the command line, you should use the *wyde* command line utility with the *node-set* option. The syntax is as follows:

```
wyde node-set <arguments>
```

Each of the arguments is followed by a space and a value. In *node-set* you can specify the following arguments:

- `bridge <value>` – The name of the bridge which node properties should be updated. If this argument is omitted the node for the current bridge will be taken.
- `name <value>` – The name of the node which properties should be updated.
- `ivr_addr <value>` – The node new IVR address, i.e. IP address and port of the node.
- `ivr_maxcalls <value>` – New maximum number of calls on the IVR for the node.
- `zone <value>` – New location zone for the node.

Argument `name` is required. Other arguments should be transferred only if you would like to update the specific node property. The arguments can be transferred to this command in any order.

To drop the existing WYDE bridge node definition using the command line, you should use the *wyde* command line utility with the *node-del* option. The syntax is as follows:

```
wyde node-del <arguments>
```

Each of the arguments is followed by a space and a value. In *node-del* you can specify the following arguments:

- `bridge <value>` – The name of the bridge which node should be deleted. If this argument is omitted the node will be deleted from the current bridge.
- `name <value>` – The name of the node that should be deleted.

Argument `name` is required. The arguments can be transferred to this command in any order.

For these *node-add*, *node-set*, and *node-del* commands the default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file:

```
[general]
bridge_id = 1
```

If the argument `bridge` is omitted in these commands the bridge defined in *dnca.conf* file will be taken as default.

If any of these commands is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#).

If you added new node, updated node properties, or deleted existing node you should reload *MF* cluster nodes list, i.e. reload *nodes* table from the database. To send the signal on the WYDE bridge *MF* engine to reload all nodes the following *mf* console command should be executed:

```
node-reload
```

If new *asterisk* computer was added this command will connect and turn on this node. If the properties of existing nodes were changed they will be renewed and actualized. If existing node was deleted all conferences and calls on this node will be dropped and the node will be disconnected and turned off.

If you would like to show nodes configuration in the database, you should use the *wyde* command line utility with the *node-show* option. The syntax is as follows:

```
wyde node-show [bridge <bridge>]
```

where

- *<bridge>* – the name of the bridge which nodes you would like to show. This argument is optional.

If you would like to show current nodes status, you should use the following *mf* console command:

```
node-show
```

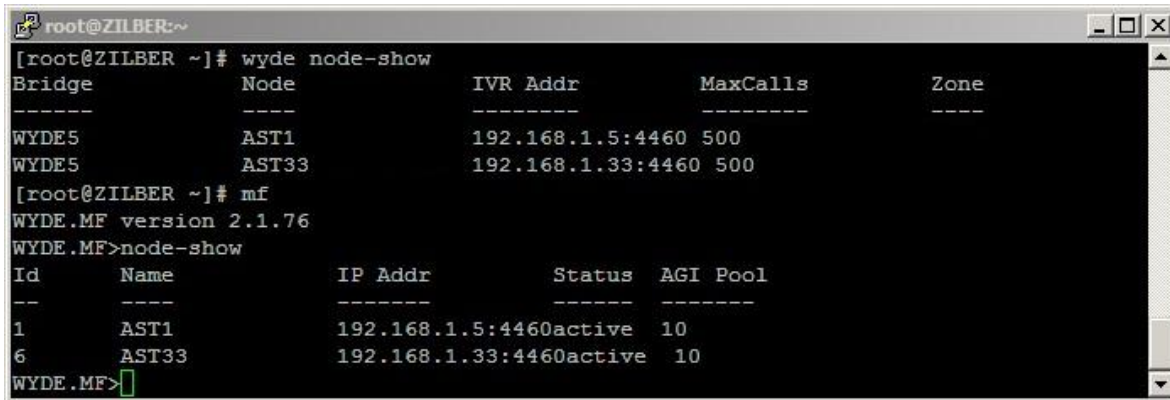
Let's assume that you have the bridge with full installation of the WYDE software on IP *192.168.1.5* and you would like to add new node on IP *192.168.1.33*. To do that first of all you should install on the *192.168.1.33* computer the *Frontend* components of the WYDE software, i.e. install the *asterisk* service. After that on your *192.168.1.5* computer you should run the following *wyde* command to add this node registration:

```
wyde node-add name AST33 ivr_addr 192.168.1.33:4460
```

Next you should run *mf* console command *node-reload* to send the signal on the WYDE bridge *MF* engine to reload all nodes.

As soon as these steps have been made you are able to call either on *192.168.1.5* or *192.168.1.33*.

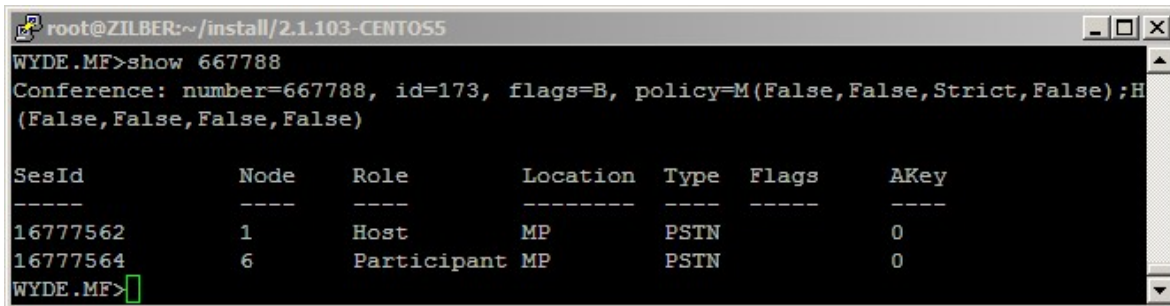
Samples of *wyde node-show* and *mf* console *node-show* commands are shown on Figure 52. Please see nodes identifiers in *mf* console *node-show* command output: *1* – for *192.168.1.5* and *6* – for *192.168.1.33*; when you are looking the conference information these identifiers are shown in *mf* Console *show*, column *Node*. Figure 53 shows two conference calls, as you can see the *host* call has been made to the node *1* (*192.168.1.5*) and the *participant* call has been made to the node *6* (*192.168.1.33*).



```

root@ZILBER:~
[ root@ZILBER ~]# wyde node-show
Bridge      Node      IVR Addr      MaxCalls      Zone
-----
WYDE5      AST1      192.168.1.5:4460 500
WYDE5      AST33     192.168.1.33:4460 500
[ root@ZILBER ~]# mf
WYDE.MF version 2.1.76
WYDE.MF>node-show
Id      Name      IP Addr      Status      AGI Pool
--
1      AST1      192.168.1.5:4460 active    10
6      AST33     192.168.1.33:4460 active    10
WYDE.MF>

```

Figure 52: *wyde node-show* and *mf* Console *node-show* Commands Output Sample


```

root@ZILBER:~/install/2.1.103-CENTOS5
WYDE.MF>show 667788
Conference: number=667788, id=173, flags=B, policy=M(False,False,Strict,False);H
(False,False,False,False)

SesId      Node      Role      Location      Type      Flags      AKey
-----
16777562    1      Host      MP      PSTN      0
16777564    6      Participant MP      PSTN      0
WYDE.MF>

```

Figure 53: *mf* Console *show* Conference – Different Nodes Calls Command Output Sample

Distributed Conferencing Administration

In terms of WYDE bridge, the *distributed conference* is the conference that is taking place on the multiple bridges simultaneously. That means that the calls are being made to the different bridges, but these calls are participating in the same conference.

You may need to use distributed conferences if your users are geographically located remotely from each other in different regions or countries and you prefer having the local bridges for them. In this case the users will be able to call to the local bridges, but their calls could be joined into distributed conferences.

This section contains only basic information about distributed conferencing configuration. If you would like to use distributed conferences please contact WYDE Voice technical support for detail instructions and assistance in distributed conferencing configuration for the WYDE Voice conferencing bridge software.

The following steps and conditions should be implemented if you are going to use distributed conferences:

- all necessary bridges that perform distributed conferences must have the same version of the WYDE software installed on them;
- all necessary bridges that perform distributed conferences should be added to the *bridges* table on all bridges that you use, you can use *wyde bridge-add* command for these purposes;

- the identifiers of your bridges (*id* field in *bridges* table) should be between 1 and 15;
- the identifiers of your bridges should be the same on all bridges that are performing distributed conferences;
- the system parameter *mf_dc* should be set 1 on all these bridges, see section WYDE Bridge Settings Management for instructions how to set the system parameter value;
- after that the bridge sends notifications about the conferences to all bridges described in *bridges* table;
- call flow attribute *conference_distributed* (Distributed conferencing) should be equal “on” either on call flow or on DNIS or on conference level for the conference where you are calling;
- if another bridge has the conference with the same conference number, and *conference_distributed* call flow attribute for this conference equal “on”, and *mf_dc* parameter is equal to 1 on that bridge as well, these conferences are being joined into single distributed conference.

The following algorithm is being applied for the distributed conferences:

- the bridge knows identifiers, IP addresses and ports of all bridges, that are participating in distributed conferencing;
- the bridge announces all own conferences each minute and when new conference is being created to all other bridges;
- to send these announcements UDP datagrams are being used, these datagrams have their own specific format;
- if the conference with the same conference number and *conference_distributed* attribute equal “on” exists on two or more bridges the master bridge is being selected for this conference, the master bridge is the bridge with minimal *create_time*, if it is the same then the master bridge is the bridge with minimal *ID*;
- subordinate bridges make connection with the master bridge using real time protocol;
- real time clients translate all RT commands to another bridges using control calls, by this approach the status of distributed conference is being kept the same on all bridges;
- if during the distributed conference the subordinate bridge RT client loses the master bridge, the existing master bridge is being canceled and new master bridge is being selected again using the same algorithm that was described above.

To add new bridge definition you should use the *wyde* command line utility with the *bridge-add* option. The syntax is as follows:

```
wyde bridge-add <arguments>
```

Each of the arguments is followed by a space and a value. In *bridge-add* you can specify the following arguments:

- *name* <value> – The name of the bridge that should be added.
- *ip_addr* <value> – IP address of the bridge that should be added.

All these arguments are required. The arguments can be transferred to this command in any order.

To delete existing bridge definition you should use the *wyde* command line utility with the *bridge-del* option. The syntax is as follows:

```
wyde bridge-del name <bridge name>
```

where

- *<bridge name>* – The required name of the bridge that should be deleted.

To see all registered WYDE bridges, i.e. the bridges described in *bridges* table you should use the *wyde* command line utility with the *bridge-show* option. The syntax is as follows:

```
wyde bridge-show
```

Let's assume that you would like to configure distributed conferences between two bridges: *192.168.1.5* (let the name of the bridge is *WYDE5* and identifier of the bridge is *1*) and *192.168.1.31* (let the name of the bridge *WYDE31* and identifier of the bridge is *2*). To do that you should run the following *wyde* command on both bridges:

```
wyde bridge-add name WYDE31 ip_addr 192.168.1.31
```

After that you should manually update *bridges* table using the *PostgreSQL* interactive terminal (database console). On the first (*WYDE5*) bridge you should run the command:

```
UPDATE bridges SET "name"='WYDE5' WHERE id=1
```

On the second (*WYDE31*) bridge you should run the commands:

```
UPDATE bridges SET "name"='WYDE5', ip_addr='192.168.1.5'
WHERE id=1
```

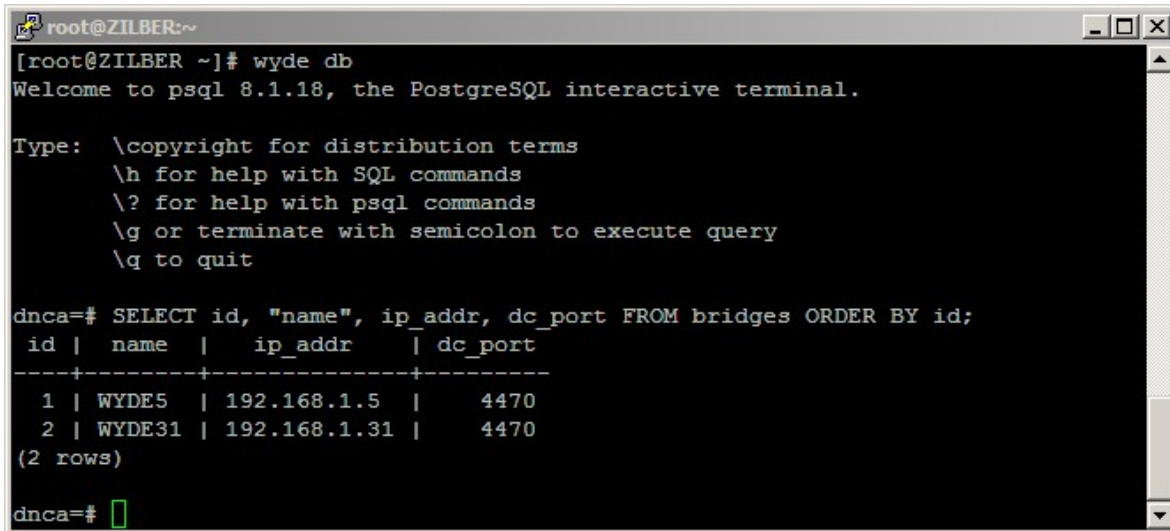
```
UPDATE bridge_settings SET bridge_id=2
```

```
UPDATE nodes SET bridge_id=2
```

The last two commands should be executed because we are changing the current bridge identifier from *1* to *2* for this second bridge. In addition you should change the current bridge ID in the */usr/local/DNCA/etc/dnca.conf* file:

```
[general]
bridge_id = 2
```

After these changes the content of *bridges* table will be exactly the same on both bridges (see Figure 54) and both bridges will be fully functional and operational.



```

root@ZILBER:~
[root@ZILBER ~]# wyde db
Welcome to psql 8.1.18, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help with psql commands
      \g or terminate with semicolon to execute query
      \q to quit

dnca=# SELECT id, "name", ip_addr, dc_port FROM bridges ORDER BY id;
 id | name  | ip_addr | dc_port
----+-----+-----+-----
  1 | WYDE5 | 192.168.1.5 | 4470
  2 | WYDE31 | 192.168.1.31 | 4470
(2 rows)

dnca=#

```

Figure 54: *bridges* Table – Distributed Conferences Configuration

Next you should switch on distributed conferences configuration mode on and set the system parameter *mf_dc* equal *1*; the following *wyde* command should be used for that:

```
wyde settings-edit name mf_dc value 1
```

After that you are able to configure the distributed conference for a subscriber. This conference should have the same conference number on both bridges, other conference properties could be different, but for this sample we configure the conference accounts exactly the same. Let's assume that the conference number is *889900*, the subscriber PIN is *nrobert*, the DNIS number to call is *8665080012*, host access code is *505051* and participant access code is *505052*.

```

wyde confuser-add subscriber nrobert did 8665080012
      conference 889900 accesscode 505051 role Host
wyde confuser-add subscriber nrobert did 8665080012
      conference 889900 accesscode 505052 role Participant
wyde conference-attr-set number 889900
      name conference_distributed value on

```

Now all calls to both bridges to the conference *889900* are being joined into single distributed conference and all conference participants will be hear and talk to each other. Figure 55 shows the distributed conference on the first bridge (*WYDE5*) and Figure 56 shows the distributed conference on the second bridge (*WYDE31*). In our sample the first bridge (*WYDE5*) is being selected as master and the second bridge (*WYDE31*) is subordinate bridge. According to the algorithm described above the subordinate bridge (*WYDE31*) makes connection with the master bridge (*WYDE5*) using real time protocol. Because of that you can see on both bridges the third connection with the role *DC* – distributed conferencing and the flag *T* – real time (RT) mode.

```

root@ZILBER:~/install/2.1.103-CENTOS5
WYDE.MF>show 889900
Conference: number=889900, id=172, flags=TB, policy=M(False,False,Strict,False);
H(False,False,False,False)

SesId      Node   Role      Location  Type  Flags  AKey
-----
16777559   1      Host      MP        PSTN             0
16777561   1      DC        MP        VoIP  T       0
33554444   0      Participant IVR       PSTN             0
WYDE.MF>

```

Figure 55: Show Distributed Conference on *WYDE5* Bridge

```

root@localhost:~
WYDE.MF>show 889900
Conference: number=889900, id=10, flags=TB, policy=M(False,False,Strict,False);H
(False,False,False,False)

SesId      Node   Role      Location  Type  Flags  AKey
-----
16777559   0      Host      IVR       PSTN             0
33554444   1      Participant MP        PSTN             0
33554445   1      DC        MP        VoIP  T       0
WYDE.MF>

```

Figure 56: Show Distributed Conference on *WYDE31* Bridge

If distributed conferencing is switched on you can see all bridges that can participate in distributed conferences using *mf* console command:

`dc-show-bridges`

This command returns bridge identifier, name, IP address, and activity information.

Activity column shows how many seconds ago the last announcement was received from the bridge; if there were no any announcements from the bridge this column contains “Never” keyword.

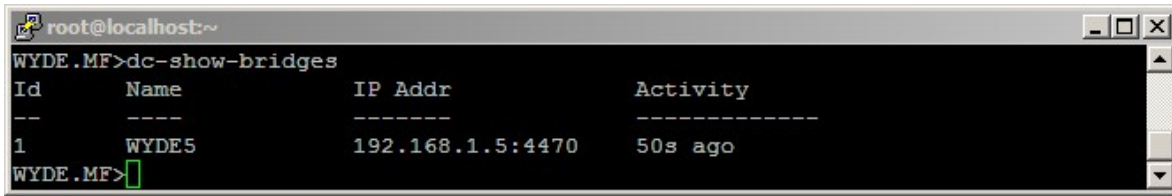
Figure 57 shows DC bridges information on the bridge *WYDE5*, the master bridge in the started conference; the first row shows the current bridge and the second row shows information about the subordinate bridge *WYDE31* and it tells that the last announcement was received from that bridge 27 seconds ago. Figure 58 shows DC bridges information on the bridge *WYDE31*, the subordinate bridge in the started conference; it has the single row with information about the master bridge *WYDE5* and it tells that the last announcement was received from that bridge 50 seconds ago.

```

root@ZILBER:~/install/2.1.103-CENTOS5
WYDE.MF>dc-show-bridges
Id      Name      IP Addr      Activity
--      -
2866183552 192.168.1.5:0 750972h ago
2      WYDE31    192.168.1.31:4470 27s ago
WYDE.MF>

```

Figure 57: *mf* Console `dc-show-bridges` Command Output Sample on *WYDE5* Bridge



```

root@localhost:~
WYDE.MF>dc-show-bridges
Id      Name      IP Addr      Activity
--      -
1       WYDE5      192.168.1.5:4470  50s ago
WYDE.MF>

```

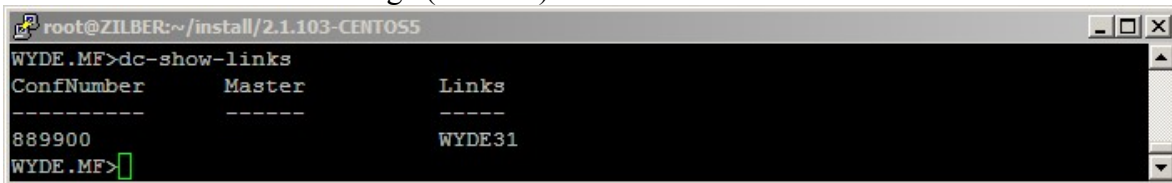
Figure 58: *mf* Console *dc-show-bridges* Command Output Sample on *WYDE31* Bridge

If distributed conferencing is switched on and there are distributed conferences running on the bridge you can see all distributed conferences using *mf* console command:

dc-show-links

For all started distributed conferences this command outputs conference number, master bridge name, and comma-separated linked bridges names. If the command is being run on the master bridge the Master column is empty.

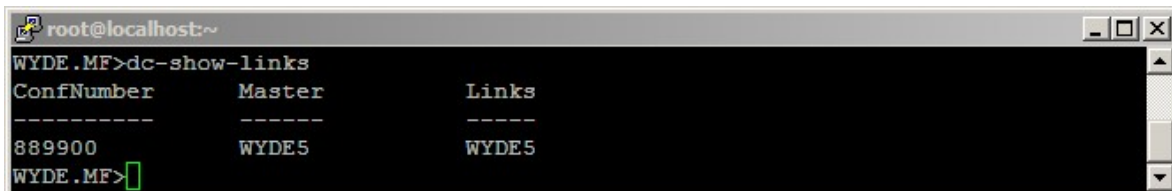
Figure 59 shows output of this command that was run on master bridge (*WYDE5*), because of that the Master column is empty and the Links column contains information about the subordinate bridge (*WYDE31*). Figure 60 shows output of this command that was run on the subordinate bridge (*WYDE31*), because of that both columns Master and Links contain information about master bridge (*WYDE5*).



```

root@ZILBER:~/install/2.1.103-CENTOS5
WYDE.MF>dc-show-links
ConfNumber  Master      Links
-----
889900      WYDE31
WYDE.MF>

```

Figure 59: *mf* Console *dc-show-links* Command Output Sample on *WYDE5* Bridge


```

root@localhost:~
WYDE.MF>dc-show-links
ConfNumber  Master      Links
-----
889900      WYDE5      WYDE5
WYDE.MF>

```

Figure 60: *mf* Console *dc-show-links* Command Output Sample on *WYDE31* Bridge

Peers Management

In terms of WYDE bridge *MF* service, the *peer* is the SIP client from which the bridge accepts the calls. Peers information is being stored in *peers* table. If you would like to configure the WYDE bridge to accept not all calls, but only the calls from specific IP address you should update *bridge_settings* table and set *ivr_checkpeer* parameter value equal to 1 (default value is 0, no peers limitation applied) and populate *peers* table with allowed IP address together with other information.

If you added new peers into *peers* table you should reload them from the database. To send the signal on the WYDE bridge *MF* engine to reload all peers from the *peers* table the following *mf* console command should be executed:

peer-reload

If the command is successful, the system will not return any errors or messages.

Calls Transferring

If you would like to transfer started conference calls from one WYDE bridge or bridge frontend to another (for instance if you need to switch off the specific frontend or entire bridge), you can make this transfer using the following *mf* console command:

```
transfer {node <node_id>|node all} | {conference <conf_number>} |  
        {did <did>} <destination_ip>
```

where

- One of the following arguments should be specified as the first argument of this command:
 - {node <node_id>|node all} – denotes the node identifier (or all nodes) from which the calls should be transferred;
 - {conference <conf_number>} – denotes the conference number for which the calls should be transferred;
 - {did <did>} – denotes the DNIS (DID) number for which the calls should be transferred.
- <destination_ip> – Denotes destination IP address of the bridge or bridge frontend to which the specified calls should be transferred. This argument is required.

You can also use *wyde* command line utility with the *transfer* option for the same purposes. See *wyde* Command Reference: *transfer* (Transfer Calls) for detail command information and parameters format.

After this command execution all conference calls that currently are started on the bridge and fit with the criteria that is specified as the first argument of this command will be transferred to the bridge or bridge frontend specified as the second argument of this command.

Audio Prompts Management

The WYDE bridge software allows you using your own audio prompts as welcome prompt and music-on-hold prompt in the conferences. If you would like using your own audio files you should perform the steps described in this section of the guide.

Custom audio files should be placed into the following subfolders of */usr/local/DNCA/var* folder:

- *welcome-prompts* (*/usr/local/DNCA/var/welcome-prompts*) – the folder for welcome prompts audio files;
 - *moh* (*/usr/local/DNCA/var/moh*) – the folder for music-on-hold prompts audio files.
- Thus you should place your audio files in required formats (**.wav*, **.ul*, **.al*, etc.) into these folders.

Note, that if you have file in *wav* format only, you can convert it into other formats on your WYDE server using *transcoder.x* utility:

```
USAGE: /usr/local/DNCA/bin/transcoder.x [options]
```

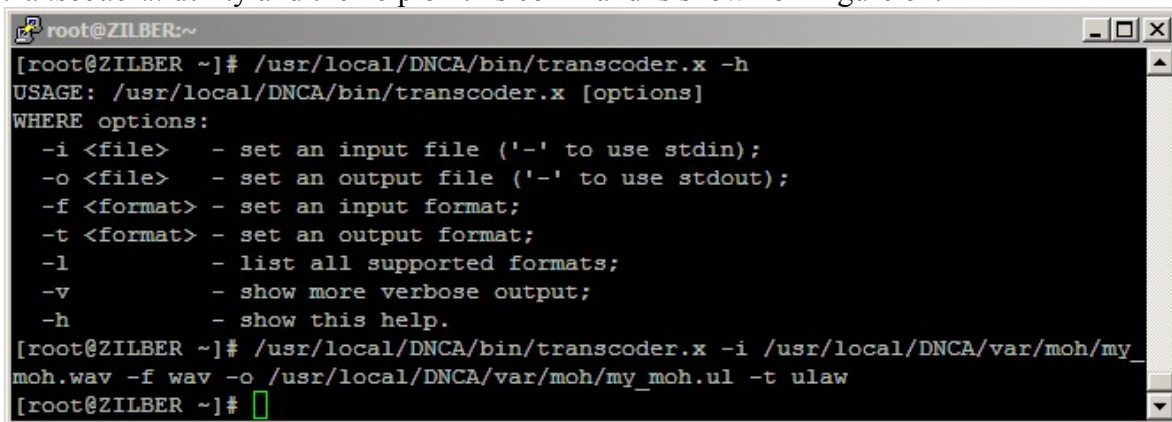
WHERE options:

```
-i <file>    - set an input file ('-' to use stdin);
-o <file>    - set an output file ('-' to use stdout);
-f <format>  - set an input format;
-t <format>  - set an output format;
-l          - list all supported formats;
-v          - show more verbose output;
-h          - show this help.
```

For example if you would like to convert music-on-hold *my_moh.wav* file from *wav* format into *ulaw* format you should use the command (the transferred command arguments are shown in *italic*):

```
/usr/local/DNCA/bin/transcoder.x
  -i /usr/local/DNCA/var/moh/my_moh.wav -f wav
  -o /usr/local/DNCA/var/moh/my_moh.ul -t ulaw
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the audio file conversion using *transcoder.x* utility and the help on this command is shown on Figure 61.



```
root@ZILBER:~
[root@ZILBER ~]# /usr/local/DNCA/bin/transcoder.x -h
USAGE: /usr/local/DNCA/bin/transcoder.x [options]
WHERE options:
  -i <file>    - set an input file ('-' to use stdin);
  -o <file>    - set an output file ('-' to use stdout);
  -f <format>  - set an input format;
  -t <format>  - set an output format;
  -l          - list all supported formats;
  -v          - show more verbose output;
  -h          - show this help.
[root@ZILBER ~]# /usr/local/DNCA/bin/transcoder.x -i /usr/local/DNCA/var/moh/my_moh.wav -f wav -o /usr/local/DNCA/var/moh/my_moh.ul -t ulaw
[root@ZILBER ~]#
```

Figure 61: Audio File Conversion using *transcoder.x* Utility Sample

After you have placed all necessary audio files into *welcome-prompts* subfolder you should reload them using *mf* console command:

```
welcomeprompt-reload
```

After you have placed all necessary audio files into *moh* subfolder you should reload them using *mf* console command:

```
moh-reload
```

If any of both these commands was implemented successfully you will receive the message: “*Success*”.

Once you reloaded your audio prompts you can use them in your call flows, DNISes and conferences. If you would like to use your own custom welcome prompt you should update *dnis_welcomeprompt* (Welcome prompt) call flow attribute value either on call flow

level or on DNIS level with your welcome prompt audio file name (without extension). If you would like to use your own custom music-on-hold prompt you should update *conference_moh* (Music on hold) call flow attribute value either on call flow level or on DNIS level or conference account level with your music-on-hold prompt audio file name (without extension).

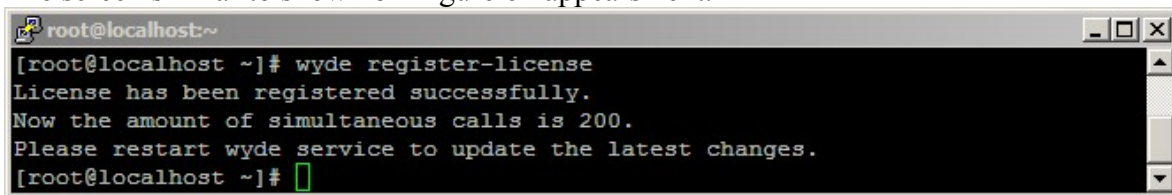
Licensing

By default the WYDE Voice conferencing bridge software has the maximum number of the simultaneous calls on the bridge equal to 10. To support larger number of calls you should obtain the additional licenses. See detail information about how to obtain additional licenses in “WYDE Software Installation Guide”, Chapter 3: Licensing.

When you obtain your licenses you receive generated license file *license.bin* (either if it is new license or renewed license). This file should be placed on the bridge into */usr/local/DNCA/etc* folder. After that you should register new license using the following command:

```
wyde register-license
```

The screen similar to shown on Figure 62 appears next.

A terminal window titled 'root@localhost:~' showing the command 'wyde register-license' and its output. The output states: 'License has been registered successfully. Now the amount of simultaneous calls is 200. Please restart wyde service to update the latest changes.' The prompt returns to '[root@localhost ~]#'.

```
[root@localhost ~]# wyde register-license
License has been registered successfully.
Now the amount of simultaneous calls is 200.
Please restart wyde service to update the latest changes.
[root@localhost ~]#
```

Figure 62: Register New/Updated Licenses on the Bridge

After that the amount of allowed simultaneous calls for the WYDE Voice conferencing bridge software will be set equal to the amount specified by your license file.

Authorization Adapters and Methods

In terms of WYDE bridge software the *Authorization Adapter* is the component (function) responsible for specifying access rights in the conferences. More formally, "to authorize" is to define access policy, i.e. the right to connect to the conference and specific role (host/moderator/listener) in the conference.

The following authorization adapters are included and supported by standard WYDE bridge software installation:

- *Free* – without authorization, anyone who called to the conference DNIS number is allowed to connect to the conference regardless of access code entered,
 - usually used for *CONF* call flow;
- *LocalDb* – authorization via local database, when the person is called to the conference DNIS number, he is being asked to enter the access code, this access code is being verified in local database (*dnca*) according to subscribers' conference accounts definitions, user roles in the conference (i.e. host, participant, listener roles) are being granted depending on DNIS numbers and access codes used,
 - usually used for *SPECTEL* call flow;

- *WYDERadius* – authorization via RADIUS server using WYDE dictionary;
 - Remote Authentication Dial In User Service (RADIUS) is a networking protocol that provides centralized Authentication, Authorization, and Accounting management for computers to connect and use a network service. RADIUS is a client/server protocol that runs in the application layer, using UDP as transport. The Remote Access Server, the Virtual Private Network server, the Network switch with port-based authentication, and the Network Access Server, are all gateways that control access to the network, and all have a RADIUS client component that communicates with the RADIUS server. The RADIUS server is usually a background process running on a UNIX or Windows NT machine. RADIUS serves three functions: to authenticate users or devices before granting them access to a network, to authorize those users or devices for certain network services and to account for usage of those services.

In our case the RADIUS server receives DNIS (DID) number/access code as login/password and returns the conference number and the user roles in the returned conference as the result of authorization if it is successful; so WYDE RADIUS should contain *confuser* class (table) definition with the fields: *did_number*, *accesscode*, *conf_number*, *role* that are used to perform authorization of callers in conferences.

- *WYDEldap* – authorization via LDAP using WYDE dictionary;
 - Lightweight Directory Access Protocol, or LDAP, is an application protocol for querying and modifying data using directory services running over TCP/IP. A directory is a set of objects with attributes organized in a logical and hierarchical manner. LDAP deployments today tend to use Domain Name System (DNS) names for structuring the topmost levels of the hierarchy. Deeper inside the directory might appear entries representing people, organizational units, printers, documents, groups of people or anything else that represents a given tree entry (or multiple entries).

The following dictionary (schema) is being used for WYDE LDAP:

- object class *confUser* (conference user entry), that contains mandatory attributes *didNumber*, *accesscode*, *role*, *confNumber*;
- object class *confInfo* (conference info entry), that contains mandatory attribute *confNumber* and optional attributes *callExitDTMF*, *callInstructionsDTMF*, *callParticipantsnumberDTMF*, *callMuteDTMF*, *callAssociateDTMF*, *callOperatorDTMF*, *conferenceMuteDTMF*, *conferenceLockDTMF*, *conferenceQADTMF*, *conferenceBroadcastDTMF*, *conferenceEntryexittonesDTMF*, *conferenceDialoutDTMF*, *recordingDTMF*, *callAnnounceparticipantcount*, *conferenceEntrytones*, *conferenceExittones*, *conferenceMaxcalls*, *conferenceMoh*, *conferenceMuteHost*, *conferenceMuteParticipant*, *conferenceMuteListener*, *conferenceHoldHost*, *conferenceHoldParticipant*, *conferenceHoldListener*, *conferenceStartHow*, *conferenceStartWait*, *conferenceStopHow*, *conferenceStopWait*, *conferenceRealtime*, *conferenceCallerdb*, *recordingStopHow*, *recordingStopWait*, *callJobcodeonenter*, *conferenceJobcodeDTMF*, *conferenceRollcall*, *callGainIncDTMF*, *callGainDecDTMF*, *conferencePlayFile*.

You can populate your data using this LDAP dictionary (hierarchical database) and use them in your *WYDEldap* authorization adapter to perform authorization of callers in conferences.

Actually all authorization adapters are routines written in Perl that perform authorization using specific protocols. These routines are placed in the */usr/local/DNCA/lib/Auth/Adapter*

folder that should contain the files *<Adapter Name>.pm* that means that this folder on your bridge contains the following files: *Free.pm*, *LocalDb.pm*, *WYDERadius.pm*, *WYDELDAP.pm* – the authorization adapters supported by your bridge.

If you have your own security infrastructure the customized authorization adapter can be written to integrate your security into call flows authorization.

For authorization in the conferences are being used authorization methods. Authorization methods determines the specific authorization adapter and if necessary its parameters that are used to perform authorization. The authorization method name should be selected in `dnis_authorizemethod` (Authorize method) call flow attribute value either on call flow or on DNIS level.

If you made any changes in authorization adapters or authorization methods you should restart *agiserver* service to actualize these changes, using the following command:

```
service agiserver restart
```

Add an Authorization Adapter

Before you add new authorization adapter, you should create the *<Adapter Name>.pm* file in the */usr/local/DNCA/lib/Auth/Adapter* folder for this adapter as it was described above.

To add new authorization adapter registration using the command line interface you should use the *wyde* command line utility with the *auth-adapter-add* option. The syntax is as follows:

```
wyde auth-adapter-add <arguments>
```

Each of the arguments is followed by a space and a value. In *auth-adapter-add* you can specify the following arguments:

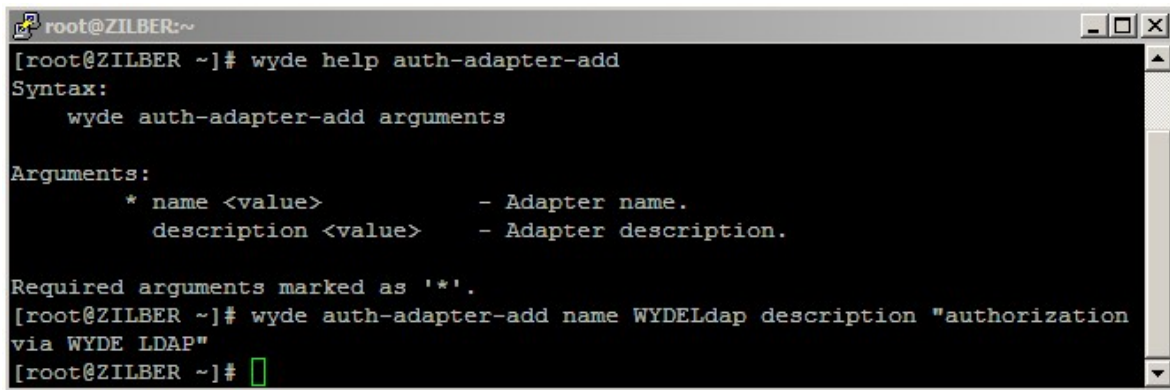
- `name <value>` – The name of the authorization adapter that should be added. This is required argument. This name should be unique, i.e. there should no be any other authorization adapter with the same name on the bridge.
- `description <value>` – The optional description of the authorization adapter that should be added.

The arguments can be transferred to this command in any order.

Let's assume that we have created the file *WYDELDAP.pm* in the folder */usr/local/DNCA/lib/Auth/Adapter* for new authorization adapter *WYDELDAP*. To add this adapter to the bridge you should use the command:

```
wyde auth-adapter-add name WYDELDAP
description "authorization via WYDE LDAP"
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *auth-adapter-add* command output and the help on this command is shown on Figure 63.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help auth-adapter-add
Syntax:
    wyde auth-adapter-add arguments

Arguments:
    * name <value>           - Adapter name.
    description <value>      - Adapter description.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde auth-adapter-add name WYDEldap description "authorization
via WYDE LDAP"
[root@ZILBER ~]#

```

Figure 63: *wyde help auth-adapter-add* and *wyde auth-adapter-add* Commands Output Sample

Delete an Authorization Adapter

To delete an authorization adapter using the *wyde* command line utility you should use *auth-adapter-del* option. The syntax is as follows:

```
wyde auth-adapter-del name <authorization adapter name>
```

where

- <authorization adapter name> – the name of the authorization adapter you wish to delete.

Note that you can delete only authorization adapters that are not in use, i.e. there should no be any authorization methods that refer to this authorization adapter. If the authorization adapter is used by any authorization method you will receive the error and the deletion will be cancelled.

For example to delete authorization adapter *VSRRadius* you should run the command:

```
wyde auth-adapter-del name VSRRadius
```

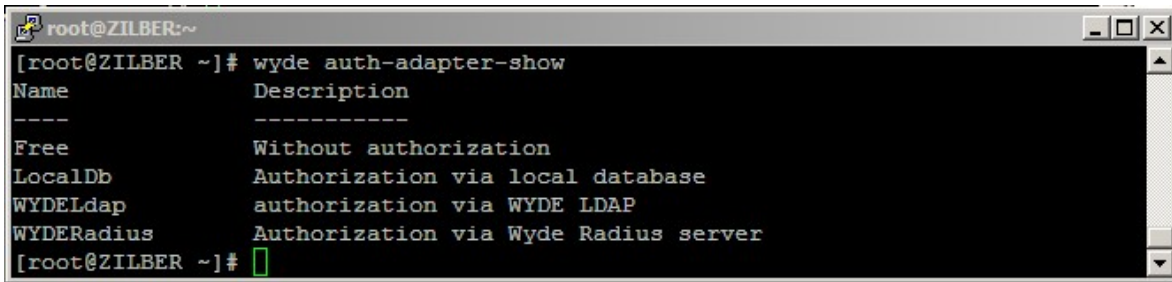
If deletion is successful, you will be returned to the command line with no additional prompts.

View Authorization Adapters

To show a list of all authorization adapters in the system using the command line, you should use the *wyde* command line utility with the *auth-adapter-show* option. The syntax is as follows:

```
wyde auth-adapter-show
```

This command will output a list of the all existed authorization adapters on the system, similar to shown on Figure 64. As you can see, the *wyde auth-adapter-show* command shows the authorization adapters that have been created in the system as well as their basic properties: authorization adapter name and description.



```

root@ZILBER:~
[ root@ZILBER ~]# wyde auth-adapter-show
Name                Description
-----
Free                Without authorization
LocalDb             Authorization via local database
WYDELDAP            authorization via WYDE LDAP
WYDERadius          Authorization via Wyde Radius server
[ root@ZILBER ~]#

```

Figure 64: *wyde auth-adapter-show* Command Output Sample

Add an Authorization Method

To create new authorization method for the authorization adapter using the command line interface you should use the *wyde* command line utility with the *auth-method-add* option. The syntax is as follows:

```
wyde auth-method-add <arguments>
```

Each of the arguments is followed by a space and a value. In *auth-method-add* you can specify the following arguments:

- *name* <value> – The name of the authorization method that should be added.
- *description* <value> – The description of the authorization method that should be added.
- *adapter* <value> – The authorization adapter name for the authorization method that should be added.
- *parameters* <value> – The list of parameters for the authorization method that should be added. The parameters are specific for authorization adapters that are being used: *Free* and *LocalDb* authorization adapters do not require any parameters; for *WYDELDAP* it is the string that defines the list of LDAP servers separated by semicolon (;), and each of these servers is defined as
 <server IP>[:<server port>]:<password>:<LDAP root DN path>
 (default port is 389, DN path means distinguished name of the LDAP folder that contains conference authorization info); for *WYDERadius* it is the string that defines the list of RADIUS servers separated by semicolon (;), and each of these servers is defined as <password>@<server IP>[:<server port>] (default port is 1812).

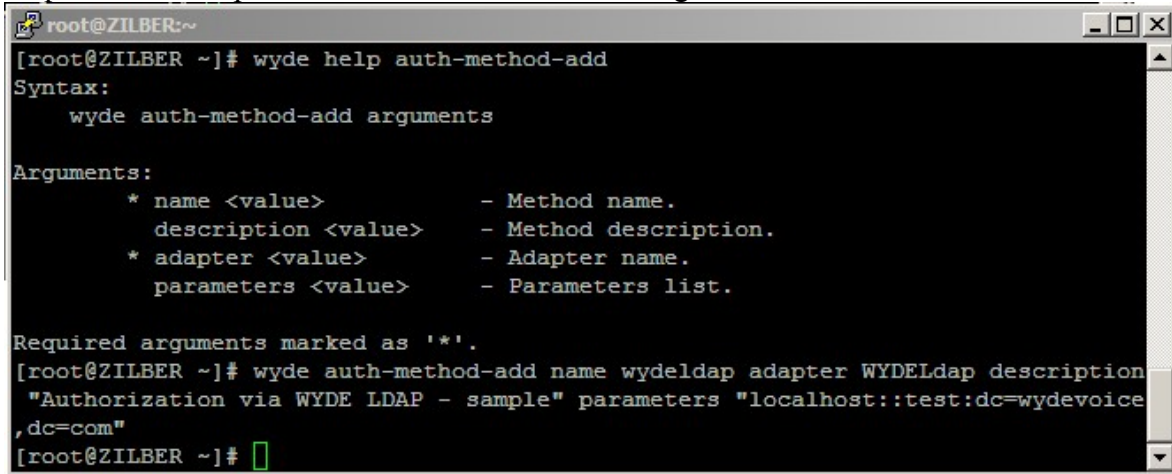
Arguments *name* and *adapter* are required. The arguments can be transferred to this command in any order.

For example if you would like to create the authorization method *wydeldap* for the authorization adapter *WYDELDAP* with description “*Authorization via WYDE LDAP – sample*” and parameters “*localhost::test:dc=wydevoice,dc=com*” you should run the following command (new authorization method properties are shown in *italic*):

```
wyde auth-method-add name wydeldap adapter WYDELDAP
description "Authorization via WYDE LDAP - sample"
parameters "localhost::test:dc=wydevoice,dc=com"
```

Note that to set the description that contains spaces and parameters you should use double quotes (").

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *auth-method-add* command output and the help on this command is shown on Figure 65.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help auth-method-add
Syntax:
    wyde auth-method-add arguments

Arguments:
    * name <value>           - Method name.
    description <value>      - Method description.
    * adapter <value>        - Adapter name.
    parameters <value>      - Parameters list.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde auth-method-add name wydeldap adapter WYDELdap description
"Authorization via WYDE LDAP - sample" parameters "localhost::test:dc=wydevoice
,dc=com"
[root@ZILBER ~]#

```

Figure 65: *wyde help auth-method-add* and *wyde auth-method-add* Commands Output Sample

Delete an Authorization Method

If you wish to delete the specific authorization method, you can use the *wyde* command line utility with *auth-method-del* option. The syntax is as follows:

```
wyde auth-method-del name <authorization method name>
```

where

- <authorization method name> – The name of the authorization method that should be deleted. This argument is required.

Note that you can delete only authorization methods that are not in use. If the method is used by any call flow and/or DNIS you will receive the error: “<authorization method name>: Authorization method is in use and can not be removed.”.

For example to delete the authorization method *vsrradius* you should run the command:

```
wyde auth-method-del name vsrradius
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Modify an Authorization Method

To modify authorization method properties, such as description and parameters, using the command line interface you should use the *wyde* command line utility with the *auth-method-set* option. The syntax is as follows:

```
wyde auth-method-set <arguments>
```

Each of the arguments is followed by a space and a value. In *auth-method-set* you can specify the following arguments:

- name <value> – The name of the authorization method that should be changed.
- description <value> – New description of the authorization method that should be set.

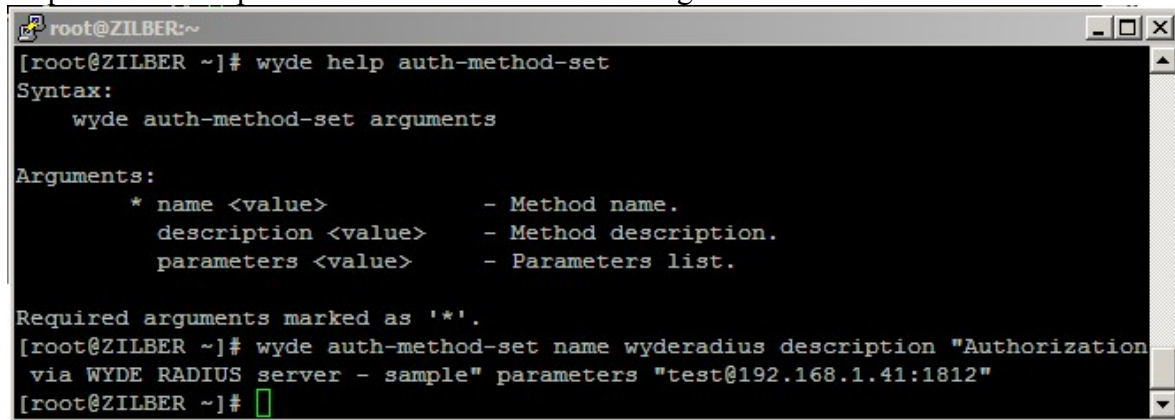
- `parameters <value>` – New list of parameters for the authorization method that should be set.

The argument name is required; you should specify arguments description and parameters only if you would like to change them. The arguments can be transferred to this command in any order.

For example if you would like to change *wyderadius* authorization method and set its description equal to “*Authorization via WYDE RADIUS server - sample*” and its parameters equal to “*test@192.168.1.41:1812*”, you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde auth-method-set name wyderadius
    description "Authorization via WYDE RADIUS server - sample"
    parameters "test@192.168.1.41:1812"
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *auth-method-set* command output and the help on this command is shown on Figure 66.



```
root@ZILBER:~
[root@ZILBER ~]# wyde help auth-method-set
Syntax:
    wyde auth-method-set arguments

Arguments:
    * name <value>           - Method name.
    description <value>     - Method description.
    parameters <value>      - Parameters list.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde auth-method-set name wyderadius description "Authorization
via WYDE RADIUS server - sample" parameters "test@192.168.1.41:1812"
[root@ZILBER ~]#
```

Figure 66: *wyde help auth-method-set* and *wyde auth-method-set* Commands Output Sample

View Authorization Methods

To show a list of all authorization methods in the system using the command line, you should use the *wyde* command line utility with the *auth-method-show* option. The syntax is as follows:

```
wyde auth-method-show
```

This command will output a list of all existed authorization methods on the system, similar to shown on Figure 67. As you can see, the *wyde auth-method-show* command shows the authorization methods that have been created in the system as well as their basic properties: authorization method name, adapter, parameters, and description.


```

root@ZILBER:~# wyde auth-method-show
Name          Adapter      Parameters      Description
-----
free          Free          Free            Without authorization
local         LocalDb       LocalDb         Authorization via local database
wydeldap      WYDELdap     localhost::test:dc=wydevoice,dc=com Authorization via WYDE LDAP - sample
wyderadius    WYDERadius   test@192.168.1.41:1812 Authorization via WYDE RADIUS server - sample
wyderadius2   WYDERadius   test2@192.168.1.42:1812 Authorization via WYDE RADIUS - sample 2
root@ZILBER:~#

```

Figure 67: *wyde auth-method-show* Command Output Sample

Sample of Authorization Adapters for LDAP and Radius

As it was previously told, you can write your own authorization adapters when it is necessary. Custom authorization adapters are routines written in Perl that perform calls authorization using specific protocols.

Each authorization adapter should have method *new* that performs class initialization, for instance access protocol initialization, database initialization, socket initialization, etc. In addition each authorization adapter should have such public methods as *get_confuser_by_accesscode*, *get_confuser_by_number*, *get_conference_attributes*, that are used for actual calls authorization in the conferences based on DNIS number, access code, conference number, etc.

Let's review the following scenario:

- you have Windows Active Directory Domain Controller, i.e. server computer (for example with Windows 2003 or Windows 2008) with Active Directory Domain Services installed;
- your Active Directory contains information about your conference accounts: conference number, DNIS number, access code and used roles, as well as your conferences definitions, as shown on Figure 68;

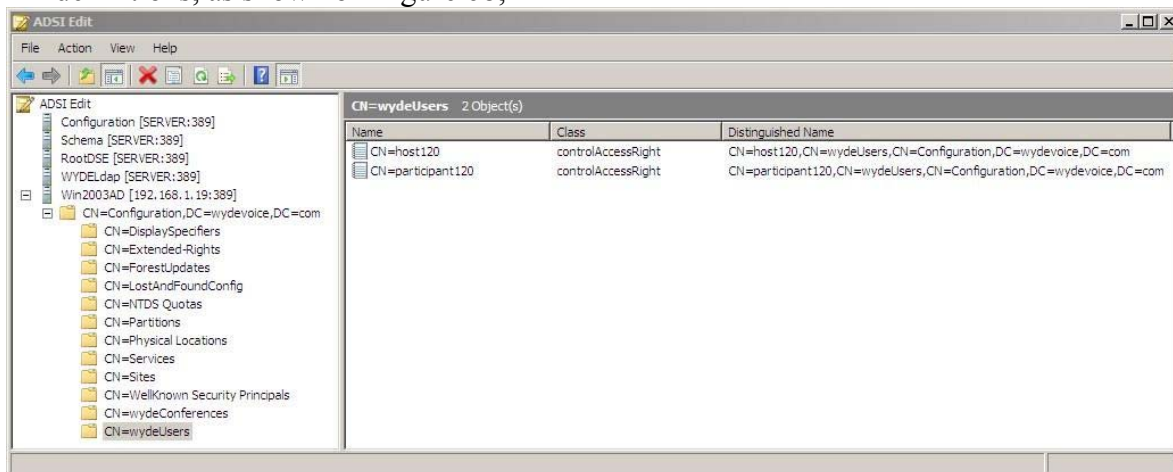


Figure 68: Active Directory Conference Accounts and Conference Numbers Data

- your domain address is *wydevoice.com*; the information is being stored under *Configuration* folder; to subfolders created: *confUser* for conference accounts information and *confNumber* for conferences information; the object class that is used: *controlAccessRight*;

- for this sample purposes we assume that the single conference with conference number *120* is defined, two conference accounts with DNIS number *12* are being created for this conference: for the *Host* role with access code *1210* and for the *Participant* role with access code *1211*; these data are being stored in Active Directory as shown in Table 9;

Table 9: Active Directory Conference Accounts Data

cn	conference accounts attributes	host120	participant120
displayName		confUser (12,host)	confUser (12,part)
adminDescription	accesscode	1210	1211
adminDisplayName	role	Host	Participant
description	did_number	12	12
uSNSource	conf_number	120	120

- we need to create and configure the authorization adapter that will read this conference accounts information from Windows Active Directory and perform conference authorization based on these data.

[Click here to see sample of the authorization adapter *WinLdap* source code that we developed to implement this request.](#)

Note that to work with Active Directory from your bridge computer the following RPM packages should be installed: *cyrus-sasl*, *krb5-libs*, *krb5-workstation*, *perl-Authen-Cyrus*, *perl-Authen-Krb5*, *perl-Authen-SASL*.

To use *Kerberos* you should configure it by editing */etc/krb5.conf* configuration files, in this file you should write down the information about you Active Directory Domain Controller, in our sample:

```
[realms]
  WYDEVOICE.COM = {
    kdc = 192.168.1.19:88
    admin_server = 192.168.1.19:749
    default_domain = wydevoice.com
  }
```

```
[domain_realm]
  .wydevoice.com = WYDEVOICE.COM
  wydevoice.com = WYDEVOICE.COM
```

In addition you should configure and use *kinit* program from *Kerberos* tool to perform authorization to Active Directory computer; it is used to obtain and cache Kerberos ticket-granting tickets; this program asks to enter your user name and password to Active Directory computer and bridge receives the ticket that is valid for 24 hours by default; you should configure *kinit* execution to have the valid ticket to your Active Directory computer if your authorization adapter uses these data.

In the authorization adapter code authentication and binding to your Active Directory computer is being made in *sub new* method:

```
my $sasl = Authen::SASL->new(mechanism => 'GSSAPI');
$self->{CLIENT} = new Net::LDAP($server->{host}, port => $server->{port},
                                onerror => 'die', debug => 0);
$self->{CLIENT}->bind(sasl => $sasl);
```

When user connects to the conference, the search within LDAP (Active Directory) data is being made based on DNIS number and access entered, for instance using the filter:

```
my $filter = "&(objectClass=controlAccessRight)(description=$did_number)
(adminDescription=$accesscode)";
```

and conference account data are being returned if the search was successful.

When design of *WinLdap.pm* file is completed you should copy this file into */usr/local/DNCA/lib/Auth/Adapter* folder and then you should restart *agiserver* service to actualize these changes, using the following command:

```
service agiserver restart
```

This command also should be run if you made any changes in your authorization adapter file.

Next you can add authorization adapter and authorization method using the following commands:

```
wyde auth-adapter-add name WinLdap
    description "authorization via Windows LDAP"
wyde auth-method-add name winldap adapter WinLdap
    description "Authorization via Windows LDAP - sample"
    parameters
        "192.168.1.19:::CN=wydeUsers,CN=Configuration,dc=wydevoic
        e,dc=com"
```

Note that after you add the authorization adapter you also should restart *agiserver* service using the command:

```
service agiserver restart
```

After that you should change *dnis_authORIZEMETHOD* (Authorize method) call flow attribute value for your DNIS 12 (as it is described in our scenario) and set it equal *winldap*.

As soon as this has been made all calls to this DNIS number will be authorized using the authorization adapter that we developed, i.e. the authorization will be made using Windows Active Directory data.

Let's review another scenario:

- assume that we have Windows PostgreSQL database *users* and its *Accounts* table contains information about account conferences, i.e. conference numbers, DNIS numbers, access codes, and user roles in the conferences, the structure of this table is the following:

```
CREATE TABLE "Accounts"
(
    "AccountID" serial NOT NULL,
```

```

"DNIS" text,
"Role" text,
"AccessCode" text,
"ConferenceNumber" text,
"CreateDate" timestamp without time zone DEFAULT now(),
CONSTRAINT "PrimaryKey_Accounts" PRIMARY KEY ("AccountID")
)
WITH (
    OIDS=FALSE
);

```

and the contents of this table is the following:

AccountID	DNIS	Role	AccessCode	ConferenceNumber
1	8665080012	Host	8001	880088
2	8665080012	Participant	8002	880088

In our sample PostgreSQL Windows computer IP address is *192.168.1.99*, database user name is *WydeAuthAdapter*, user password is *123*;

- assume that we should use the *Radius* server to access the data from this database; this *Radius* server could be installed on any computer but for the purpose of this sample we assume that it is installed on the same computer with your WYDE bridge;
- we need to perform conference authorization using the *Radius* server and configure the WYDE bridge and the *Radius* server to read the conference accounts information from the described database using the *Radius* authorization server; the standard *WYDERadius* authorization adapter should be used to implement this request.

To implement this scenario first you should install the *Radius* server (*freeradius*) and adapter to work with PostgreSQL (*freeradius-postgresql*) on your bridge computer using the following command:

```
yum install freeradius freeradius-postgresql
```

This command installs two RPM packages that are necessary to use authorization via Radius using PostgreSQL database.

Your WYDE bridge computer contains */usr/local/DNCA/lib/Auth/Radius* folder; this folder contains the files that would be necessary to implement this request and few samples regarding to the Radius authorization:

- *dictionary.wyde* – WYDE dictionary file, this file should be copied into */etc/raddb* folder;
- *wyde_sql.conf.sample*, *wyde_sql.conf.sample_fcc2*, *wyde_sql.conf.sample_fcc2_oracle* – the configuration files samples, that could be used to connect to different databases; let's use *wyde_sql.conf.sample* file as basis of our configuration file, rename it to *wyde_sql.conf* and copy it into */etc/raddb* folder;
- *radiusd.conf.sample* – the sample of the main Radius server main configuration file – you should rename it to *radiusd.conf* and copy it into */etc/raddb* folder.

Next we should update the files from */etc/raddb* folder:

- *dictionary* file should be changed – INCLUDE statement for *dictionary.wyde* file should be added to this file as follows:
\$INCLUDE dictionary.wyde

- *clients.conf* file should be changed to define the Radius clients; *localhost* (127.0.0.1) is enabled by default:

```
client 127.0.0.1 {
    secret = testing123
    ...
}
```

this configuration defines that the access to the Radius server could be made from this computer using the password *testing123*; because we have installed the Radius server on the same computer with our WYDE bridge and we are going to use it from the same computer, it is enough to have this configuration;

- you can use *radiusd.conf* main configuration file of the Radius server without additional changes if you copied it as described above, draw attention to the sections *modules*, *authorize*, *authenticate*:

```
# Module Configuration.
modules {
    # DEFAULT: crypt
    pap {
        encryption_scheme = crypt
    }
    $INCLUDE ${confdir}/wyde_sql.conf

    always fail {
        rcode = fail
    }
    always reject {
        rcode = reject
    }
    always ok {
        rcode = ok
        simulcount = 0
        mpp = no
    }
}

# Authorization.
authorize {
    wyde_confuser
}

# Authentication.
authenticate {
    Auth-Type PAP {
        pap
    }
}
```

- *wyde_sql.conf* configuration file should be changed to reflect the PostgreSQL server (IP address, user name and password, database name) and your specific data structure:

```
sql wyde_confuser {
    driver = "rlm_sql_postgresql"
    server = "192.168.1.99"
    login = "WydeAuthAdapter"
    password = "123"
    radius_db = "users"

    # Remove stale session if checkrad does not see a double login
    deletestalesessions = yes
    # Print all SQL statements when in debug mode (-x)
    sqltrace = yes
    sqltracefile = ${logdir}/sqltrace.sql
    # number of sql connections to make to server
    num_sql_socks = 5

    authorize_check_query = "SELECT 0 as id, \"AccessCode\" as UserName, \
        'User-Password' as Attribute, \"DNIS\" as Value, '==' as Op \
        FROM \"Accounts\" WHERE \"AccessCode\" = '%{User-Name}' ORDER BY id "
```

```

authorize_reply_query = "SELECT 0 as id, \"AccessCode\" as UserName, \
    'conf_number' as Attribute, \"ConferenceNumber\" as Value, '=' as Op \
FROM \"Accounts\" \
WHERE \"AccessCode\" = '%{User-Name}' \
UNION \
SELECT 1 as id, \"AccessCode\" as UserName, \
    'role' as Attribute, \"Role\" as Value, '=' as Op \
FROM \"Accounts\" \
WHERE \"AccessCode\" = '%{User-Name}' \
ORDER BY id "
}

```

In this configuration file *driver* determines which driver is used to connect to the database (in our sample “*rlm_sql_postgresql*” is used for PostgreSQL, for MySQL should be used “*rlm_sql_mysql*” driver), *server* determines the IP address of the server, *login/password* – credentials that should be used to access the data, *radius_db* – the name of the database;

When Radius server implements the requests this configuration file receives two variables: *%{User-Name}* variable is equal to the access code entered by the caller and *%{User-Password}* variable is equal to the DNIS number the caller called (for example if user called to the DNIS number 8665080012 and entered the access code 8001, *%{User-Name}* variable would be equal to 8001 and *%{User-Password}* variable would be equal to 8665080012);

Two queries should be defined in this configuration file:

- *authorize_check_query* – for the performed call if the access code is valid and authorization is successful this check-authorization query should return the single row with the following columns: id, UserName (access code used), Attribute (‘User-Password’ string), Value (DNIS number called), Op (‘==’ string):

id	UserName	Attribute	Value	Op
0	8001	User-Password	8665080012	==

- *authorize_reply_query* – for the performed call if the access code is valid and authorization is successful this query should return two rows with the same fields, but different data: the first row with information about the conference number (UserName equals to access code used, Attribute equals to ‘conf_number’ string, Value equals to 880088 in our case, Op equals to ‘=’ string) and the second row with information about the caller role in the conference (UserName equals to access code used, Attribute equals to ‘role’ string, Value equals to ‘Host’ string if access code equal to 8001 or ‘Participant’ string if access code equal to 8002, Op equals to ‘=’ string):

id	UserName	Attribute	Value	Op
0	8001	conf_number	880088	=
1	8001	role	Host	=

these data are being transferred to authorization adapter in the form “*the attribute equals the value*”, i.e. in our case *conf_number=880088* and *role=Host*.

As soon as you completed the Radius server configuration you should start its service using the command:

```
service radiusd start
```

[Click here to see sample of the authorization adapter WYDERadius source code.](#)

If you performed all steps as described above you do not need to make any changes in this adapter and the standard WYDERadius authorization adapter can be used for this Radius authorization.

Note that if you made any changes in your authorization adapter the following command should be run:

```
service agiserver restart
```

This *WYDERadius* adapter already exists in your WYDE bridge. You can see it using the command:

```
wyde auth-adapter-show
```

But you need to update authorization method *wyderadius* that is working with this authorization adapter using the command:

```
wyde auth-method-set name wyderadius
    parameters testing123@localhost
```

Here *testing123* – the password to your Radius server that you described in the *clients.conf* file in the parameter *secret*; *localhost* – denotes that your Radius server is installed on the same computer with your WYDE bridge.

After that you should change *dnis_authmethod* (Authorize method) call flow attribute value either on call flow level or on DNIS level and set it equal *wyderadius*.

As soon as this has been made all calls to the updated DNIS or call flow will be authorized using the *WYDERadius* authorization adapter, i.e. the authorization will be made using the Radius server according to your *users* database *Accounts* table.

Let's review another similar scenario:

- assume that in addition to previously described data the same Windows PostgreSQL *users* database contains *Conferences* table with call flow attributes defined for the specific conferences, i.e. this table contains conference numbers, call flow attributes names and values, the structure of this table is the following:

```
CREATE TABLE "Conferences"
(
    "ConferenceID" serial NOT NULL,
    "ConferenceNumber" text,
    "CallFlowAttributeName" text,
    "CallFlowAttributeValue" text,
    "CreateDate" timestamp without time zone DEFAULT now(),
    CONSTRAINT "PrimaryKey_Conferences" PRIMARY KEY ("ConferenceID")
)
WITH (
    OIDS=FALSE
);
```

and the contents of this table is the following:

ConferenceID	ConferenceNumber	CallFlowAttributeName	CallFlowAttributeValue
1	880088	conference_entrytones	off
2	880088	conference_exittones	off
3	880088	call_instructions_dtmf	h

To use these new data all previously made settings stay the same, you should only update *wyde_sql.conf* configuration file from */etc/raddb* folder:

- this file should be changed to use your *Conferences* table data as call flow attributes for the specific conferences; you should add to the end of *sql wyde_confuser* settings code the definition of *authorize_group_reply_query* parameter:

```
sql wyde_confuser {
    .....
    authorize_group_reply_query = "SELECT \"ConferenceID\" as id, \
                                  \"Conferences\".\"ConferenceNumber\" as GroupName, \
                                  \"CallFlowAttributeName\" as Attribute, \
                                  \"CallFlowAttributeValue\" as Value, '=' as Op \
    FROM \"Conferences\" INNER JOIN \"Accounts\" ON \
                                  \"Conferences\".\"ConferenceNumber\" = \
                                  \"Accounts\".\"ConferenceNumber\" \
    WHERE \"AccessCode\" = '{User-Name}' \
    ORDER BY id "
```

As you can see the third query should be defined in this configuration file:

- *authorize_group_reply_query* – for the performed call if the access code is valid and authorization is successful this query should return the rows for any call flow attributes specific for the conference with the following columns: id, GroupName (the conference number, i.e. 880088 in our case), Attribute (call flow attribute name), Value (call flow attribute value), Op ('=' string):

id	GroupName	Attribute	Value	Op
1	880088	conference_entrytones	off	=
2	880088	conference_exittones	off	=
3	880088	call_instructions_dtmf	h	=

As soon as you changed *wyde_sql.conf* configuration file you should restart Radius service using the command:

```
service radiusd restart
```

Once this has been done not only all calls to this DNIS/call flow will be authorized using the *WYDERadius* authorization adapter, i.e. the authorization will be made using the Radius server according to your *users* database *Accounts* table, but also the conference call flow attributes will be taken from this database *Conferences* table.

Billing

To implement billing the WYDE bridge software can store and transmit CDR data that could be used for billing. Note that the WYDE bridge software is not responsible for financial billing; it neither tracks credit cards nor sends invoices to the clients. It only provides CDR data and it is up to you how to use them in your financial billing.

The WYDE bridge software provides storing CDR information in the following locations:

- in text file */usr/local/DNCA/log/CDR.log*, this file contains information for today's CDRs only, in addition the history is being saved for the last 10 days in the same folder in the files from *CDR.log.1* (yesterday) till *CDR.log.10* (10 days ago); this file is comma-separated file, the information that is being stored in this file is listed in Table 10;

- in local *dnca_calls* database (the database name is defined using *billing_localdb_name* configuration parameter), by default the database contains CDR data for the last 180 days (this period can be redefined using *billing_localdb_storing_period* configuration parameter); if you are going to use these billing data, they should be transferred to your external database within this time period;
the basic tables that store CDR information are *conferencedr* and *calls* tables, their fields with descriptions are listed in Table 11.

Table 10: CDR.log File Data Structure**Field Name and Description**

Bridge name
 Call session identifier
 Conference number
 Date when the call was created
 Connection type, i.e. call direction (*In* for inbound calls, *Out* for outbound calls)
 Calling number
 Called number
 Time when the call was created
 Time when the call was dropped
 Duration of the call in seconds
 Who disconnected the call (for instance, *USER*, *BRIDGE*)
 The reason why the call was disconnected (for instance *Normal*, *Error*)
 Call flow name
 Access code used
 Role in the conference (*Host*, *Participant*, *Listener*)
 Custom call type (for instance, *CONTROLLED*, *PSTN*, *RECORDING*, *VoIP*)
 Conference identifier
 Conference flag – 2 value of this flag determines that this call is the last call in the conference and the conference was completed when this call ended; otherwise this flag is empty

Table 11: Local *dnca_calls* Database *conferencedr* and *calls* Tables Data Structure

Table	Field	Description
conferencedr	conferenceid	Conference identifier
conferencedr	number	Conference number
conferencedr	created	Date and time when the conference was created, i.e. the first caller arrived
conferencedr	closed	Date and time when the conference was closed (ended)
conferencedr	duration	Conference duration in seconds, number of seconds which have elapsed since the conference was created till the time when it was terminated
conferencedr	confduration	Total duration in seconds of all calls that were joined to the conference, i.e. sum of all conference calls durations

conferencedr	recduration	Conference recording duration in seconds
conferencedr	totalcount	Total number of conference calls
conferencedr	reccount	Number of the conference recording
conferencedr	modcount	Number of hosts (moderators) that were joined to the conference
conferencedr	cdrid	Internal serial identifier of <i>conferencedr</i> table
calls	id	Internal serial identifier of <i>calls</i> table
calls	bridge	Bridge name
calls	node	Node name
calls	call_id	Call identifier
calls	connection_type	Connection type, i.e. call direction (<i>In</i> for inbound calls, <i>Out</i> for outbound calls)
calls	calling_number	Incoming calling number, i.e. the number from which called the caller
calls	called_number	Called number, i.e. the number to which the caller had called
calls	addr_to	Full address TO, i.e. full qualified callee's address
calls	addr_from	Full address FROM, i.e. full qualified caller's address
calls	call_created	Date and time when the call was created (started)
calls	call_dropped	Date and time when the call was dropped (ended)
calls	duration	Duration of the call in seconds
calls	disconnect_who	Who disconnected the call (for instance, <i>USER</i> , <i>BRIDGE</i>)
calls	disconnect_cause	Standard Q.931 (ISDN) Disconnect Cause Codes; the cause codes list can be used to decode the disconnect reasons in ISDN messages (PBX or PSTN interfaces); Q.931 messages used to communicate over IP, Tenor CDR records and Tenor/Radius messages (for instance, <i>16</i> – Normal Call Clearing, <i>18</i> – No User Responding, <i>34</i> – No Circuit/Channel Available, <i>38</i> – Network Out-of-Order, <i>127</i> – Interworking, Unspecified) ¹
calls	disconnect_reason	The reason why the call was disconnected (for instance <i>Normal</i> , <i>Dropped by host</i> , <i>Incorrect access code</i> , <i>Moved to other conference</i> , <i>NOANSWER</i> , <i>CONGESTION</i> , etc.)
calls	conf_id	Conference identifier
calls	access_code	Access code used
calls	callflow	Call flow name
calls	conf_number	Conference number

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

- http://www.quintum.com/support/xplatform/network/Q931_Disconnect_Cause_Code_List.pdf
- http://www.quintum.com/support/xplatform/ivr_acct/webhelp/Disconnect_Cause_Codes.htm

calls	conf_joined	Date and time when the call was joined to the conference
calls	conf_rejected	Date and time when the call was disconnected from the conference
calls	conf_duration	Total duration in seconds of how long the call was in joined to the conference
calls	conf_mode	Role in the conference (<i>Host, Participant, Listener</i>)
calls	custom_name	Custom caller name either set from the web or IVR (PIN)
calls	subscriber_name	Name of the subscriber assigned by this call
calls	custom_call_type	Custom call type (for instance, <i>CONTROLLED, PSTN, RECORDING, VoIP</i>)
calls	audio_key	Audio key assigned to this call
calls	job_code	Active billing (business) code
calls	conf_flag	Conference flag – 2 value of this flag determines that this call is the last call in the conference and the conference was completed when this call ended; otherwise this flag is empty

The following methods could be used to receive CDR information by external system:

- 1) CDR information could be periodically taken from the billing database or *CDR.log* file, you can write your own routine that in given time interval will periodically take new CDR data from the WYDE bridge billing database and place them into your own database that you use for your billing procedures;
- 2) CDR information could be placed to your FTP using the special script; this script usually are being run daily, it takes CDR records for the last day, creates the file with these data in the requested format and places this file to the specified FTP;
 - please contact WYDE Voice technical support if you need to place your CDR information to your FTP;
- 3) The billing adapter could be written that will store CDR information once the calls are completed; it will operate according to your needs, i.e. using your specific rules; using this approach you should write your own billing adapter to receive CDR data into your database or other location.

In terms of WYDE bridge software the *Billing Adapter* is the component (function) responsible for storing billing, i.e. CDR information. Billing adapter is listening information on a special UDP port; on this port *MF* sending information about completed calls; billing adapter receives this information, transforms it into required format and stores it in required data carrier. More formally, "billing" is the information about completed conferences and calls also called CDR.

The following billing adapters are included and supported by standard WYDE bridge software installation:

- *file* – the adapter that saves CDR information into */usr/local/DNCA/log/CDR.log* file;
- *localdb* – the adapter that saves CDR information into *dnca_calls* local database.

Note that both these billing adapters are standard adapters and automatically supported by the WYDE bridge software. They should not be added in the list of billing adapters maintained by *billing-adapter-**** commands; these commands are managing custom billing adapters only.

Custom billing adapters are routines, i.e. drivers, written in Perl that perform saving of CDR information using specific protocols. These drivers are placed in the */usr/local/DNCA/lib/Billing/Adapter* folder that should contain the files *<Adapter Driver>.pm*; for example if you have custom billing adapter *tfcc*, with the driver *TFCC* this folder on your bridge should contain the file *TFCC.pm* – the billing adapter driver that can be used.

If you would like to store CDR information in your own database you can create your custom billing adapter that will be responsible for saving CDR data as it is required for your organization. The input format for billing adapters, i.e. the information that comes from *MF* to the billing adapter UDP port is shown in Table 12.

Table 12: Input Data Format for Billing Adapters

Field	Description
subscriber_name	First and last names of the subscriber
conf_flag	Conference flag – 2 value of this flag determines that this call is the last call in the conference and the conference was completed when this call ended; otherwise this flag is empty
calling_number	Incoming calling number, i.e. the number from which called the caller
custom_call_type	Custom call type (for instance, <i>CONTROLLED</i> , <i>PSTN</i> , <i>RECORDING</i> , <i>VoIP</i>)
disconnect_reason	The reason why the call was disconnected (for instance <i>Normal</i> , <i>Dropped by host</i> , <i>Incorrect access code</i> , <i>Moved to other conference</i> , <i>NOANSWER</i> , <i>CONGESTION</i> , etc.)
callflow	Call flow name
node	Node name
conf_number	Conference number
call_dropped	Date and time when the call was dropped (ended)
call_created	Date and time when the call was created (started)
job_code	Active billing (business) code
called_number	Called number, i.e. the number to which the caller had called
custom_name	Custom caller name either set from the web or IVR (PIN)
audio_key	Audio key assigned to this call
bridge	Bridge name
call_id	Call identifier
call_duration	Duration of the call in seconds
addr_to	Full address TO, i.e. full qualified callee's address
access_code	Access code used
conf_id	Conference identifier
disconnect_cause	Standard Q.931 (ISDN) Disconnect Cause Codes; see detail description for this field above in Table 11

connection_type	Connection type, i.e. call direction (<i>In</i> for inbound calls, <i>Out</i> for outbound calls)
role	Role in the conference (<i>Host</i> , <i>Participant</i> , <i>Listener</i>)
disconnect_who	Who disconnected the call (for instance, <i>USER</i> , <i>BRIDGE</i>)
addr_from	Full address FROM, i.e. full qualified caller's address

For the conference billing information storing are being used billing rules. Billing rules determines the specific billing adapters that are used to save CDR information, this includes comma-separated billing adapters that should be used in the specific billing rule, i.e.

`{{file}[.][localdb][.][adapter:<name>]}`:

- *file* – denotes that CDR information should be stored into `/usr/local/DNCA/log/CDR.log` file;
- *localdb* – denotes that CDR information should be stored into *dnca_calls* local database;
- *adapter:<name>* – denotes custom billing adapter name that should be used to store CDR information.

The billing rule name should be selected in `dnis_billingrule` (Billing rule) call flow attribute value either on call flow or on DNIS level.

Add a Billing Adapter

Before you add new billing adapter, you should create the `<Adapter Driver>.pm` driver file in the `/usr/local/DNCA/lib/Billing/Adapter` folder for this adapter driver as it was described above.

To add new billing adapter registration using the command line interface you should use the *wyde* command line utility with the *billing-adapter-add* option. The syntax is as follows:

```
wyde billing-adapter-add <arguments>
```

Each of the arguments is followed by a space and a value. In *billing-adapter-add* you can specify the following arguments:

- `name <value>` – The name of the billing adapter that should be added. This name should be unique, i.e. there should no be any other billing adapter with the same name on the bridge.
- `description <value>` – The optional description of the billing adapter that should be added.
- `driver <value>` – The driver name for the billing adapter that should be added. The file `<Adapter Driver Name>.pm` should exist in the `/usr/local/DNCA/lib/Billing/Adapter` folder.
- `bindaddr <value>` – The bind address and port for the billing adapter that should be added, the adapter is listening information on this address/port.
- `parameters <value>` – The list of parameters for the billing adapter that should be added.

Arguments `name`, `driver`, `bindaddr` and `parameters` are required. The arguments can be transferred to this command in any order.

Let's assume that we have created the file *TFCC.pm* in the folder */usr/local/DNCA/lib/Billing/Adapter* for new billing adapter *tfcc*, bind address is *127.0.0.1:5901*, parameters that should be used are *192.168.1.45:9000*. To add this billing adapter to the bridge you should use the command:

```
wyde billing-adapter-add name tfcc driver TFCC
    description "Billing using TFCC - sample"
    bindaddr 127.0.0.1:5901 parameters 192.168.1.45:9000
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *billing-adapter-add* command output and the help on this command is shown on Figure 69.

```

root@ZILBER:~
[root@ZILBER ~]# wyde help billing-adapter-add
Syntax:
    wyde billing-adapter-add arguments

Arguments:
    * name <value>           - Adapter name.
    * description <value>    - Adapter description.
    * driver <value>         - Driver name.
    * bindaddr <value>       - Bind address.
    * parameters <value>    - Parameters list.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde billing-adapter-add name tfcc driver TFCC description "Billing using TFCC - sample" bindaddr 127.0.0.1:5901 parameters 192.168.1.45:9000
[root@ZILBER ~]#

```

Figure 69: *wyde help billing-adapter-add* and *wyde billing-adapter-add* Commands Output Sample

Delete a Billing Adapter

To delete a billing adapter using the *wyde* command line utility you should use *billing-adapter-del* option. The syntax is as follows:

```
wyde billing-adapter-del name <billing adapter name>
```

where

- *<billing adapter name>* – the name of the billing adapter you wish to delete.
- Note that you can delete billing adapters that are not in use only, i.e. there should no be any billing rules that refer to this billing adapter. If the billing adapter is used by any billing rule you will receive the error message: “*<billing adapter name>: Billing adapter is in use and can not be removed.*” and the deletion will be cancelled.

For example to delete billing adapter *tfcc* (created in previous sample) you should run the command:

```
wyde billing-adapter-del name tfcc
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Modify a Billing Adapter

To modify billing adapter properties, such as description, driver, bind address and parameters, using the command line interface you should use the *wyde* command line utility with the *billing-adapter-set* option. The syntax is as follows:

```
wyde billing-adapter-set <arguments>
```

Each of the arguments is followed by a space and a value. In *billing-adapter-set* you can specify the following arguments:

- `name <value>` – The name of the billing adapter that should be changed.
- `description <value>` – New description of the billing adapter that should be set.
- `driver <value>` – New driver name for the billing adapter that should be set.
- `bindaddr <value>` – New bind address and port for the billing adapter that should be set, the adapter is listening information on this address/port.
- `parameters <value>` – New list of parameters for the billing adapter that should be set.

The argument name is required; you should specify other arguments only if you would like to change them. The arguments can be transferred to this command in any order.

For example if you would like to change *tfcc* billing adapter and set its parameters equal to “192.168.1.46:9000”, you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde billing-adapter-set name tfcc
      parameters 192.168.1.46:9000
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *billing-adapter-set* command output and the help on this command is shown on Figure 70.

```

root@ZILBER:~
[root@ZILBER ~]# wyde help billing-adapter-set
Syntax:
  wyde billing-adapter-set arguments

Arguments:
  * name <value>          - Rule name.
  description <value>     - Adapter description.
  driver <value>          - Driver name.
  bindaddr <value>        - Bind address.
  parameters <value>      - Parameters list.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde billing-adapter-set name tfcc parameters 192.168.1.46:9000
[root@ZILBER ~]#

```

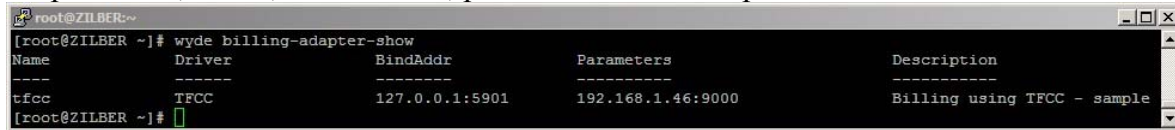
Figure 70: *wyde help billing-adapter-set* and *wyde billing-adapter-set* Commands Output Sample

View Billing Adapters

To show a list of all billing adapters in the system using the command line, you should use the *wyde* command line utility with the *billing-adapter-show* option. The syntax is as follows:

```
wyde billing-adapter-show
```

This command will output a list of the all existed billing adapters on the system, similar to shown on Figure 71. As you can see, the *wyde billing-adapter-show* command shows the billing adapters that have been created in the system as well as their basic properties: billing adapter name, driver, bind address, parameters and description.



Name	Driver	BindAddr	Parameters	Description
tfcc	TFCC	127.0.0.1:5901	192.168.1.46:9000	Billing using TFCC - sample

Figure 71: *wyde billing-adapter-show* Command Output Sample

Add a Billing Rule

To create new billing rule for the billing adapter using the command line interface you should use the *wyde* command line utility with the *billing-rule-add* option. The syntax is as follows:

```
wyde billing-rule-add <arguments>
```

Each of the arguments is followed by a space and a value. In *billing-rule-add* you can specify the following arguments:

- *name* <value> – The name of the billing rule that should be added.
- *description* <value> – The description of the billing rule that should be added.
- *rule* <value> – The billing rule content that should be added, comma-separated values: { [file] [,] [localdb] [,] [adapter:<name>] } :
 - *file* – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file;
 - *localdb* – denotes that CDR information should be stored into *dnca_calls* local database;
 - *adapter:<name>* – denotes custom billing adapter name that should be used to store CDR information.

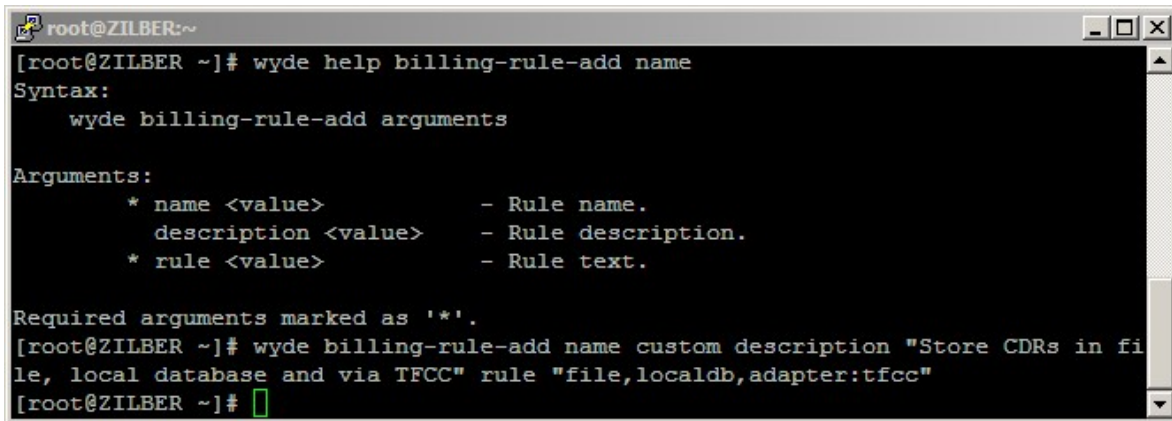
Arguments *name* and *rule* are required. The arguments can be transferred to this command in any order.

For example if you would like to create the billing rule *custom* that defines that CDR records should be stored into file, local *dnca_calls* database and using *tfcc* billing adapter with description “*Store CDRs in file, local database and via TFCC*” you should run the following command (new billing rule properties are shown in *italic*):

```
wyde billing-rule-add name custom
  description "Store CDRs in file, local database and via TFCC"
  rule "file,localdb,adapter:tfcc"
```

Note that to set the description that contains spaces and parameters you should use double quotes (").

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *billing-rule-add* command output and the help on this command is shown on Figure 72.



```

root@ZILBER:~
[root@ZILBER ~]# wyde help billing-rule-add name
Syntax:
    wyde billing-rule-add arguments

Arguments:
    * name <value>          - Rule name.
    description <value>    - Rule description.
    * rule <value>         - Rule text.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde billing-rule-add name custom description "Store CDRs in fi
le, local database and via TFCC" rule "file,localdb,adapter:tfcc"
[root@ZILBER ~]#

```

Figure 72: *wyde help billing-rule-add* and *wyde billing-rule-add* Commands Output Sample

Delete a Billing Rule

If you wish to delete the specific billing rule, you can use the *wyde* command line utility with *billing-rule-del* option. The syntax is as follows:

```
wyde billing-rule-del name <billing rule name>
```

where

- *<billing rule name>* – The name of the billing rule that should be deleted. This argument is required.

Note that you can delete only billing rules that are not in use. If the rule is used by any call flow and/or DNIS you will receive the error: “*<billing rule name>: Billing rule is in use and can not be removed.*” and the deletion will be cancelled.

For example to delete the billing rule *custom* you should run the command:

```
wyde billing-rule-del name custom
```

If deletion is successful, you will be returned to the command line with no additional prompts.

Modify a Billing Rule

To modify billing rule properties, such as description and rule content, using the command line interface you should use the *wyde* command line utility with the *billing-rule-set* option. The syntax is as follows:

```
wyde billing-rule-set <arguments>
```

Each of the arguments is followed by a space and a value. In *billing-rule-set* you can specify the following arguments:

- *name <value>* – The name of the billing rule that should be updated.
- *description <value>* – New description of the billing rule that should be set.
- *rule <value>* – New billing rule content that should be set, comma-separated values: { [file] [,] [localdb] [,] [adapter:<name>] } :
 - *file* – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file;

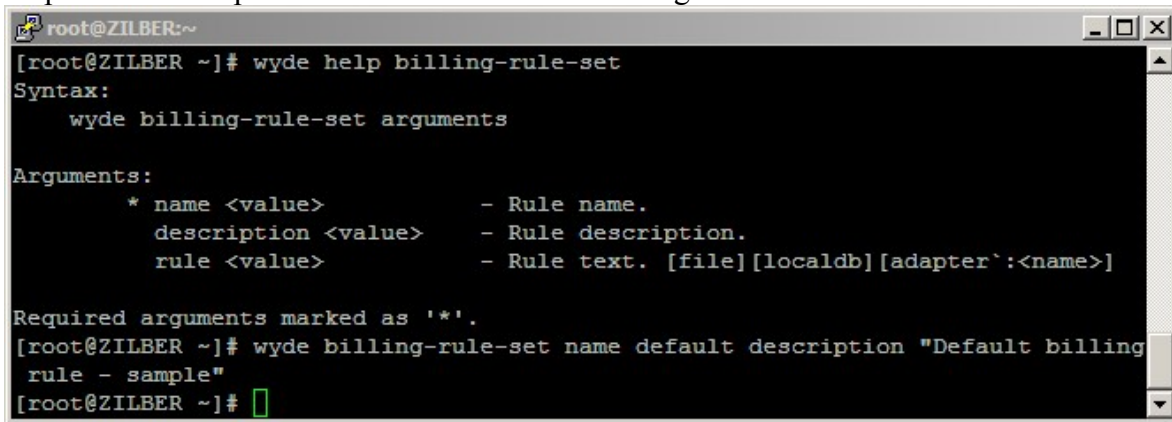
- o `localdb` – denotes that CDR information should be stored into *dnca_calls* local database;
- o `adapter:<name>` – denotes custom billing adapter name that should be used to store CDR information.

The argument `name` is required; you should specify arguments `description` and `rule` only if you would like to change them. The arguments can be transferred to this command in any order.

For example if you would like to change *default* billing rule and set its description equal to “*Default billing rule - sample*”, you should run the following command (the transferred command arguments are shown in *italic*):

```
wyde billing-rule-set name default
      description "Default billing rule - sample"
```

If the command is successful, the system will not return any errors or messages; it will just return you back to the command prompt (#). The sample of the *billing-rule-set* command output and the help on this command is shown on Figure 73.



```
root@ZILBER:~
[root@ZILBER ~]# wyde help billing-rule-set
Syntax:
  wyde billing-rule-set arguments

Arguments:
  * name <value>          - Rule name.
  description <value>     - Rule description.
  rule <value>            - Rule text. [file][localdb][adapter`:<name>]

Required arguments marked as '*'.
[root@ZILBER ~]# wyde billing-rule-set name default description "Default billing
rule - sample"
[root@ZILBER ~]#
```

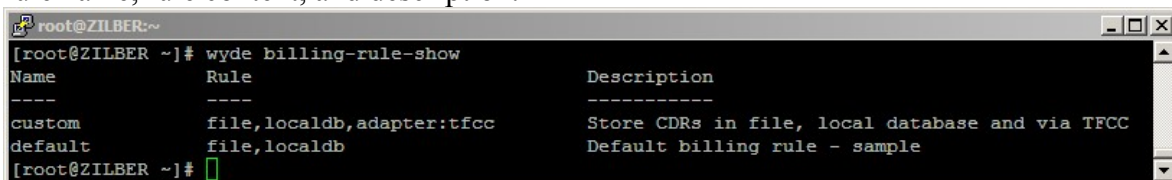
Figure 73: *wyde help billing-rule-set* and *wyde billing-rule-set* Commands Output Sample

View Billing Rules

To show a list of all billing rules in the system using the command line, you should use the *wyde* command line utility with the *billing-rule-show* option. The syntax is as follows:

```
wyde billing-rule-show
```

This command will output a list of the all existed billing rules on the system, similar to shown on Figure 74. As you can see, the *wyde billing-rule-show* command shows the billing rules that have been created in the system as well as their basic properties: billing rule name, rule content, and description.



```
root@ZILBER:~
[root@ZILBER ~]# wyde billing-rule-show
Name      Rule      Description
-----
custom    file,localdb,adapter:tfcc  Store CDRs in file, local database and via TFCC
default   file,localdb  Default billing rule - sample
[root@ZILBER ~]#
```

Figure 74: *wyde billing-rule-show* Command Output Sample

Samples of Billing Adapters

As it was previously told, you can write your own billing adapters when it is necessary. Custom billing adapters are routines, i.e. drivers, written in Perl that perform saving of CDR information using specific protocols.

Each billing adapter should have two methods: *new* – that performs class initialization, for instance billing receiver initialization, database initialization, socket initialization, etc., and *run* – that is being called when calls are completed and where you can implement data saving that is necessary for your organization, for example saving information into your database, writing information to socket, etc.

Let's review the following scenario:

- we need to create and configure the billing adapter that will write all CDR information (CDR fields and their values) into Windows PostgreSQL *dnca_calls* database *CDRs* table with the following structure:

```
CREATE TABLE "CDRs"
(
    "CdrID" serial NOT NULL,
    "CdrDATA" text,
    "CreateDate" timestamp without time zone DEFAULT now(),
    CONSTRAINT "PrimaryKey" PRIMARY KEY ("CdrID")
)
WITH (
    OIDS=FALSE
);
```

In our sample PostgreSQL Windows computer IP address is *192.168.1.99*, database user name is *WydeBillingAdapter*, user password is *123*.

[Click here to see sample of the billing adapter *WINPGSQL* source code that we developed to implement this request.](#)

Database specific connection operator is defined in *sub new* method:

```
$self->{db} = DBI->connect ("dbi:Pg:dbname=$database;host=$host", $user,
    $password) || proc_error("Connect: ".DBI::errstr);
```

This operator defines database format, server address, database name, user name and password.

Specific data saving mechanism is implemented in *sub run* method; it includes data formatting according to your needs and insert statement design and execution (see sample in this guide appendix).

When design of *WINPGSQL.pm* file is completed you should copy this file into */usr/local/DNCA/lib/Billing/Adapter* folder and then you should restart *agiserver* service to actualize these changes, using the following command:

```
service agiserver restart
```

This command also should be run if you made any changes in your billing adapter file.

Next you can add billing adapter and billing rule using the following commands:

```
wyde billing-adapter-add name winpgsql driver WINPGSQL
description "Billing to PostgreSQL on Windows"
bindaddr 127.0.0.1:5902 parameters 192.168.1.99
wyde billing-rule-add name winpg description "Store CDRs in
file, local database and in Windows PostgreSQL database"
rule "file,localdb,adapter:winpgsql"
```

Note that the bind address in your billing adapter should be unique, i.e. there should no be any other billing adapters with the same binding address. The billing rule *winpg* defines that the CDR data should be stored into CDR file, local database and using *winpgsql* adapter into Windows PostgreSQL database.

Also note that after you add the billing adapter you also should restart *agiserver* service using the command:

```
service agiserver restart
```

After that you should change *dnis_billingrule* (Billing rule) call flow attribute either on call flow level or on DNIS level and set it equal *winpg*.

As soon as this has been made when the calls are ended their CDR information will be stored not only in traditional locations (CDR file and local database), but also in your PostgreSQL database. Here we provide few samples of your stored data:

```
"subscriber_name=,conf_flag=0,calling_number=,custom_call_type=PSTN,disconnect_reason=Normal
,callflow=SPECTEL,node=AST1,conf_number=667788,call_dropped=2010-06-07
19:12:11,call_created=2010-06-07
19:11:51,job_code=,called_number=12,custom_name='unknown',audio_key=0,bridge=WYDE5,call_id=16777996,call_duration=20,addr_to="12_6602"
<sip:12_6602@192.168.1.5>,access_code=6602,conf_id=430,disconnect_cause=16,connection_type=In,role=Participant,disconnect_who=USER,addr_from="unknown" <sip:192.168.1.99>"
"subscriber_name=,conf_flag=2,calling_number=admin,custom_call_type=PSTN,disconnect_reason=Normal,callflow=SPECTEL,node=AST1,conf_number=667788,call_dropped=2010-06-07
19:12:16,call_created=2010-06-07
19:11:23,job_code=,called_number=12,custom_name='100',audio_key=0,bridge=WYDE5,call_id=16777995,call_duration=53,addr_to="12_6601"
<sip:12_6601@192.168.1.5>,access_code=6601,conf_id=430,disconnect_cause=16,connection_type=In,role=Host,disconnect_who=USER,addr_from="100"<sip:admin@192.168.1.5>"
```

Let's assume that instead of Windows PostgreSQL database the same data (CDR fields and their values) should be stored into Windows Microsoft SQL database:

- database name is *dnca_calls*, table name is *CDRs*, the table has the following structure:

```
CREATE TABLE [dbo].[CDRs] (
    [CdrID] [int] IDENTITY(1,1) NOT NULL,
    [CdrData] [text] NULL,
    [CreateDate] [datetime] NOT NULL,
    CONSTRAINT [PK_CDRs] PRIMARY KEY CLUSTERED
    (
        [CdrID] ASC
    )
)
```

In our sample Microsoft SQL Server Windows computer IP address is *192.168.1.9*, database user name is *WydeBillingAdapter*, user password is *123*.

Note that to access to Microsoft SQL databases from Linux CentOS computer you should install additional packages on your WYDE bridge computer:

- *FreeTDS* package (<http://www.freetds.org/>) should be installed:

```
./configure --prefix=/opt/freetds
make
make install
```
- *DBD::Sybase* package (<http://search.cpan.org/~mewp/DBD-Sybase-1.10/>) should be installed:

```
export SYBASE=/opt/freetds
perl Makefile.PL
make
make install
```

In addition you should edit */opt/freetds/etc/freetds.conf* configuration file and write down the information about your SQL server:

```
[MSSQL]
    host = 192.168.1.9
    port = 1433
    tds version = 8.0
```

Here *MSSQL* is named instance of your Microsoft SQL server computer with given IP address, port and version. You can use this name in your code to access to your SQL server.

[Click here to see sample of the billing adapter MSSQL source code that we developed to implement this request.](#)

Database specific connection operator is defined in *sub new* method:

```
$self->{db} = DBI->connect("dbi:Sybase:server=$host:database=$database",
    $user, $password) || die("Connect: ".DBI::errstr."\n");
```

This operator defines database format (*Sybase* keyword should be used for Microsoft SQL Server), server address, database name, user name and password.

Specific data saving mechanism is implemented in *sub run* method; it includes data formatting according to your needs and insert statement design and execution (see sample in this guide appendix). This method is almost the same as it was used for previous billing adapter written for PostgreSQL; the only difference could be in SQL statement format.

When design of *MSSQL.pm* file is completed you should copy this file into */usr/local/DNCA/lib/Billing/Adapter* folder and then you should restart *agiserver* service to actualize these changes as it was previously described.

Next you can add billing adapter and billing rule using the following commands:

```
wyde billing-adapter-add name mssql driver MSSQL
    description "Billing to Microsoft SQL on Windows"
    bindaddr 127.0.0.1:5903 parameters MSSQL
wyde billing-rule-add name winms description "Store CDRs in
    file, local database and in Windows Microsoft SQL
    database" rule "file,localdb,adapter:mssql"
```

Note that the bind address for this adapter is different from the previous adapter because it should be unique. The billing rule *winms* defines that the CDR data should be stored into CDR file, local database and using *mssql* adapter into Windows Microsoft SQL database.

Also note that after you add the billing adapter you also should restart *agiserver* service using the command:

```
service agiserver restart
```

After that you should change *dnis_billingrule* (Billing rule) call flow attribute either on call flow level or on DNIS level and set it equal *winms*.

As soon as this has been made when the calls are ended their CDR information will be stored not only in traditional locations (CDR file and local database), but also in your Microsoft SQL database. These stored data will be exactly the same with the data saved into PostgreSQL database.

Database Administration

The WYDE bridge administration also can be made using the *PostgreSQL* interactive terminal (i.e. database console). This is standard external tool to work with *PostgreSQL* databases.

The following databases are being used by the WYDE bridge software:

- *dnca* – the main database;
- *dnca_calls* – the billing database.

To perform initialization of the databases, i.e. database creation, necessary tables creation and population using the command line, you should use the *wyde* command line utility with the *db-init* option. The syntax is as follows:

```
wyde db-init [database {Main|Billing}]
```

where

- *database {Main|Billing}* – Denotes what kind of database should be initialized:
 - *Main* is being used for *dnca* database.
 - *Billing* is being used for *dnca_calls* database.

When the *database* argument is omitted the main (*dnca*) database is being initialized. Usually you do not need to run this command. This command is being run automatically during new system installation.

To apply the latest patches (changes) to the existing databases using the command line, you should use the *wyde* command line utility with the *db-patch* option. The syntax is as follows:

```
wyde db-patch
```

Usually you also do not need to run this command. This command is being run automatically during the system upgrade installation.

To connect to the WYDE *dnca* (main) database, i.e. to open the PostgreSQL interactive terminal (*psql*) for this database using the command line, you should use the *wyde* command line utility with the *db* option. The syntax is as follows:

```
wyde db
```

To connect to the WYDE *dnca_calls* (billing) database, i.e. to open the PostgreSQL interactive terminal (*psql*) for this database using the command line, you should use the *wyde* command line utility with the *db-bill* option. The syntax is as follows:

```
wyde db-bil
```

Chapter 3: Command Reference

***wyde* Command Reference**

All *wyde* commands have the following format:

wyde command [arguments]

This chapter describes all available *wyde* commands with their arguments. For any of the commands the required arguments marked with asterisk (*).

ast-status (Show WYDE *asterisk* Status)

Syntax:

wyde ast-status

auth-adapter-add (Add *auth* Adapter)

Syntax:

wyde auth-adapter-add arguments

Arguments:

name <value> – The name of the authorization adapter that should be added (*);

description <value> – The description of the authorization adapter that should be added.

auth-adapter-del (Delete *auth* Adapter)

Syntax:

wyde auth-adapter-del arguments

Arguments:

name <value> – The name of the authorization adapter that should be deleted (*).

auth-adapter-show (Show *auth* Adapters)

Syntax:

wyde auth-adapter-show

auth-method-add (Add *auth* Method)

Syntax:

wyde auth-method-add arguments

Arguments:

name <value> – The name of the authorization method that should be added (*);

description <value> – The description of the authorization method that should be added;

adapter <value> – The authorization adapter name for the authorization method that should be added (*);

parameters <value> – The list of parameters for the authorization method that should be added.

auth-method-del (Delete *auth* Method)*Syntax:*`wyde auth-method-del arguments`*Arguments:*`name <value>` – The name of the authorization method that should be deleted (*).**auth-method-set (Set *auth* Method)***Syntax:*`wyde auth-method-set arguments`*Arguments:*`name <value>` – The name of the authorization method that should be changed (*);`description <value>` – New description of the authorization method that should be set;`parameters <value>` – New list of parameters for the authorization method that should be set.**auth-method-show (Show *auth* Methods)***Syntax:*`wyde auth-method-show`**billing-adapter-add (Add Billing Adapter)***Syntax:*`wyde billing-adapter-add arguments`*Arguments:*`name <value>` – The name of the billing adapter that should be added (*);`description <value>` – The description of the billing adapter that should be added;`driver <value>` – The driver name for the billing adapter that should be added (*);`bindaddr <value>` – The bind address for the billing adapter that should be added (*);`parameters <value>` – The list of parameters for the billing adapter that should be added (*).**billing-adapter-del (Delete Billing Adapter)***Syntax:*`wyde billing-adapter-del arguments`*Arguments:*`name <value>` – The name of the billing adapter that should be deleted (*).**billing-adapter-set (Set Billing Adapter Properties)***Syntax:*`wyde billing-adapter-set arguments`*Arguments:*`name <value>` – The name of the billing adapter that should be updated (*);

description <value> – New description of the billing adapter that should be set;
 driver <value> – New driver name for the billing adapter that should be set;
 bindaddr <value> – New bind address for the billing adapter that should be set;
 parameters <value> – New list of parameters for the billing adapter that should be set.

billing-adapter-show (Show Billing Adapters)

Syntax:

```
wyde billing-adapter-show
```

billing-rule-add (Add Billing Rule)

Syntax:

```
wyde billing-rule-add arguments
```

Arguments:

name <value> – The name of the billing rule that should be added (*);
 description <value> – The description of the billing rule that should be added;
 rule <value> – The billing rule content that should be added (*), comma-separated values: { [file] [,] [localdb] [,] [adapter:<name>] }:
 o file – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file;
 o localdb – denotes that CDR information should be stored into *dnca_calls* local database;
 o adapter:<name> – denotes custom billing adapter name that should be used to store CDR information.

billing-rule-del (Delete Billing Rule)

Syntax:

```
wyde billing-rule-del arguments
```

Arguments:

name <value> – The name of the billing rule that should be deleted (*).

billing-rule-set (Set Billing Rule)

Syntax:

```
wyde billing-rule-set arguments
```

Arguments:

name <value> – The name of the billing rule that should be updated (*);
 description <value> – New description of the billing rule that should be set;
 rule <value> – New billing rule content that should be set, comma-separated values: { [file] [,] [localdb] [,] [adapter:<name>] }:
 o file – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file;
 o localdb – denotes that CDR information should be stored into *dnca_calls* local database;

- o `adapter:<name>` – denotes custom billing adapter name that should be used to store CDR information.

billing-rule-show (Show Billing Rules)

Syntax:

```
wyde billing-rule-show
```

bridge-add (Add WYDE Bridge)

Syntax:

```
wyde bridge-add arguments
```

Arguments:

`name <value>` – The name of the bridge that should be added (*);

`ip_addr <value>` – IP address of the bridge that should be added (*).

bridge-del (Delete WYDE Bridge)

Syntax:

```
wyde bridge-del arguments
```

Arguments:

`name <value>` – The name of the bridge that should be deleted (*).

bridge-show (Show WYDE Bridges)

Syntax:

```
wyde bridge-show
```

callflow-add (Add Call Flow)

Syntax:

```
wyde callflow-add arguments
```

Arguments:

`name <value>` – The name of the call flow (*);

`directory <value>` – Call flow directory path.

callflow-attr-set (Set Call Flow Attribute)

Syntax:

```
wyde callflow-attr-set arguments
```

Arguments:

`callflow <value>` – The name of the call flow (*);

`name <value>` – This call flow attribute name (*);

`value <value>` – This call flow attribute value.

callflow-attr-show (Show Call Flow Attributes)

Syntax:

```
wyde callflow-attr-show arguments
```

Arguments:

`callflow <value>` – The name of the call flow (*).

callflow-attr-update-db (Update Call Flow Attributes Definition in a Database)*Syntax:*

```
wyde callflow-attr-update-db arguments
```

Arguments:

```
callflow <value> – The name of the call flow (*).
```

callflow-del (Delete Call Flow)*Syntax:*

```
wyde callflow-del arguments
```

Arguments:

```
name <value> – The name of the call flow (*).
```

callflow-reload (Reload All Call Flows)*Syntax:*

```
wyde callflow-reload
```

Note:

This command works the same as *mf* console command: `callflow-reload (Reload Call Flows)`.

callflow-show (Show Call Flows Table)*Syntax:*

```
wyde callflow-show
```

conference-attr-del (Remove Conference Attribute Redefinition)*Syntax:*

```
wyde conference-attr-del arguments
```

Arguments:

```
number <value> – The conference number (*);
```

```
name <value> – This conference call flow attribute name (*).
```

conference-attr-set (Set Conference Attribute)*Syntax:*

```
wyde conference-attr-set arguments
```

Arguments:

```
number <value> – The conference number (*);
```

```
name <value> – This conference call flow attribute name (*);
```

```
value <value> – This conference call flow attribute new value (*).
```

conference-attr-show (Show Conference Attributes)*Syntax:*

```
wyde conference-attr-show arguments
```

Arguments:

```
number <value> – The conference number (*).
```

config-restore (Restore WYDE Configuration)*Syntax:*

```
wyde config-restore arguments
```

Arguments:

- `file <value>` – The file name to restore configuration settings (*);
- `force <value>` – Force required {yes|no} (default no):
 - o `yes` denotes that the configuration file should be restored even if the versions of previously saved configuration data and currently installed WYDE software are different;
 - o `no` denotes that the configuration file should be restored only if the versions of previously saved configuration data and currently installed WYDE software are the same.

config-save (Save WYDE Configuration)*Syntax:*

```
wyde config-save arguments
```

Arguments:

- `file <value>` – The file name to save configuration settings (*).

confuser-add (Add Conference User)*Syntax:*

```
wyde confuser-add arguments
```

Arguments:

- `accesscode <value>` – Access code, if this parameter is omitted unique numeric access code will be generated;
- `did <value>` – DID (DNIS) number (*);
- `conference <value>` – Conference number, if this parameter is omitted unique numeric access code will be generated;
- `conf-accesscode` – Set conference number the same as access code, if this parameter is transferred the conference number will be set equal to access code (if `conference` and `conf-accesscode` parameters both transferred, the `conference` parameter will be ignored and the conference number will be set equal to the access code);
- `role <value>` – Role in the conference (case-sensitive): *Host* or *Participant* or *Listener* (*);
- `subscriber <value>` – Subscriber login name, i.e. subscriber PIN (*).

confuser-del (Delete Conference User)*Syntax:*

```
wyde confuser-del arguments
```

Arguments:

- `accesscode <value>` – Access code (*).

confuser-show (Show Conference Users Table)*Syntax:*`wyde confuser-show arguments`*Arguments:*

`subscriber <value>` – Subscriber login name, i.e. subscriber PIN;
`accesscode <value>` – Access code.

db (Connect to WYDE Main, i.e. dnca, Database)*Syntax:*`wyde db`**db-bil (Connect to WYDE Billing, i.e. dnca_calls, Database)***Syntax:*`wyde db-bil`**db-init (Initialize of WYDE Database)***Syntax:*`wyde db-init arguments`*Arguments:*

`database <value>` – Denotes what kind of database should be initialized; the possible values: {Main|Billing} –

- o *Main* is being used for *dnca* database (default);
- o *Billing* is being used for *dnca_calls* database.

db-patch (Apply Last Patches for Databases)*Syntax:*`wyde db-patch`**did-add (Add DNIS/DID Number)***Syntax:*`wyde did-add arguments`*Arguments:*

`number <value>` – Dial-in number (*);
`callflow <value>` – The name of the call flow (*);
`description <value>` – DNIS (DID) description.

did-alias-add (Add DNIS/DID Number Alias)*Syntax:*`wyde did-alias-add arguments`*Arguments:*

`number <value>` – Main dial-in number (*);
`alias <value>` – Alias of the number (*);
`description <value>` – Description of the alias.

did-alias-del (Delete DNIS/DID Number Alias)*Syntax:*

```
wyde did-alias-del arguments
```

Arguments:

```
number <value> – Main dial-in number (*);  
alias <value> – Alias of the number (*).
```

did-alias-show (Show DNIS/DID Number Aliases)*Syntax:*

```
wyde did-alias-show arguments
```

Arguments:

```
number <value> – Main dial-in number (*).
```

did-alias-show-all (Show all DID Number Aliases)*Syntax:*

```
wyde did-alias-show-all
```

did-attr-del (Remove DNIS/DID Attribute Redefinition)*Syntax:*

```
wyde did-attr-del arguments
```

Arguments:

```
number <value> – DNIS (DID) number (*);  
name <value> – This DNIS call flow attribute name (*).
```

did-attr-set (Set DNIS/DID Attribute)*Syntax:*

```
wyde did-attr-set arguments
```

Arguments:

```
number <value> – DNIS (DID) number (*);  
name <value> – This DNIS call flow attribute name (*);  
value <value> – This DNIS call flow attribute new value (*).
```

did-attr-show (Show DNIS's Attributes)*Syntax:*

```
wyde did-attr-show arguments
```

Arguments:

```
number <value> – DNIS (DID) number (*).
```

did-del (Delete DNIS/DID Number)*Syntax:*

```
wyde did-del arguments
```

Arguments:

```
number <value> – DNIS (DID) number (*).
```

did-reload (Reload all DNISes/DIDs Caches)*Syntax:*

```
wyde did-reload
```

Note:

This command works the same as *mf* console command: `did-reload` (Reload all DNISes/DIDs Caches).

did-show (Show DNISes/DIDs Table)*Syntax:*

```
wyde did-show
```

drop-call (Drop Call)*Syntax:*

```
wyde drop-call arguments
```

Arguments:

`conference <value>` – the number of the conference which calls you wish to drop, use '*confless*' keyword instead of conference number to indicate calls which have not placed to any conference (*);
`call <value>` – the call session ID (*).

Note:

This command works the same as *mf* console command: `call-drop` (Drop Call in the Conference).

drop-conf (Drop Conference)*Syntax:*

```
wyde drop-conf arguments
```

Arguments:

`number <value>` – the number of the conference you wish to drop (disconnect), you can use the keyword *all* instead of conference number to drop all conferences.

Note:

This command works the same as *mf* console command: `conf-drop` (Drop Conference).

help (Show Help Page and Exit)*Syntax:*

```
wyde help arguments
```

Arguments:

`command` – the specific wyde command on which you would like to get help.

ivr (Connect to IVR/asterisk Console)*Syntax:*

```
wyde ivr
```

node-add (Add Node to the WYDE Bridge)*Syntax:*

```
wyde node-add arguments
```

Arguments:

`bridge <value>` – the name of the bridge, if this argument is omitted the node will be added for the current bridge (default bridge is defined in `/usr/local/DNCA/etc/dnca.conf` file);

`name <value>` – the name of the node that should be added (*);

`ivr_addr <value>` – the node IVR address, i.e. IP address and port of the node (*);

`ivr_maxcalls <value>` – the maximum number of calls on the IVR for the node;

`zone <value>` – the location zone for the node.

node-del (Delete Node from the WYDE Bridge)*Syntax:*

```
wyde node-del arguments
```

Arguments:

`bridge <value>` – the name of the bridge, if this argument is omitted the node will be deleted from the current bridge (default bridge is defined in `/usr/local/DNCA/etc/dnca.conf` file);

`name <value>` – the name of the node that should be deleted (*).

node-set (Set Node Properties)*Syntax:*

```
wyde node-set arguments
```

Arguments:

`bridge <value>` – the name of the bridge which node properties should be updated, if this argument is omitted the node for the current bridge will be taken (default bridge is defined in `/usr/local/DNCA/etc/dnca.conf` file);

`name <value>` – the name of the node which properties should be updated (*);

`ivr_addr <value>` – the node new IVR address, i.e. IP address and port of the node;

`ivr_maxcalls <value>` – new maximum number of calls on the IVR for the node;

`zone <value>` – new location zone for the node.

node-show (Show Nodes of WYDE Bridge)*Syntax:*

```
wyde node-show arguments
```

Arguments:

`bridge <value>` – the name of the bridge, if this argument is omitted the command shows nodes for all available bridges.

Note:

This command works similar to *mf* console command: `node-show` (Show *MF* Cluster Nodes List) , but it has few differences in format, output and returned data – this *wyde*

command shows nodes configuration in the database, *mf console node-show*
command shows the current nodes status.

register-license (Register License)

Syntax:

```
wyde register-license
```

set-email (Change Email Address)

Syntax:

```
wyde set-email arguments
```

Arguments:

email <value> – new notification email address (*);
bridge <value> – the name of the bridge which email should be changed, if this argument is omitted the changes will be made for the current bridge (default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file).

set-ip (Change IP Address)

Syntax:

```
wyde set-ip arguments
```

Arguments:

ip <value> – new IP address of the bridge (*);
bridge <value> – the name of the bridge which IP address should be changed, if this argument is omitted the changes will be made for the current bridge (default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file).

settings-edit (Edit WYDE System Settings)

Syntax:

```
wyde settings-edit arguments
```

Arguments:

bridge <value> – the name of the bridge which settings you would like to edit, if this argument is omitted the settings will be edited for the current bridge (default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file);
name <value> – the parameter name that should be edited (*);
value <value> – new parameter value that should be set.

settings-show (Show WYDE Settings)

Syntax:

```
wyde settings-show arguments
```

Arguments:

bridge <value> – the name of the bridge which settings you would like to show, if this argument is omitted the settings will be shown for the current bridge (default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file);
prefix <value> – the prefix of the parameters names that should be shown.

settings-update (Update WYDE Settings)*Syntax:*

```
wyde settings-update arguments
```

Arguments:

bridge <value> – the name of the bridge which settings should be updated, if this argument is omitted the settings are being updated for the current bridge (default bridge is defined in */usr/local/DNCA/etc/dnca.conf* file).

show-conf (Show Conference or Conferences List)*Syntax:*

```
wyde show-conf arguments
```

Arguments:

- number <value> – Conference number:
- if this parameter is omitted the command returns the list of all conferences on the bridge;
 - if the conference number is transferred the command returns the list of all calls joined to the requested conference;
 - use '*confless*' keyword instead of conference number to show calls which have not placed to any conference.

Note:

This command works the same as *mf* console command: *show* (Show Conferences and Calls).

status (Show WYDE Status)*Syntax:*

```
wyde status
```

subscriber-add (Add Subscriber)*Syntax:*

```
wyde subscriber-add arguments
```

Arguments:

- login <value> – Login name, i.e. subscriber PIN, if this parameter is omitted unique numeric subscriber PIN will be generated;
- name <value> – Subscriber's first name;
- lastname <value> – Subscriber's last name;
- parent <value> – Parent login name, i.e. parent subscriber PIN;
- password <value> – Subscriber's password;
- email <value> – Subscriber's email address.

subscriber-del (Delete Subscriber)*Syntax:*

```
wyde subscriber-del arguments
```

Arguments:

- login <value> – Login name, i.e. subscriber PIN (*).

subscriber-show (Show Subscribers Table)*Syntax:*`wyde subscriber-show`**transfer (Transfer Calls)***Syntax:*`wyde transfer arguments`*Arguments:*

One of the following arguments should be specified as the first argument of this command (*):

- o {node <node_id>|node all} – denotes the node identifier (or all nodes) from which the calls should be transferred;
- o {conference <conf_number>} – denotes the conference number for which the calls should be transferred;
- o {did <did>} – denotes the DNIS (DID) number for which the calls should be transferred;

destination <destination IP value> – denotes destination IP address to which the specified calls should be transferred (*).

Note:

This command works the same as *mf* console command: `transfer` (Transfer Calls).

version (Show WYDE Version)*Syntax:*`wyde version`**watch (Watch WYDE Status)***Syntax:*`wyde watch arguments`*Arguments:*

interval <value> – Watch interval in seconds to refresh the WYDE bridge status, if it is omitted the 5 seconds interval is used by default.

***mf* Console Command Reference**

Once you enter into *mf* console you can run any of the console command just typing the command name and optionally the command arguments:

command [arguments]

This chapter describes all available *mf* console commands with their arguments. For any of the commands the required arguments marked with asterisk (*); optional arguments are shown in square brackets – [...] – in the command syntax.

call-associate (Set Bundle for the Call)

Syntax:

```
call-associate <conf_number> <ses_id> [<audiokey>]
```

Arguments:

<conf_number> – the number of the conference for which call you wish to define the audio key, i.e. the bundle (*);

<ses_id> – the call session identifier (*);

<audiokey> – new audio key for the call, if this argument is omitted the audio key for this call will be set to 0, i.e. the call in this case will not belong to any bundle.

call-custom-name (Set Custom Name for the Call)

Syntax:

```
call-custom-name <conf_number> <ses_id> [<name>]
```

Arguments:

<conf_number> – the number of the conference for which call you wish to define the custom name (*);

<ses_id> – the call session identifier (*);

<name> – new custom name for the call, if this argument is omitted the empty custom name will be set to this call.

call-drop (Drop Call in the Conference)

Syntax:

```
call-drop <conf_number> <ses_id> [force [nocdr]]
```

Arguments:

<conf_number> – the number of the conference which calls you wish to drop (disconnect), use 'confless' keyword instead of conference number to indicate calls which have not placed to any conference (*);

<ses_id> – the call session identifier (*);

force – denotes that the command should force call drop;

nocdr – denotes that CDR record should not be created for this call.

Note:

This command works similar to *wyde* command: drop-call (Drop Call).

call-hold (Hold Call)*Syntax:*

```
call-hold {true|false} <conf_number> <ses_id>
```

Arguments:

{true|false} – *true* indicates that the call should be placed on hold, *false* indicates that the call should be taken of hold (*);

<conf_number> – the number of the conference which call you wish to place on hold or take of hold (*);

<ses_id> – the call session identifier (*).

call-move (Move Call to Other Conference)*Syntax:*

```
call-move <conf_number> <ses_id> <new_did_number>  
         <new_accesscode> [<new_role>]
```

Arguments:

<conf_number> – the source conference number whose call you would like to move to another conference (*);

<ses_id> – the call session identifier that you wish to move to another conference (*);

<new_did_number> – new (i.e. target) conference DNIS (DID) number where the call should be moved (*);

<new_accesscode> – the access code that should be used to join to new (i.e. target) conference (*);

<new_role> – 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).

Note:

This command is asynchronous.

call-mute (Mute Call)*Syntax:*

```
call-mute {false|strict|relaxed} <conf_number> <ses_id>
```

Arguments:

{false|strict|relaxed} – *false* indicates that the call should be un-muted, *strict* indicates that the call should be muted and participants can not un-mute themselves, *relaxed* indicates that the call should be muted but participants can un-mute themselves (*);

<conf_number> – the number of the conference which call you wish to mute or un-mute (*);

<ses_id> – the call session identifier (*).

call-qa-request (Start/Stop Q&A Request for the Call)*Syntax:*

```
call-qa-request {start|stop} <conf_number> <ses_id>
```

Arguments:

`{start|stop}` – *start* indicates that the call should start the request to speak (the request to ask the question), i.e. placed into Q&A queue, *stop* indicates that the call should stop (cancel) the request to speak, i.e. removed from Q&A queue (*);
`<conf_number>` – the number of the conference where Q&A session is started (*);
`<ses_id>` – the call session identifier that should be placed into Q&A queue or removed from it (*).

Note:

When Q&A session is started the participants can use *6 (default) on their DTMF keypads and confirm that he wants to ask a question. The participants will be placed in a queue in the order that they requested to speak.

call-qa-talk (Enable/Disable Q&A Session for the Call in the Queue)*Syntax:*

```
call-qa-talk {enable|disable} <conf_number> <ses_id>
```

Arguments:

`{enable|disable}` – *enable* indicates that the call from Q&A queue should be unmuted and the participants should be able to speak (to ask his question), *disable* indicates that the unmuted call should be muted and removed from Q&A queue (*);
`<conf_number>` – the number of the conference where Q&A session is started (*);
`<ses_id>` – the call session identifier that should be unmuted to ask his question or should be muted again (*).

Note:

When Q&A session is started the hosts can use their DTMF keypads to manage Q&A sessions: default *1 2 can be used to move to the next questioner and *1 4 can be used to mute or unmute the active questioner.

callflow-reload (Reload Call Flows)*Syntax:*

```
callflow-reload
```

Note:

This command works the same as *wyde* command: `callflow-reload` (Reload All Call Flows).

cmdcount-show (Display Values of Command Counters)*Syntax:*

```
cmdcount-show
```

Note:

Reports the number of commands executed on bridge from *mf* starts till now. The following counters are being returned by this command:

- o `rt_drop` – number of DROP commands from RT;
- o `rt_mute` – number of MUTE commands from RT;
- o `rt_hold` – number of HOLD commands from RT;
- o `rt_mutegroup` – number of MUTE-GROUP commands from RT;

- o `rt_holdgroup` – number of HOLD-GROUP commands from RT;
- o `rt_setrole` – number of SET-ROLE commands from RT;
- o `rt_broadcast` – number of START-BROADCAST commands from RT;
- o `rt_playfile` – number of PLAY-FILE commands from RT;
- o `rt_customname` – number of SET-CUSTOMNAME commands from RT;
- o `rt_secure` – number of SECURE commands from RT;
- o `rt_recording` – number of START-RECORDING commands from RT;
- o `rt_broadcast` – number of START-BROADCAST commands from RT;
- o `rt_setrole` – number of SET-ROLE commands from RT;
- o `rt_jobcode` – number of SET-JOBCODE commands from RT;
- o `rt_qamode` – number of QA-MODE commands from RT;
- o `rt_qarequest` – number of QA-REQUEST commands from RT;
- o `rt_qatalk` – number of QA-TALK commands from RT;
- o `rt_subconf` – number of SUB-CONFERENCE commands from RT;
- o `cn_drop` – number of DROP commands from web or *mf* console;
- o `cn_mute` – number of MUTE commands from web or *mf* console;
- o `cn_hold` – number of HOLD commands from web or *mf* console;
- o `cn_customname` – number of SET-CUSTOMNAME commands from web or *mf* console;
- o `cn_move` – number of re-attach commands from web or *mf* console;
- o `cn_secure` – number of SECURE commands from web or *mf* console;
- o `cn_mutegroup` – number of MUTE-GROUP commands from web or *mf* console;
- o `cn_holdgroup` – number of HOLD-GROUP commands from web or *mf* console;
- o `cn_recording` – number of RECORDING commands from web or *mf* console;
- o `cn_broadcast` – number of START-BROADCAST commands from web or *mf* console;
- o `cn_playfile` – number of PLAY-FILE commands from web or *mf* console;
- o `cn_dialout` – number of DIALOUT commands from web or *mf* console;
- o `cn_setrole` – number of SET-ROLE commands from web or *mf* console;
- o `cn_jobcode` – number of JOB-CODE commands from web or *mf* console;
- o `cn_transfer` – number of TRANSFER commands from web or *mf* console;
- o `cn_qamode` – number of QA-MODE commands from web or *mf* console;
- o `cn_qarequest` – number of QA-REQUEST commands from web or *mf* console;
- o `cn_qatalk` – number of QA-TALK commands from web or *mf* console;
- o `cn_subconf` – number of SUBCONF commands from web or *mf* console;
- o `dtmf_mute` – number of MUTE commands from DTMF;
- o `dtmf_secure` – number of SECURE commands from DTMF;
- o `dtmf_mutegroup` – number of MUTEGROUP commands from DTMF;
- o `dtmf_broadcast` – number of START-BROADCAST commands from DTMF;

- o `dtmf_recording` – number of START-RECORDING commands from DTMF;
- o `dtmf_jobcode` – number of SET-JOBCODE commands from DTMF;
- o `dtmf_qamode` – number of QA-MODE commands from DTMF;
- o `dtmf_garequest` – number of QA-REQUEST commands from DTMF;
- o `dtmf_qatalk` – number of QA-TALK commands from DTMF;
- o `dtmf_subconf` – number of SUBCONFERENCE commands from DTMF.

conf-broadcast (Start/Stop Broadcast Mode for Listeners)

Syntax:

```
conf-broadcast {start|stop} <conf_number>
```

Arguments:

`{start|stop}` – denotes should the conference be broadcasted to listeners or not: *start* indicates that the listeners should hear the conference, *stop* indicates that all listeners should be on hold;

`<conf_number>` – the number of the conference for which you would like to change the broadcast mode.

conf-drop (Drop Conference)

Syntax:

```
conf-drop <conf_number>|all
```

Arguments:

One of the following arguments should be specified (*):

- o `<conf_number>` – the number of the conference you wish to drop (disconnect);
- o `all` – the keyword determines that all conferences should be dropped.

Note:

This command works the same as *wyde* command: `drop-conf` (Drop Conference).

conf-hold-group (Hold Group)

Syntax:

```
conf-hold-group {true|false} {participant|listener}
<conf_number>
```

Arguments:

`{true|false}` – *true* indicates that the group (participants or listeners) should be placed on hold, *false* indicates that the group (participants or listeners) should be taken of hold (*);

`{participant|listener}` – denotes who (all participants or all listeners) should be placed on hold or taken of hold (*);

`<conf_number>` – the number of the conference which calls you wish to place on hold or take of hold (*).

conf-jobcode (Set Job Code for the Conference)

Syntax:

```
conf-jobcode <conf_number> [<code>]
```

Arguments:

`<conf_number>` – the number of the conference which job code you would like to set (*);
`<code>` – new job code for the conference.

conf-mute-group (Mute Group)*Syntax:*

```
conf-mute-group {false|strict|relaxed} {host|participant}
               <conf_number>
```

Arguments:

`{false|strict|relaxed}` – *false* indicates that the group (hosts or participants) should be un-muted, *strict* indicates that the group (hosts or participants) should be muted and participants can not un-mute themselves, *relaxed* indicates that the group (hosts or participants) should be muted but participants can un-mute themselves (*);
`{host|participant}` – denotes who (all hosts or all participants) should be muted or un-muted (*);
`<conf_number>` – the number of the conference which calls you wish to mute or un-mute (*).

conf-play-file (Manage of Playing File to the Conference)*Syntax:*

```
conf-play-file <conf_number> <ses_id> {assign <dir>
               <filename>|start|stop|seek <offset> <whence>}
```

Arguments:

`<conf_number>` – the number of the conference where you would like to play the audio file (*);
`<ses_id>` – the session identifier (usually the identifier of the control call) which should be used to play the audio file (*);
`{assign <dir> <filename>|start|stop|seek <offset> <whence>}` – one of the following arguments should be specified here (*):

- o `assign <dir> <filename>` – assign the file for the playback:
 - o `<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);
 - o `<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;
- o `start` – start the playback from the current position;
- o `stop` – stop the playback;
- o `seek <offset> <whence>` – seek the audio file playback indicator (pointer) on `offset` seconds relative to the parameter `whence`:
 - o `0` – starting from the beginning of the file;
 - o `1` – starting from the current position in the file;

- 2 – starting from the end of the file.

conf-polling (Conference Polling)

Syntax:

```
conf-polling <conf_number> {start <keys>|stop}
```

Arguments:

- <conf_number> – the number of the conference for which you would like to start or to stop the polling (*);
- {start <keys>|stop} – *start* indicates that the polling should be started for the conference, *stop* indicates that the polling should be stopped for the conference (*);
 - <keys> – available polling options (i.e. digits 1, 2, ..., 9, 0) that should be specified when the polling is started.

conf-qa-mode (Manage Q&A Sessions)

Syntax:

```
conf-qa-mode {start|stop|clear} <conf_number>
```

Arguments:

- {start|stop|clear} – *start* indicates that Q&A session should be started for the conference, i.e. the conference should be placed in *question* mode, *stop* indicates that Q&A session should be stopped for the conference, *clear* indicates that Q&A queue should be cleared for the conference (*);
- <conf_number> – the number of the conference for which you would like to manage the Q&A session (*).

Note:

To implement the same Q&A sessions management hosts can use DTMF keypad on their phones: Q&A session can be started using default *1 1; Q&A session can be stopped using default *1 3; Q&A queue can be cleared using default *1 5.

conf-qa-mute (Mute/Unmute Active Q&A Session)

Syntax:

```
conf-qa-mute {true|false} <conf_number>
```

Arguments:

- {true|false} – *true* indicates that the active Q&A session should be muted, *false* indicates that the active Q&A session should be unmuted (*); both these options do not remove the questioner from Q&A queue, the command just temporary allows/disallows the active questioner to speak;
- <conf_number> – the number of the conference where Q&A session is started and where you would like to mute/unmute the active Q&A session (*).

Note:

To implement the same Q&A sessions actions hosts can use DTMF keypad on their phones: to mute or unmute the active questioner in Q&A session default *1 4 should be used.

conf-qa-talk (Enable Q&A Session for the First Call in the Queue)*Syntax:*

```
conf-qa-talk <conf_number>
```

Arguments:

<conf_number> – the number of the conference where Q&A session is started and where you would like to enable (unmute) the first call in the Q&A queue; the active questioner (if exists) will be removed from Q&A queue, so this command ends Q&A session for the current questioner and starts it for the next one (*).

Note:

To implement the same Q&A sessions action, i.e. to allow the first questioner to speak, hosts can use DTMF keypad on their phones and press default *1 2.

conf-recording (Start/Stop Conference Recording)*Syntax:*

```
conf-recording {start|stop} <conf_number> [<accesscode>]
```

Arguments:

{start|stop} – denotes should the conference recording be started or stopped (*);
<conf_number> – the number of the conference you wish to record (*);
<accesscode> – denotes pin, i.e. password to the recording server if “*Recording method*” call flow attribute value is “*remote*” (this value can be either defined on call flow level or overridden on DNIS level).

conf-schedule-extend (Extend Scheduled Conference Duration)*Syntax:*

```
conf-schedule-extend <conf_number> <seconds>
```

Arguments:

<conf_number> – the number of the started scheduled conference which duration you would like to extend (*);
<seconds> – the increment in seconds that should be added to the allowed duration for this scheduled conference (*).

conf-schedule-incsize (Resize Scheduled Conference Subscription)*Syntax:*

```
conf-schedule-incsize <conf_number> <count>
```

Arguments:

<conf_number> – the number of the started scheduled conference where you would like to increase the number of the allowed participants (*);
<count> – the increment in the count of the conference participants that should be added to the number of maximal allowed the scheduled conference participants (*).

conf-secure (Secure Conference)*Syntax:*

```
conf-secure {secure|unsecure} <conf_number>
```

Arguments:

- { *secure* | *unsecure* } – denotes should the conference be made secured or unsecured (*);
- <*conf_number*> – the number of the conference you wish to make secured or unsecured (*).

conf-shunt (Make/Drop Shunt between Two Conferences)*Syntax:*

```
conf-shunt {start <conf_number> [<peer_conf_number>|
                                stop <conf_number>}
```

Arguments:

- { *start* | *stop* } – *start* denotes that the shunt between two conferences should be made (started), *stop* denotes that the shunt between two conferences should be dropped (stopped) (*);
- o <*conf_number*> – the number of the conference you wish to shunt or unshunt (*);
- o <*peer_conf_number*> – the number of the peer conference you wish to shunt with the first one.

confcount-show (Display Values of *confcount* Counters)*Syntax:*

```
confcount-show
```

Note:

Reports the break-up of conferences according to participants size from *mf* starts till now. The following counters are being returned by this command:

- o *co_total_count* – gives the total number of conferences;
- o *co_1_count* – is the total number of 1-person conferences;
- o *co_2_count* – is the total number of 2-person conferences;
- o *co_3_count* – is the total number of conferences with 3 people;
- o *co_le_10_count* – is the total number of conferences with 4-10 participants;
- o *co_le_100_count* – gives the total number of conferences with 11-100 participants;
- o *co_gt_100_count* – gives the total number of conferences with 100+ participants;
- o *co_avg_pt_size* – gives the average size of conferences hosted on bridge;
- o *co_max_pt_size* – max size of conference;
- o *co_min_pt_size* – min size of conference;
- o *co_md_pt_size* – median size of conference.

dc-show-bridges (Show Known DC Bridges)*Syntax:*

```
dc-show-bridges
```

dc-show-links (Show DC Links)*Syntax:*`dc-show-links`**dialout (Do Dialout)***Syntax:*`dialout <peer_number> <timeout> <originator_conf_number>
<did_number> <accesscode> [<role>]`*Arguments:*

- <peer_number> – denotes the phone number you wish to dial (*);
- <timeout> – denotes allowed timeout in seconds, i.e. how many seconds the call can wait the answer (*);
- <originator_conf_number> – the original conference number, i.e. the conference from which you would like to make dial-out, this conference must be started prior to dial-out (*);
- <did_number> – the target conference DNIS (DID) number where the call should be joined after the dial-out is complete (*);
- <accesscode> – the access code that should be used to join to the target conference (*);
- <role> – the role that should be granted to the call when it joins to the target conference (applicable only for call flows without authorization, for example *CONF* call flow).

Note:

This command is asynchronous.

dialout-attr (Show Dialout Attributes for Specified Conference)*Syntax:*`dialout-attr <originator_conf_number>`*Arguments:*

- <originator_conf_number> – the number of the started conference which dialout attributes, i.e. DNISes and access codes with roles, you would like to show (*).

did-reload (Reload DID Entries)*Syntax:*`did-reload`*Note:*

This command works the same as *wyde* command: `did-reload` (Reload all DNISes/DIDs Caches).

errcount-show (Display Values of Error Counters)*Syntax:*`errcount-show`

Note:

Reports the number of terminated/incomplete calls on bridge from *mf* starts till now.

The following counters are being returned by this command:

- o `pt_total_terminated` – gives the total number of terminated calls;
- o `pt_sip_terminated` – provides the total number of calls terminated because of incomplete SIP handshakes;
- o `pt_max_calls_terminated` – terminate by call amount limits;
- o `pt_agi_terminated` – terminated by *agiserver*;
- o `pt_mp_terminated` – terminated by *mp*;
- o `pt_asterisk_terminaned` – terminated by *asterisk*;
- o `pt_max_duration_terminated` – provides the total number of participants that exceeded the maximum call duration.

freenumbers-show (Show Free Number Leases)*Syntax:*

```
freenumbers-show
```

help (Show Help for Console Commands)*Syntax:*

```
help command
```

Arguments:

`command` – the specific *mf* console command on which you would like to get help.

moh-reload (Reload Customer's Music-On-Hold Prompts)*Syntax:*

```
moh-reload
```

node-reload (Reload MF Cluster Nodes List)*Syntax:*

```
node-reload
```

node-show (Show MF Cluster Nodes List)*Syntax:*

```
node-show
```

Note:

This command works similar to *wyde* command: `node-show` (Show Nodes of WYDE Bridge), but it has few differences in format, output and returned data – this *mf* console command shows the current nodes status, *wyde node-show* command shows nodes configuration in the database.

op-call-move (Move User that Currently Talking with Operator to other Conference)*Syntax:*

```
op-call-move <operator_number> <new_did_number>
            <new_accesscode> [<new_role>]
```

Arguments:

- <operator_number> – the operator conference number whose current call (i.e. the user that currently talking to the operator) you would like to move to another conference (*);
- <new_did_number> – new (i.e. target) conference DNIS (DID) number where the call should be moved (*);
- <new_accesscode> – the access code that should be used to join to new (i.e. target) conference (*);
- <new_role> – 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).

Note:

To implement the same operator conference actions, i.e. to attach current user to a different conference, operator conference hosts can use DTMF keypad on their phones and press default *5.

This command is asynchronous.

op-dialout (Initiate Dialout from Operator's Console)*Syntax:*

```
op-dialout <operator_number> <peer_number>
```

Arguments:

- <operator_number> – the operator conference number where you are initiating the dialout (*);
- <peer_number> – denotes the phone number you wish to dial (*).

Note:

To implement the same operator conference actions, i.e. to dial-out to another user, operator conference hosts can use DTMF keypad on their phones and press default *7.

This command is asynchronous.

op-listen (Listen Conference)*Syntax:*

```
op-listen <operator_number> {start <conf_number>
                             [{directlink|shunt} [mute]]|stop}
```

Arguments:

- <operator_number> – the operator conference number that would like to listen other users conferences (*);
- {start|stop} – *start* denotes that the listening should be started, *stop* denotes that the listening should be stopped (*);
 - o <conf_number> – the number of the conference you wish to start listening, must be indicated if *start* option is specified;

- {*directlink*|*shunt*} – *directlink* 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), if this argument is omitted the *directlink* mode is being used by default;
- *mute* – when specified denotes that the operator is muted, so he can only hear the specified conference and he is unable to talk.

Note:

To connect and listen another conference operator conference hosts can use DTMF keypad on their phones and press default *4, the operator also has the options to connect with current user (*shunt* mode) and without current user (*directlink* mode).

op-queue (Display Operator Calls Queue)*Syntax:*

```
op-queue
```

op-scan (Scan Conferences)*Syntax:*

```
op-scan <operator_number> {start|stop}
```

Arguments:

<operator_number> – the operator conference number where you would like to start or stop conference monitoring (surveillance call) (*);
 {start|stop} – *start* denotes that the monitoring should be started, *stop* denotes that the monitoring should be stopped (*).

Note:

To implement the same operator conference actions, i.e. to start or stop conference monitor (surveillance call) operator conference hosts can use DTMF keypad on their phones and press default *1.

op-show (Display Operators)*Syntax:*

```
op-show [<operator_number>]
```

Arguments:

<operator_number> – the operator conference number that you would like to show.

Note:

To hear the operator conference current status the operators can use DTMF keypad on their phones and press default *6.

op-talk (Operator Talk to User)*Syntax:*

```
op-talk <operator_number> {start|stop}
```

Arguments:

<operator_number> – the operator conference number where you would like to start or stop talking to a user (*);

{start|stop} – *start* denotes that the talking should be started, *stop* denotes that the talking should be stopped (*).

Note:

To implement the same operator conference actions, i.e. to start talking with the next user from the operator queue or stop talking to the user and return him to his conference or ivr operator conference hosts can use DTMF keypad on their phones and press default *2 to start talking or *3 to stop talking.

partcount-show (Display Values of *partcount* Counters)*Syntax:*

```
partcount-show
```

Note:

Reports the counters related to the participants from *mf* starts till now. The following counters are being returned by this command:

- o pt_total_count – gives the total number of participants;
- o pt_ivr_count – gives the total number of participants that does not joined to any conference;
- o pt_1_count – gives number of participants in 1-person conferences;
- o pt_2_count – gives number of participants in 2-person conferences;
- o pt_3_count – gives number of participants in 3-person conferences;
- o pt_le_10_count – gives number of participants from 4 to 10-person conferences;
- o pt_le_100_count – gives number of participants from 11 to 100-person conferences;
- o pt_gt_100_count – gives number of participants from 100+ conferences;
- o pt_pcmu_count – gives number of participants in PCMU codec;
- o pt_pcmua_count – gives number of participants in PCMA codec;
- o pt_isac_count – gives number of participants in ISAC codec;
- o pt_ilbc_count – gives number of participants in ILBC codec;
- o pt_g729a_count – gives number of participants in g729a codec;
- o pt_g722_count – gives number of participants in g722 codec;
- o pt_voip_full_count – gives number of participants that join via voip and have right to speak;
- o pt_voip_listen_count – gives number of participants that join via voip and does not have right to speak;
- o pt_voip_ctrl_count – gives number of control calls;
- o pt_voip_rec_count – gives number of recording calls;

- o `pt_pstn_full_count` – gives number of participants that join via pstn and have right to speak;
- o `pt_pstn_listen_count` – gives number of participants that join via pstn and does not have right to speak.

peer-reload (Reload Peers)

Syntax:

```
peer-reload
```

quit (Quit Console)

Syntax:

```
quit
```

set-log-level (Set Logger Level)

Syntax:

```
set-log-level {debug|event|info}
```

Arguments:

- o `info` – denotes that only information and error messages should be stored in the log file (default); `event` – denotes that additionally the log should contain all events logging; `debug` – denotes that debugging mode is switched on, so additionally debug messages should be added to the log.

Note:

Log file name: `/usr/local/DNCA/log/mf.log`.

settings-reload (Reload System Settings)

Syntax:

```
settings-reload
```

show (Show Conferences and Calls)

Syntax:

```
show [<number>|confless]
```

Arguments:

- o the command without parameters returns the list of all conferences on the bridge;
- o `<number>` – the conference number – in this case the command returns the list of all calls joined to the requested conference;
- o `confless` – the keyword determines that the command should show the calls which have not placed to any conference.

Note:

This command works the same as `wyde` command: `show-conf` (Show Conference or Conferences List).

transfer (Transfer Calls)*Syntax:*

```
transfer {node <node_id>|all}|{conference <conf_number>}|  
{did <did>} <destination_ip>
```

Arguments:

One of the following arguments should be specified as the first argument of this command (*):

- o {node <node_id>|all} – denotes the node identifier (or all nodes) from which the calls should be transferred;
- o {conference <conf_number>} – denotes the conference number for which the calls should be transferred;
- o {did <did>} – denotes the DNIS (DID) number for which the calls should be transferred;

<destination_ip> – denotes destination IP address to which the specified calls should be transferred (*).

Note:

This command works the same as *wyde* command: `transfer` (Transfer Calls).

welcomeprompt-reload (Reload Customer's Welcome Prompts)*Syntax:*

```
welcomeprompt-reload
```

***asterisk* Console Command Reference**

Once you enter into *asterisk* console you can run any of the console command just typing the command name and optionally the command arguments:

command [arguments]

This chapter describes available *asterisk* console *wyde* commands with their arguments. For any of the commands the required arguments marked with asterisk (*); optional arguments are shown in square brackets – [...] – in the command syntax.

wyde drop session (Drop Session)

Syntax:

```
wyde drop session <conf_number> <session_id>
```

Arguments:

<conf_number> – the number of the conference which call you wish to drop (disconnect) (*);

<session_id> – the call session identifier you wish to drop (*).

wyde show conferences (Show Active Conferences)

Syntax:

```
wyde show conferences
```

wyde show conference (Show Conference Members)

Syntax:

```
wyde show conference <number>
```

Arguments:

<number> – the number of the conference which calls you would like to show (*).

wyde show session queue (Show Session Events Queue)

Syntax:

```
wyde show session queue <conf_number> <session_id>
```

Arguments:

<conf_number> – the call conference number which events queue you would like to show (*);

<session_id> – the call session identifier which events queue you would like to show (*).

wyde show sessions (Show Sessions not Attached to Conference)

Syntax:

```
wyde show sessions
```

wyde show statistic (Show Statistic)

Syntax:

```
wyde show statistic
```

mp Console Command Reference

Once you enter into *mp* console you can run any of the console command just typing the command name and optionally the command arguments:

command [arguments]

This chapter describes available *mp* console commands with their arguments. For any of the commands the required arguments marked with asterisk (*); optional arguments are shown in square brackets – [...] – in the command syntax.

drop (Drop Boards and Calls)

Syntax:

```
drop {<boards <serial> [<serial>] ...>|  
      <calls <id> [<id>] ...>|all}
```

Arguments:

One of the following arguments should be specified as the argument of this command (*):

- o <boards <serial> [<serial>] ...> – denotes *MPw* boards that should be dropped (placed on hold); use <serial> to specify the boards that should be dropped;
 - ✓ this option works as toggle switch – if you repeat this command the board will be switched on;
- o <calls <id> [<id>] ...>|all> – denotes that the calls should be dropped from *mp*; use <id> to specify the call sessions identifiers that should be dropped or use *all* keyword to drop all calls.

kill (Stop Specific Timer or All Timers)

Syntax:

```
kill {<id> [<id>] ...|all}
```

Arguments:

<id> – the specific timer identifier that should be stopped; *all* – denotes that all started timers should be stopped (*).

restart (Restart *mp* Boards and Logs)

Syntax:

```
restart {<log [<level>]>|<board <serial>|all>}
```

Arguments:

One of the following arguments should be specified as the argument of this command (*):

- o <log [<level>]> – denotes that the log should be restarted; <level> – specifies what messages should be stored in the log: *3* – denotes that only information and error messages should be stored in the log file (default); *9* – denotes that debugging mode is switched on and all possible log messages should be added to the log file.

- `<board <serial>|all>` – denotes the *MPw* boards that should be restarted; use `all` keyword to restart all *MPw* boards.

show (Show Different *mp* Statistics)

Syntax:

```
show {sw|conf|conf <conf_number>|calls|stat|boards}
```

Arguments:

One of the following arguments should be specified as the argument of this command (*):

- `sw` – show the number of available *MPw* boards, i.e. backend components count;
- `conf` – show all conferences currently running on the *mp*;
- `conf <conf_number>` – show statistics for the specific conference only, including active speaker notifications;
- `calls [<id1>-<id2>]` – show all calls currently running on the *mp* (the command returns calls identifiers, conference numbers, IP addresses, ports, codecs, status, etc.);
if you specify the calls sessions identifiers range, only the calls within this range will be returned by the command;
- `stat` – show overall *mp* statistics about calls, conferences and *MPs* boards (*now* column shows current data, *total* column shows data from *mp* start, *peak* column shows acme/height value);
- `boards` – show detail *MPs* boards statistics, including the board current state, capacity, CPU load, number of conferences and calls, etc.

stop (Stop *mp* Components)

Syntax:

```
stop
```

timer (Start Timer Running *show* Command)

Syntax:

```
timer <interval> show <options>
```

Arguments:

- `<interval>` – denotes interval in seconds in which the *show* command should be re-run (*);
- `show <options>` – denotes specific *show* command with its options that should be repeated in the given *interval* (*).

Appendix A: Code Samples

Authorization Adapters

Sample of Authorization Adapter for Windows Active Directory (WinLdap)

```
package Auth::Adapter::WinLdap;

use Misc::Logger;
use Net::LDAP 0.33;
use Authn::SASL 2.10;

my %attr_map = (
    description => 'did_number',
    adminDescription => 'accesscode',
    adminDisplayName => 'role',
    uSNSource => 'conf_number'
);

sub factory {
    return new Auth::Adapter::WinLdap(@_);
}

sub new {
    my $self = {};
    my $class = shift;
    my $object = bless($self, $class);
    my $parameters = shift;

    $logger->debug("Create auth adapter for LDAP: parameters=$parameters");

    my @servers = ();

    foreach my $server_info (split(';', $parameters)) {
        my $server = {};
        ($server->{host}, $server->{port}, $server->{password}, $server->{base}) =
            split(':', _trim($server_info));
        $server->{port} = 389 if ($server->{port} eq '');
        push(@servers, $server);
    }

    $self->{ret_code} = -1;

    foreach my $server (@servers) {
        my $sasl = Authn::SASL->new(mechanism => 'GSSAPI');
        $self->{CLIENT} = new Net::LDAP($server->{host}, port => $server->{port},
                                         onerror => 'die', debug => 0);

        if (!defined($self->{CLIENT})) {
            $logger->error("Could not contact LDAP server $server->{host}");
            next;
        }

        $self->{CLIENT}->bind(sasl => $sasl);
        $self->{SERVER_BASE} = $server->{base};
        last;
    }

    return $object;
}

sub _trim {
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}
```

```

sub ldap_search {
    my ($self, $base, $filter) = @_;
    my $reply = undef;

    if (defined($self->{CLIENT})) {
        $logger->debug("LDAP search : base=$base, filter=$filter");
        my $mesg = $self->{CLIENT}->search(base => $base, filter => $filter);

        my @entries = $mesg->entries;
        my $entry = shift(@entries);
        if (defined($entry)) {
            $reply = {};
            my @attrs = $entry->attributes();

            foreach $attr (@attrs) {
                my $key = $attr_map{$attr};
                if ($key ne '') {
                    $reply->{$key} = $entry->get_value($attr);
                    $logger->debug("$key=$reply->{$key}");
                }
            }
        }

        if (defined($reply)) {
            $self->{ret_code} = 1;
        } else {
            $self->{ret_code} = 0;
        }
    }
    return $reply;
}

#####
# public methods
#####
sub get_confuser_by_accesscode {
    my ($self, $did_number, $accesscode) = @_;
    return undef if (!defined($self->{CLIENT}));

    my $base = "$self->{SERVER_BASE}";
    my $filter = "&(objectClass=controlAccessRight)(description=$did_number)(adminDescription=$accesscode)";

    $self->{confuser} = $self->ldap_search($base, $filter);

    if( defined($self->{confuser}) ) {
        $logger->debug("confuser found via ldap: did_number=$did_number, conf_number=$self->{confuser}->{conf_number}, accesscode=$self->{confuser}->{accesscode}, role=$self->{confuser}->{role}");
    } else {
        $logger->error("confuser not found via ldap: did_number=$did_number, accesscode=$accesscode, filter=$filter");
    }

    return $self->{confuser};
}

sub get_confuser_by_number {
    my ($self, $did_number, $conf_number) = @_;
    return undef if (!defined($self->{CLIENT}));

    my $base = "$self->{SERVER_BASE}";
    my $filter = "&(objectClass=controlAccessRight)(description=$did_number)(uSNSource=$conf_number)";

    $self->{confuser} = $self->ldap_search($base, $filter);

    if( defined($self->{confuser}) ) {
        $logger->debug("confuser found via ldap: did_number=$did_number, conf_number=$self->{confuser}->{conf_number}, accesscode=$self->{confuser}->{accesscode}, role=$self->{confuser}->{role}");
    }
}

```

```
    } elsif( $res != -1 ) {  
        $logger->error("confuser not found via ldap: did_number=$did_number,  
conf_number=${conf_number}");  
    }  
  
    return $self->{confuser};  
}  
  
sub get_conference_attributes {  
    my ($self) = @_;  
    return undef if (!defined($self->{CLIENT}));  
  
    my $base = "$self->{SERVER_BASE}";  
    my $filter = "&(objectClass=controlAccessRight) (uSNSource=$self->{confuser}->{conf_number})";  
  
    my $attributes = $self->ldap_search($base, $filter);  
    $attributes = {} if (!defined($attributes));  
  
    return $attributes;  
}  
  
sub ret_code {  
    my ($self) = @_;  
    return $self->{ret_code};  
}
```


Sample of Authorization Adapter for WYDE Radius (WYDERadius)

```

package Auth::Adapter::WYDERadius;

use Misc::Logger;
use Misc::Config;
use Misc::SystemSettings;
use Authen::Radius;

sub factory {
    return new Auth::Adapter::WYDERadius(@_);
}

sub new {
    my $self = {};
    my $class = shift;
    my $object = bless($self, $class);

    my $parameters = shift;

    my $conf = get_config();
    Authen::Radius->load_dictionary($conf->get('general_lib_dir')."/Auth/Radius/dictionary");

    $logger->debug("Create auth adapter for Radius: parameters=$parameters");

    my @servers = ();

    foreach my $server_info (split(',', $parameters)) {
        my $server = {};
        ($server->{secret}, $server->{host}) = split('@', _trim($server_info));
        push(@servers, $server);
    }

    $self->{servers} = \@servers;
    $self->{ret_code} = -1;

    return $object;
}

sub _trim {
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

sub get_radius_client {
    my ($server) = @_;
    my $radius = new Authen::Radius(
        Host => $server->{host},
        Secret => $server->{secret}
    );
    return $radius
}

sub send_radius_request {
    my ($client) = @_;

    $logger->debug("Sending request to RADIUS: username=$username, password=$password");

    $client->send_packet(ACCESS_REQUEST);
    my $reply = $client->recv_packet();

    $logger->debug("got reply from RADIUS: type=$reply");
    if( $reply != 2 ) {
        my $error = Authen::Radius::strerror();
        if ($error ne 'none') {
            $logger->error("Got error on RADIUS request : $error");
        }
    }
}

```

```

        return -1;
    }
    return 0;
}

return 1;
}

sub confuser_request {
    my ($self, $client, $username, $password) = @_;

    $client->add_attributes (
        { Name => 'User-Name', Value => $username },
        { Name => 'User-Password', Value => $password },
    );

    my $res = send_radius_request($client);
    return $res if ($res <= 0);

    $self->{confuser} = {};
    $self->{attributes} = {};

    for my $attr ( $client->get_attributes() ) {
        $logger->debug("---$attr->{Name}=$attr->{Value}");

        if( $attr->{Name} =~ /conf_number|role|subscriber_id/ ) {
            $self->{confuser}->{$attr->{Name}} = $attr->{Value};
        } else {
            $self->{attributes}->{$attr->{Name}} = $attr->{Value};
            $self->{attributes}->{$attr->{Name}} = "" if( $attr->{Value} eq '-' );
        }
    }

    $self->{confuser}->{subscriber_id} = -1 if( !defined($self->{confuser}->{subscriber_id}) );
};
return 1;
}

sub subscriber_request {
    my ($self, $client, $pin) = @_;

    $client->add_attributes (
        { Name => 'User-Name', Value => $pin },
        { Name => 'User-Password', Value => $pin },
    );

    my $res = send_radius_request($client);
    return $res if ($res <= 0);

    $self->{subscriber} = {};

    for my $attr ( $self->{radius}->get_attributes() ) {
        $logger->debug("---$attr->{Name}=$attr->{Value}");
        $self->{subscriber}->{$attr->{Name}} = $attr->{Value};
        $self->{subscriber}->{$attr->{Name}} = "" if( $attr->{Value} eq '-' );
    }
    return 1;
}

sub get_confuser_by_accesscode {
    my ($self, $did_number, $accesscode) = @_;
    my $res = -1;

    foreach my $server (@{$self->{servers}}) {
        my $client = get_radius_client($server);

        if (defined($client)) {
            $res = $self->confuser_request($client, $accesscode, $did_number);
            last if ($res != -1);
        }
    }
}

```

```

}

if( $res > 0 && $self->{confuser}->{conf_number} ne '' ) {
    $logger->debug("confuser found in radiusdb: did_number=$did_number,
        accesscode=${accesscode}, conf_number=$self->{confuser}->{conf_number},
        role=$self->{confuser}->{role}");
} elseif( $res != -1 ) {
    $logger->error("confuser not found in radiusdb: did_number=$did_number,
        accesscode=${accesscode}");
}

$self->{ret_code} = $res;
return $self->{confuser};
}

sub get_confuser_by_number {
    my ($self, $did_number, $conf_number) = @_;
    my $res = -1;

    foreach my $server (@{$self->{servers}}) {
        my $client = get_radius_client($server);

        if (defined($client)) {
            $res = $self->confuser_request($client, $accesscode, $did_number);
            last if ($res != -1);
        }
    }

    if( $res > 0 && defined($self->{confuser}->{accesscode}) ) {
        $logger->debug("confuser found in radiusdb: did_number=$did_number,
            conf_number=$self->{confuser}->{conf_number},
            accesscode=$self->{confuser}->{accesscode},
            role=$self->{confuser}->{role}");
    } elseif( $res != -1 ) {
        $logger->error("confuser not found in radiusdb: did_number=$did_number,
            conf_number=${conf_number}");
    }

    $self->{ret_code} = $res;
    return $self->{confuser};
}

sub get_conference_attributes {
    my ($self) = shift;
    return $self->{attributes};
}

sub get_subscriber_by_pin {
    my ($self, $pin) = @_;
    my $res = -1;

    foreach my $server (@{$self->{servers}}) {
        my $client = get_radius_client($server);

        if (defined($client)) {
            $res = $self->subscriber_request($client, $pin);
            last if ($res != -1);
        }
    }

    if( $res > 0 && defined($self->{subscriber}->{custom_name}) ) {
        $logger->debug("subscriber found in radiusdb: pin=$pin");
    } elseif( $res != -1 ) {
        $logger->error("subscriber not found in radiusdb: pin=$pin");
    }

    $self->{ret_code} = $res;
    return $self->{subscriber};
}

```

```
sub get_subscriber_by_id {  
    my ($self, $id) = @_;  
    $self->{ret_code} = 0;  
    return $undef;  
}  
  
sub ret_code {  
    my ($self) = @_;  
    return $self->{ret_code};  
}
```

Billing Adapters

Sample of Billing Adapter for Windows PostgreSQL Database (WINPGSQL)

```

package Billing::Adapter::WINPGSQL;

use IO::Socket::INET;

use Misc::Logger;
use Billing::Receiver;
use DBI;

sub factory {
    return new Billing::Adapter::WINPGSQL(@_);
}

sub new {
    my $self = {};
    my $class = shift;
    my $object = bless($self, $class);
    my $database = "dnca_calls";
    my $user = "WydeBillingAdapter";
    my $password = "123";
    my $bindaddr = shift;
    my $host = shift;

    $logger->info("Create adapter WINPGSQL : bindaddr=$bindaddr, parameters=$host");

    $self->{RECEIVER} = new Billing::Receiver($bindaddr);

    $self->{db} = DBI->connect("dbi:Pg:dbname=$database;host=$host", $user, $password)
        || proc_error("Connect: ".DBI::errstr);

    return $object;
}

sub run {
    my $self = shift;
    my $cdr;
    my $query;
    my $sth;

    while (1) {
        $cdr = $self->{RECEIVER}->get();

        $query = "INSERT INTO \"CDRs\" (\"CdrDATA\") VALUES (?);";
        $sth = $self->{db}->prepare( $query );

        my @data_array = ();
        foreach my $k (keys(%$cdr)) {
            push(@data_array, $k."=".$cdr->{$k});
        }
        my $data_str = join(',', @data_array);

        $sth->execute($data_str) || proc_error(DBI::errstr."\nquery: $query\ndata: $data_str\n");
    }
}

```

Sample of Billing Adapter for Microsoft SQL Database (MSSQL)

```

package Billing::Adapter::MSSQL;

use IO::Socket::INET;

use Misc::Logger;
use Billing::Receiver;
use DBI;
use Misc::Database;

sub factory {
    return new Billing::Adapter::MSSQL(@_);
}

sub new {
    my $self = {};
    my $class = shift;
    my $object = bless($self, $class);
    my $database = "dnca_calls";
    my $user = "WydeBillingAdapter";
    my $password = "123";
    my $bindaddr = shift;
    my $host = shift;

    $logger->info("Create adapter MSSQL : bindaddr=$bindaddr, parameters=$host");

    $self->{RECEIVER} = new Billing::Receiver($bindaddr);

    $self->{db} = DBI->connect("dbi:Sybase:server=$host:database=$database", $user,
        $password) || die("Connect: ".DBI::errstr."\n");

    return $object;
}

sub run {
    my $self = shift;
    my $cdr;
    my $query;
    my $sth;

    while (1) {
        $cdr = $self->{RECEIVER}->get();

        $query = "INSERT INTO CDRs (CdrDATA) VALUES (?);";
        $sth = $self->{db}->prepare( $query );

        my @data_array = ();
        foreach my $k (keys(%$cdr)) {
            push(@data_array, $k."=".$cdr->{$k});
        }
        my $data_str = join(',', @data_array);

        $sth->execute($data_str) || proc_error(DBI::errstr."\nquery: $query\ndata: $data_str\n");
    }
}

```

Appendix B: Support Resources

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

Support Documentation

In addition to this Guide, you may obtain other WYDE Voice documentation from WYDE Voice or from the 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 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.