



WYDE Authorization Guide

(version 3.1)

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://docs.wydevoice.com/>

May 18, 2012

Symbols and Notations in this Manual

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



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



Denotes any useful information or tips

Figure

Denotes any illustration

Table

Denotes any table

Text

Denotes any text output

Folder/File

Denotes any folders (paths) or files names

commands

Denotes any commands, attributes and parameters

Table of Contents

Symbols and Notations in this Manual	3
Table of Contents	4
Tables List	6
Figures List	7
Chapter 1: Introduction	8
Section 1.1: Authorization Overview	8
Section 1.2: Assumed Skills	8
Section 1.3: Architecture Overview	8
Section 1.4: Integration Adapters	8
Chapter 2: Authorization	10
Section 2.1: Standard Authorization Adapters and Methods	10
Section 2.2: Authorization Integration	12
Section 2.3: Custom Authorization Adapters and Methods	12
Section 2.4: WYDE Commands to Manage Authorization Adapters and Methods	13
Add an Authorization Adapter	13
Delete an Authorization Adapter	14
View Authorization Adapters	14
Add an Authorization Method	15
Delete an Authorization Method	16
Modify an Authorization Method	16
View Authorization Methods	17
Chapter 3: Samples of Authorization Adapters for LDAP and Radius	19
Section 3.1: Sample of Simple Authorization Adapter for LDAP	19
Sample of Active Directory Installation and Configuration on the Bridge	21
Sample of WYDE Bridge Configuration for Simple LDAP Authorization	22
Section 3.2: Sample of Enhanced Authorization Adapter for LDAP with Conferences Call Flow Attributes	22
Sample of WYDE Bridge Configuration for Enhanced LDAP Authorization	24
Section 3.3: Sample of Simple Authorization Adapter for Radius	25
Radius Server Installation and Configuration Sample	25
Sample of Database Access Configuration for Conference Authorization	27
Sample of WYDE Bridge Configuration for Radius Authorization	28
Section 3.4: Sample of Authorization Adapter for Radius with Conferences Call Flow Attributes	29
Sample of Database Access Configuration for Conference Call Flow Attributes Definition	29
Sample of WYDE Bridge Configuration for Radius Authorization	30
Chapter 4: wyde Authorization Command Reference	31
auth-adapter-add (Add <i>auth</i> Adapter)	31
auth-adapter-del (Delete <i>auth</i> Adapter)	31
auth-adapter-show (Show <i>auth</i> Adapters)	31
auth-method-add (Add <i>auth</i> Method)	31
auth-method-del (Delete <i>auth</i> Method)	31
auth-method-set (Set <i>auth</i> Method)	31

auth-method-show (Show <i>auth</i> Methods).....	32
auth-reload (Reload <i>auth</i> configuration)	32
Appendix A: Authorization Adapters Code Samples.....	33
Sample of Simple Authorization Adapter for Windows Active Directory (WinLdap)....	33
Sample of Enhanced Authorization Adapter for Windows Active Directory (WinLdapEnh).....	36
Sample of Authorization Adapter for WYDE Radius (WYDERadius)	39
Appendix B: Definitions, Acronyms and Abbreviations.....	43
Appendix C: Support Resources	46
Support Documentation.....	46
Web Support.....	46
Telephone Support.....	46
Email Support.....	46

Tables List

Table 1: Sample of Active Directory Conference Accounts Data for Simple Authorization Adapter	20
Table 2: Sample of Active Directory Conference Accounts Data for Enhanced Authorization Adapter	23

Figures List

Figure 1: WYDE Bridge Authorization via RADIUS Server	11
Figure 2: <i>wyde help auth-adapter-add</i> and <i>wyde auth-adapter-add</i> Commands Output Sample	14
Figure 3: <i>wyde auth-adapter-show</i> Command Output Sample	15
Figure 4: <i>wyde help auth-method-add</i> and <i>wyde auth-method-add</i> Commands Output Sample	16
Figure 5: <i>wyde help auth-method-set</i> and <i>wyde auth-method-set</i> Commands Output Sample	17
Figure 6: <i>wyde auth-method-show</i> Command Output Sample.....	18
Figure 7: Sample of Active Directory Structure for Simple Authorization Adapter.....	19
Figure 8: Sample of Active Directory Data for Simple Authorization Adapter.....	20
Figure 9: Sample of Active Directory Structure for Enhanced Authorization Adapter	23
Figure 10: Sample of Active Directory Data for Enhanced Authorization Adapter	24

Chapter 1: Introduction

This is the Authorization guide for the WYDE conferencing bridges (like SB-HD100, SB-HD1000, and SB-HD10000). Within this guide you will learn how to integrate WYDE bridge conferences authorization into your security system, i.e. how to verify the right to connect to the conference and specific role (host/moderator/listener) in the conference based on your organization security storage.

Section 1.1: Authorization Overview

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. This could be done either based on the standard WYDE bridge conference authorization features as well as using your own security infrastructure.

If you have your own security infrastructure the customized authorization adapter can be written to integrate your security into call flows authorization. This guide explains how to create custom authorization adapters and methods for these purposes.

Section 1.2: Assumed Skills

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

- PC usage
- WYDE system administration
- Operational experience in Linux/CentOS, including system administration
- Experience with LDAP and/or RADIUS usage and administration
- VOIP basics
- TCP/IP networking
- Command Line Administration Interface - User Guide (recommended)
- Web Administration Interface – User Guide (recommended)

Section 1.3: Architecture Overview

The WYDE architecture is made up of both hardware as well as software services 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.

Section 1.4: Integration Adapters

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

- **Billing service** – For billing purposes the WYDE bridge software can store and transmit CDRs (Call Detail Records), the CDR storage is the storage location for the individual call records. You can store this information into SQL database or use other data storage.
- **Authorization service** – This allows the WYDE software to integrate into the enterprise authorization 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.

This document is devoted to authorization process only. It explains how to develop your own authorization adapters to perform conference authorization and how to configure authorization methods. If you need additional documentation regarding to “*WYDE Command Line Administration Interface*” or “*WYDE Web Administration Interface*” please download it from the WYDE Voice documentation Web site as noted in Appendix C: Support Resources, Support Documentation section.

Chapter 2: Authorization

As it was previously mentioned the conference authorization can be made either based on the standard WYDE bridge software conference authorization features or the right to join to the conference and specific role (host/moderator/listener) in the conference could be defined based on your own security infrastructure using customized authorization adapter written to integrate your security into call flows authorization.

WYDE bridge authorization is being formed from *Authorization Adapters* and *Authorization 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.

In terms of WYDE bridge software the *Authorization Method* is the specific authorization adapters together with its parameters (if necessary) that are used to authorize in the conference by specific call flow or DNIS. The authorization methods are being used for the conference authorization configuration. The authorization method could be defined either on call flow level or on DNIS level.

Section 2.1: Standard Authorization Adapters and Methods

The following three predefined authorization adapters are included and supported by standard WYDE bridge software installation:

- *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. WYDE bridge authorization using RADIUS server is shown on Figure 1; in its work first the authorization adapter sends request to the database by means of Radius server and next it receives response from the database via Radius server as well.

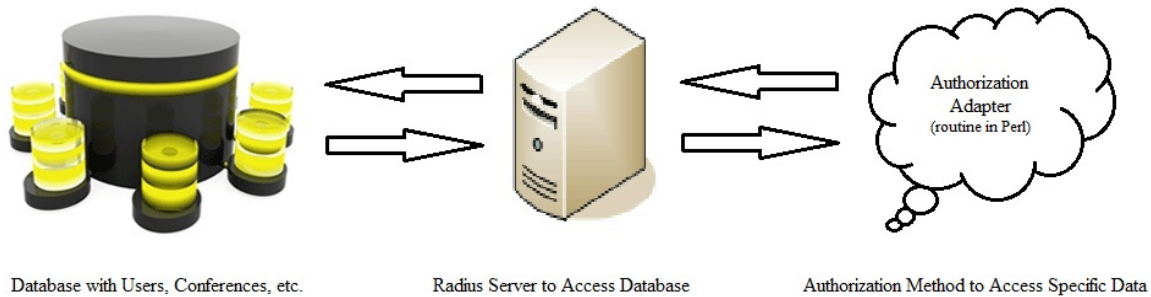


Figure 1: WYDE Bridge Authorization via RADIUS Server

- *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.

Of course, if you would like to use your company security infrastructure you can create your custom authorization adapter that will be responsible for verification if the user has the right to connect to the conference and what specific role should be granted to the user in the conference. This approach will be described in next sections of this guide.

There are three predefined standard authorization methods:

- *local* – Authorization via local database, *LocalDb* authorization adapter used;
- *wydelldap* – Authorization via LDAP, *WYDEldap* authorization adapter used;
- *wyderadius* – Authorization via Radius using WYDE dictionary, *WYDERadius* authorization adapter used.

They are included and supported by standard WYDE bridge software installation.

The authorization method name should be selected in `dnis_authORIZEMETHOD` (Authorize method) call flow attribute value either on call flow or on DNIS level.

Section 2.2: Authorization Integration

Conference authorization, i.e. defining the right to connect to the conference and specific role (host/moderator/listener) in the conference, can be made in one of the following ways:

1. You can do not use authorization, i.e. anyone who called to the conference DNIS number is allowed to connect to the conference regardless of access code entered. Usually this approach is used in CONF call flow.
2. Authorization can be made 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 *dnca* database 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 this approach is used in SPECTEL call flow.
3. Authorization can be made via RADIUS server using WYDE dictionary by means of *WYDERadius* standard authorization adapter as it was previously described.
4. Authorization can be made via LDAP using WYDE dictionary by means of *WYDEldap* standard authorization adapter as it was previously described.
5. Custom authorization adapter can be written to determine can or can not the user connect to the conference and if the connection is allowed what role should be granted to the user, i.e. should the user be host or participant or listener. Usual this information can be received either from your external SQL database (for instance using RADIUS server) or from Active Directory Domain Controller or others.

Section 2.3: Custom Authorization Adapters and Methods

As it was previously told 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.

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: *LocalDb.pm*, *WYDERadius.pm*, *WYDEldap.pm* – the authorization adapters supported by your bridge.

In addition to standard *LocalDb*, *WYDERadius*, and *WYDELDAP* authorization adapters described in previous sections of this guide you can create your own authorization adapters that will perform conference authorization according to security infrastructure of your organization. So the customized authorization adapter can be written to integrate your security into call flows authorization.

As it was previously mentioned 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_authmethod` (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 run the *wyde* command line utility with the *auth-reload* option:

```
wyde auth-reload
```

Section 2.4: WYDE Commands to Manage Authorization Adapters and Methods

Authorization adapters and methods can be managed using *wyde* command with different options that will be listed and described below. The command line interface is the powerful tool to administer your authorization adapters and authorization methods.

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:

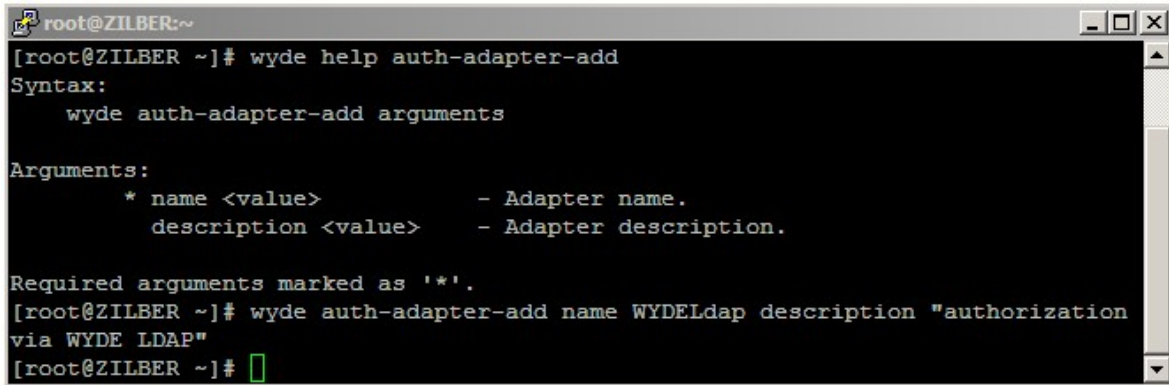
- `name <value>` – The name of the authorization adapter that should be added. This is required argument. This name should be unique, i.e. there should no be any other authorization adapter with the same name on the bridge.
- `description <value>` – The optional description of the authorization adapter 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 2.



```

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 2: *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
```

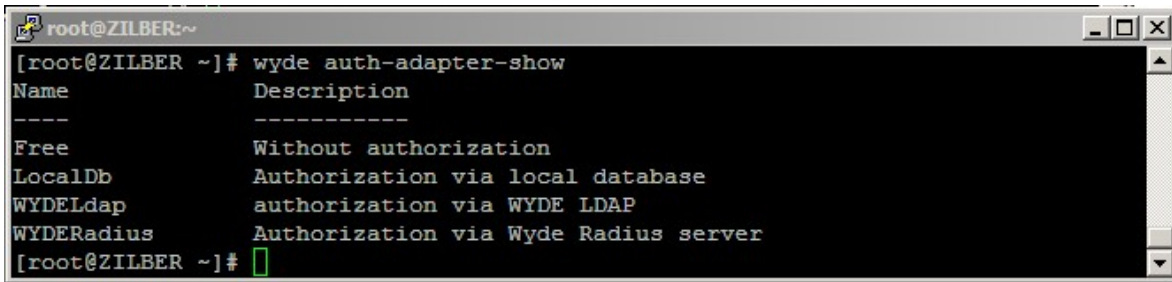
If deletion is successful, you will be returned to the command line with no additional prompts.

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 3. 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 3: *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).

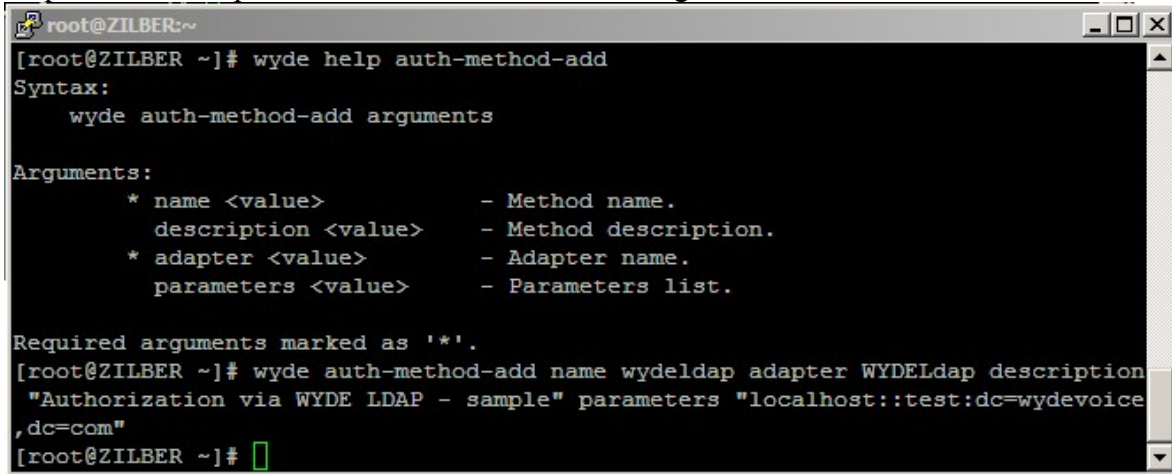
Arguments *name* and *adapter* are required. The arguments can be transferred to this command in any order.

For example if you would like to create the authorization method *wydelldap* 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 wydelldap 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 4.



```

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 4: *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.

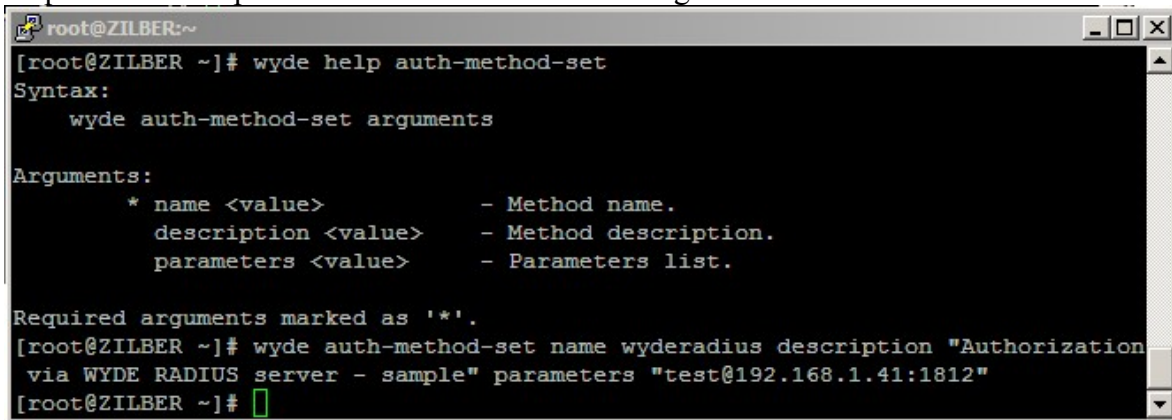
- `parameters <value>` – New list of parameters for the authorization method that should be set.

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 5.



```
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 ~]#
```

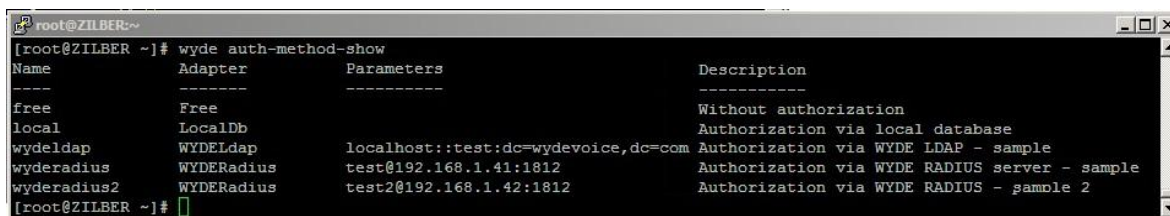
Figure 5: *wyde help auth-method-set* and *wyde auth-method-set* Commands Output Sample

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 existing authorization methods on the system, similar to shown on Figure 6. 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:~  
[root@ZILBER ~]# wyde auth-method-show  
Name           Adapter      Parameters      Description  
-----  
free           Free         Without authorization  
local          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 6: *wyde auth-method-show* Command Output Sample

Chapter 3: Samples 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.

Section 3.1: Sample of Simple Authorization Adapter for LDAP

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 7;

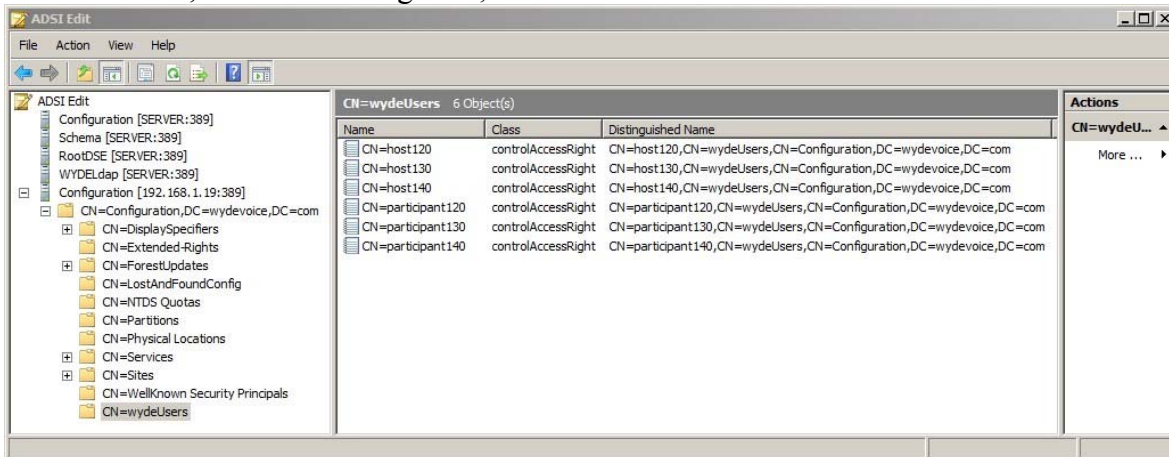


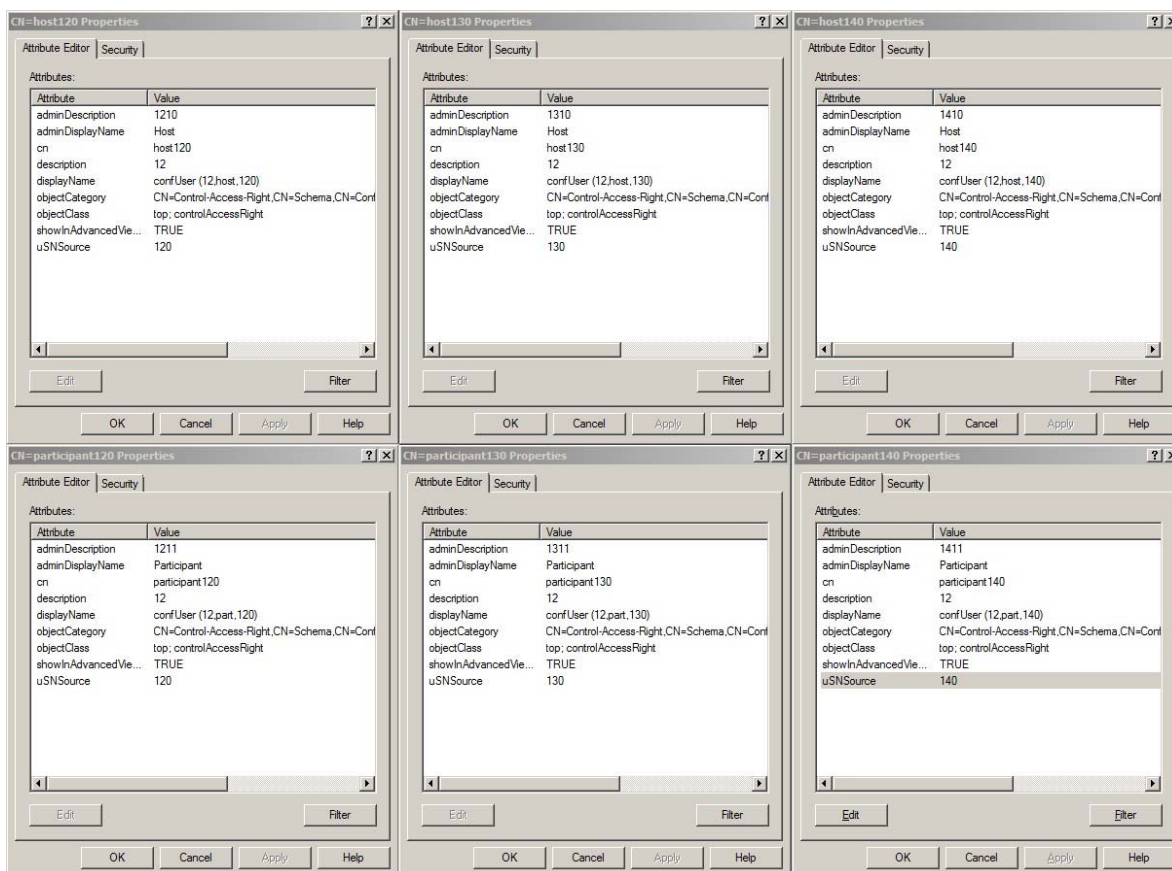
Figure 7: Sample of Active Directory Structure for Simple Authorization Adapter

- your domain address is *wydevoice.com*; the information is being stored under *Configuration* folder in *wydeUsers* subfolder (for conference accounts information); the object class that is used: *controlAccessRight*; if you wish to define the conference call flow attributes, these attributes should be defined using the same *controlAccessRight* class objects for the host's records;
- for this sample purposes we assume that three conferences are defined:
 - conference number 120, 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;

- conference number 130, two conference accounts with DNIS number 12 are being created for this conference: for the *Host* role with access code 1310 and for the *Participant* role with access code 1311;
 - conference number 140, two conference accounts with DNIS number 12 are being created for this conference: for the *Host* role with access code 1410 and for the *Participant* role with access code 1411;
- these data are being stored in Active Directory as shown in Table 1 and in Figure 8;

Table 1: Sample of Active Directory Conference Accounts Data for Simple Authorization Adapter

cn	conference accounts attributes	host120	participant120	host130	participant130	host140	participant140
displayName	(optional)	confUser	confUser	confUser	confUser	confUser	confUser
adminDescription	accesscode	(12,host,120)	(12,part,120)	(12,host,130)	(12,part,130)	(12,host,140)	(12,part,140)
adminDisplayName	role	Host	Participant	Host	Participant	Host	Participant
description	did_number	12	12	12	12	12	12
uSNSource	conf_number	120	120	130	130	140	140

**Figure 8: Sample of Active Directory Data for Simple Authorization Adapter**


- 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.](#)

Sample of Active Directory Installation and Configuration on the Bridge

Consider that to work with Active Directory, the DNS should be configured to resolve the names of your Active Directory domain controller and WYDE bridge computer, i.e. *ping* by computer names should work on both these computers for each of them.

Also note that to work with Active Directory from your bridge computer the following RPM packages should be installed: *cyrus-sasl*, *cyrus-sasl-gssapi*, *krb5-libs*, *krb5-workstation*, *perl-Authen-Krb5*, *perl-Authen-SASL*, *perl-GSSAPI*.

 You may use the command:
`yum install <package(s) name>`
 to install them on your WYDE bridge computer.

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:

```
[libdefaults]
    default_realm = WYDEVOICE.COM

[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.

Sample of WYDE Bridge Configuration for Simple LDAP Authorization

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 use the *wyde* command line utility with *auth-reload* option. The syntax is as follows:

```
wyde auth-reload
```

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 (Simple) "
wyde auth-method-add name winldap adapter WinLdap
    description
    "Authorization via Windows LDAP (Simple) - sample"
    parameters
    "192.168.1.19:::CN=wydeUsers,CN=Configuration,dc=wydevoice,dc=com"
```

Note that after you add the authorization adapter you also should use the *wyde* command line utility with *auth-reload* option:

```
wyde auth-reload
```

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*; for example you can use the command:

```
wyde did-attr-set number 12 name dnis_authorizemethod
    value 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.

Section 3.2: Sample of Enhanced Authorization Adapter for LDAP with Conferences Call Flow Attributes

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, host and participant access codes, *conference_start_how* (how conference begins) call flow attribute value, as well as your conferences definitions, as shown on Figure 9;

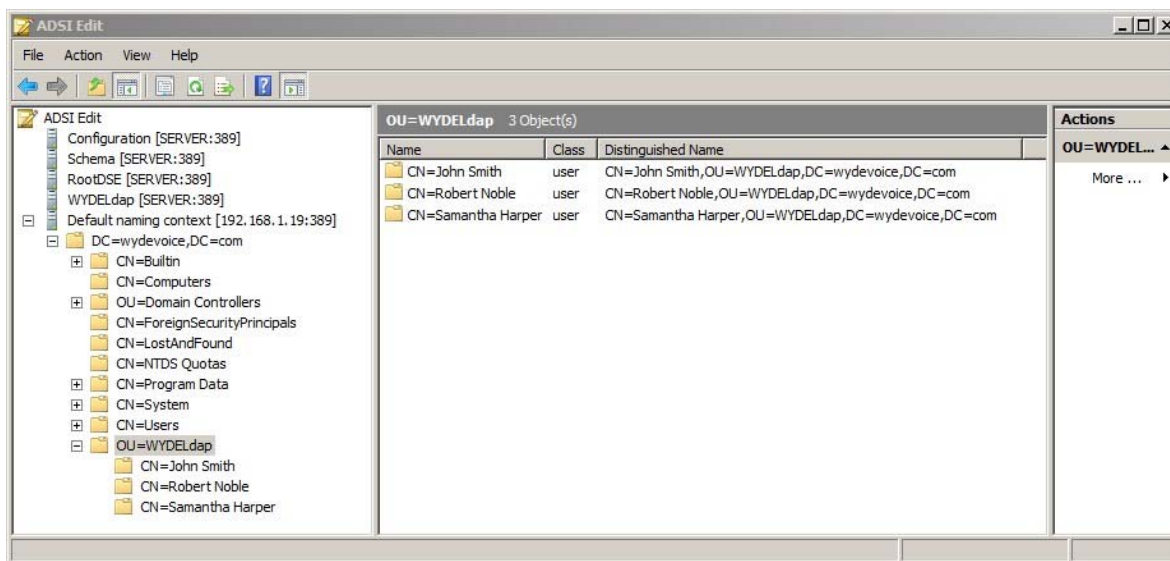


Figure 9: Sample of Active Directory Structure for Enhanced Authorization Adapter

- your domain address is *wydevoice.com*; the information is being stored under “*Default naming context*” folder in *WYDELdap* subfolder (for conference accounts information); the object class that is used: *user*;
- for this sample purposes we assume that three conferences are defined:
 - conference number *120*, DNIS number is *12*, the *Host* access code is *1210*, the *Participant* access code is *1211*, *conference_start_how* (how conference begins) call flow attribute value is *moderator*;
 - conference number *130*, DNIS number is *12*, the *Host* access code is *1310*, the *Participant* access code is *1311*, *conference_start_how* (how conference begins) call flow attribute value is *first*;
 - conference number *140*, DNIS number is *12*, the *Host* access code is *1410*, the *Participant* access code is *1411*, *conference_start_how* (how conference begins) call flow attribute value is *first*;
 these data are being stored in Active Directory as shown in Table 2 and in Figure 10;

Table 2: Sample of Active Directory Conference Accounts Data for Enhanced Authorization Adapter

cn	conference accounts attributes	Samantha Harper	John Smith	Robert Noble
department	conference_start_how	moderator	first	first
homeDirectory	conf_number	120	130	140
homePhone	did_number	12	12	12
homeDrive	host_accesscode	1210	1310	1410
homePostalAddress	participant_accesscode	1211	1311	1411

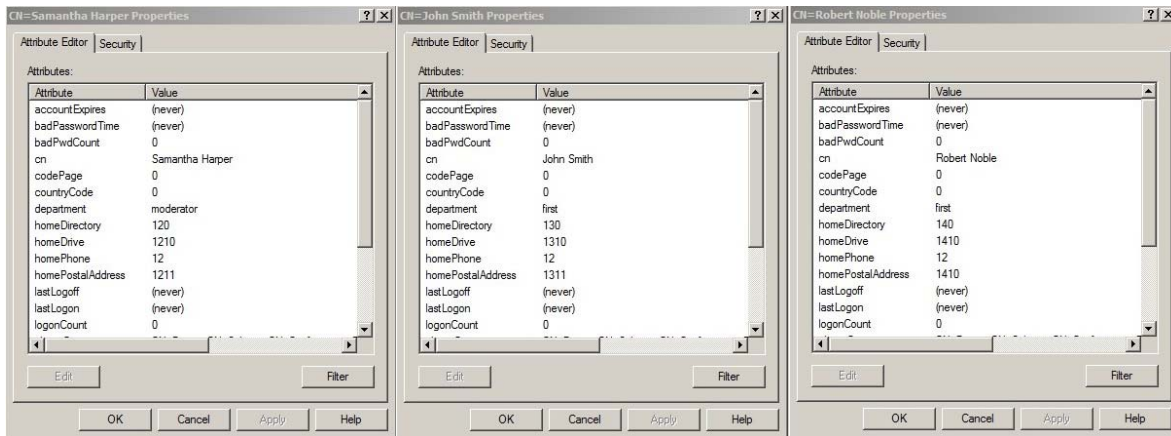


Figure 10: Sample of Active Directory Data for Enhanced Authorization Adapter

- 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 *WinLdapEnh* source code that we developed to implement this request.](#)

Installation and configuration of the Active Directory on your WYDE bridge was previously described in the section: Sample of Active Directory Installation and Configuration on the Bridge. Follow the steps described in that section, if you did not implemented them for the previous sample yet.

Sample of WYDE Bridge Configuration for Enhanced LDAP Authorization

When design of *WinLdapEnh.pm* file is completed you should copy this file into */usr/local/DNCA/lib/Auth/Adapter* folder and then you should use the *wyde* command line utility with *auth-reload* option. The syntax is as follows:

```
wyde auth-reload
```

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 WinLdapEnh
    description "Authorization via Windows LDAP (Enhanced) "
wyde auth-method-add name winldapenh adapter WinLdapEnh
    description
    "Authorization via Windows LDAP (Enhanced) - sample"
    parameters
    "192.168.1.19:::OU=WYDEldap,DC=wydevoice,DC=com"
```

Note that after you add the authorization adapter you also should use the *wyde* command line utility with *auth-reload* option:

```
wyde auth-reload
```


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 *winldapenh*; for example you can use the command:

```
wyde did-attr-set number 12 name dnis_authorizemethod
    value winldapenh
```

As soon as this has been done, 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.

Section 3.3: Sample of Simple Authorization Adapter for Radius

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.

Radius Server Installation and Configuration Sample

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
        }
    }
}

```

Sample of Database Access Configuration for Conference Authorization

After that to provide access to specific data from the database we should update the *wyde_sql.conf* configuration file from */etc/raddb* folder:

- this 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`.

Note that as *authorize_reply_query* and *authorize_reply_query* you can also use the stored procedures with parameters that return the same data as described above.

Sample of WYDE Bridge Configuration for Radius Authorization

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:

```
wyde auth-reload
```

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_authorizemethod* (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.

Section 3.4: Sample of Authorization Adapter for Radius with Conferences Call Flow Attributes

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

Sample of Database Access Configuration for Conference Call Flow Attributes Definition

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	=

Note that as *authorize_group_reply_query* you can also use the stored procedure with parameters that return the same data as described above.

Sample of WYDE Bridge Configuration for Radius Authorization

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.

Chapter 4: wyde Authorization Command Reference

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
```

auth-reload (Reload *auth* configuration)

Syntax:

```
wyde auth-reload
```


Appendix A: Authorization Adapters Code Samples

Sample of Simple 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};
    }

    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}) (adminDisplayName='Host')";

    my $attributes = $self->ldap_search($base, $filter);
    $attributes = {} if (!defined($attributes));

    delete($attributes->{conf_number});
    delete($attributes->{role});

    return $attributes;
}

sub ret_code {
    my ($self) = @_;
    return $self->{ret_code};
}

```

Sample of Enhanced Authorization Adapter for Windows Active Directory (WinLdapEnh)

```

package Auth::Adapter::WinLdapEnh;

use Misc::Logger;
use Net::LDAP 0.33;
use Authn::SASL 2.10;

my %attr_map = (
    homePhone => 'did_number',
    homeDrive => 'host_accesscode',
    homePostalAddress => 'participant_accesscode',
    homeDirectory => 'conf_number',
    department => 'conference_start_how'
);

sub factory {
    return new Auth::Adapter::WinLdapEnh(@_);
}

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};
    }

    return $object;
}

sub _trim {
    my $string = shift;
    $string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
}

```

```

sub ldap_search {
    my ($self, $base, $filter, $ac) = @_;
    my $res = -1;

    if (defined($self->{CLIENT})) {
        $logger->debug("LDAP search : base=$base, filter=$filter");
        my $mesg = $self->{CLIENT}->search(base => $base, filter => $filter);
        $res = 0;

        my @entries = $mesg->entries;
        my $entry = shift(@entries);
        if (defined($entry)) {
            $res = 1;
            self->{confuser} = {};
            $self->{attributes} = {};

            my @attrs = $entry->attributes();

            foreach $attr (@attrs) {
                my $key = $attr_map{$attr};
                if ($key ne '') {
                    my $value = $entry->get_value($attr);
                    $logger->debug("-- $key = $value");

                    if ($key eq 'host_accesscode') {
                        if ($value eq $ac) {
                            $self->{confuser}->{role} = 'Host';
                            $logger->debug("-- role = Host");
                        }
                    }
                    elsif ($key eq 'participant_accesscode') {
                        if ($value eq $ac) {
                            $self->{confuser}->{role} = 'Participant';
                            $logger->debug("-- role = Participant");
                        }
                    }
                    elsif ($key eq 'conf_number') {
                        $self->{confuser}->{conf_number} = $value;
                    }
                    elsif ($key ne 'did_number') {
                        $self->{attributes}->{$key} = $value;
                    }
                }
            }
        }
    }
    return $res;
}

#####
# public methods
#####
sub get_confuser_by_accesscode {
    my ($self, $did_number, $accesscode) = @_;

    if (!defined($self->{CLIENT})) {
        $self->{ret_code} = -1;
        $self->{error} = "LDAP error";
        return undef;
    }

    my $base = "$self->{SERVER_BASE}";
    my $filter =
        "&(objectClass=user)(homePhone=$did_number)(|(homeDrive=$accesscode)(homePostalAddress=$accesscode))";

    my $res = $self->ldap_search($base, $filter, $accesscode);

    if( $res > 0 && $self->{confuser}->{conf_number} ne '' ) {

```

```

    $logger->debug("confuser found via ldap: did_number=$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");
}

$self->{ret_code} = $res;
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 $filter = "&(objectClass=user) (homePhone=$did_number) (homeDirectory=$conf_number)";

    $self->{confuser} = $self->ldap_search($base, $filter, $accesscode);

    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 $self->{attributes};
}

sub ret_code {
    my ($self) = @_;
    return $self->{ret_code};
}

sub error {
    my ($self) = @_;
    return $self->{error};
}

```

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};  
}
```

Appendix B: Definitions, Acronyms and Abbreviations

While we discussed the WYDE Bridge Authorization process in this guide, we used a common set of terminology. Here we provide the dictionary for the terms you could 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 un-muted to enable them to talk; however, the listener group as a whole will never be un-muted.
- **Authorization Adapter** – A component (function) responsible for specifying access rights in the conferences. 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.
- **Authorization Method** – A method that is used to determine the specific authorization adapter that is applied to authorize in the conference and if necessary the parameters that could be transferred to this authorization adapter. The authorization method could be defined either on call flow level or on DNIS level.

Appendix C: Support Resources

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

Support Documentation

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

Web Support

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

Telephone Support

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

Email Support

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