



# WYDE Billing Guide

(version 3.0)

**Disclaimer**

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

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

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

**Copyright**

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

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

For the most current versions of documentation, go to the WYDE support Web site:

<http://docs.wydevoice.com/>

November 14, 2011

## Symbols and Notations in this Manual

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



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

<b>Figure</b>	Denotes any illustration
<b>Table</b>	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 .....	6
Figures List .....	7
Chapter 1: Introduction .....	8
Section 1.1: Billing Overview .....	8
Section 1.2: Assumed Skills .....	8
Section 1.3: Architecture Overview .....	8
Section 1.4: Integration Adapters .....	8
Section 1.5: Internal Reporting .....	9
Chapter 2: Billing .....	10
Section 2.1: CDR Format .....	10
Section 2.2: Standard Billing Adapters and Rules .....	12
Section 2.3: Billing Integration .....	14
Section 2.4: Custom Billing Adapters and Rules .....	14
Section 2.5: WYDE Commands to Manage Billing Adapters and Rules .....	16
Add a Billing Adapter .....	16
Delete a Billing Adapter .....	17
Modify a Billing Adapter .....	18
View Billing Adapters .....	19
Add a Billing Rule .....	19
Delete a Billing Rule .....	20
Modify a Billing Rule .....	21
View Billing Rules .....	22
Billing Configurations Reloading .....	22
Chapter 3: Samples of Billing Adapters .....	23
Section 3.1: Sample of Calls Billing Adapter to Text File .....	23
Sample of WYDE Bridge Configuration for Calls Text File Billing Adapter .....	23
Section 3.2: Sample of Conferences Billing Adapter to Text File .....	24
Sample of WYDE Bridge Configuration for Conferences Text File Billing Adapter .....	25
Section 3.3: Sample of Billing Adapter to Windows PostgreSQL Database .....	25
PostgreSQL Database Access Configuration Sample .....	26
Sample of WYDE Bridge Configuration for PostgreSQL Billing Adapter .....	26
Section 3.4: Sample of Billing Adapter to Windows Microsoft SQL Database .....	27
Microsoft SQL Server Installation and Configuration Sample .....	27
Microsoft SQL Database Access Configuration Sample .....	28
Sample of WYDE Bridge Configuration for Microsoft SQL Billing Adapter .....	28
Chapter 4: wyde Billing Command Reference .....	29
billing-adapter-add (Add Billing Adapter) .....	29
billing-adapter-del (Delete Billing Adapter) .....	29
billing-adapter-set (Set Billing Adapter Properties) .....	29
billing-adapter-show (Show Billing Adapters) .....	29
billing-reload (Reload Billing Configuration) .....	29
billing-rule-add (Add Billing Rule) .....	29

billing-rule-del (Delete Billing Rule) .....	30
billing-rule-set (Set Billing Rule) .....	30
billing-rule-show (Show Billing Rules) .....	30
Appendix A: Billing Adapters Code Samples .....	31
Billing Adapter Base Class (Adapter.pm) .....	31
Sample of Calls Billing Adapter for Text File (TEXTCSV) .....	32
Sample of Conferences Billing Adapter for Text File (TEXTCONF) .....	33
Sample of Billing Adapter for Windows PostgreSQL Database (WINPGSQL) .....	34
Sample of Billing Adapter for Microsoft SQL Database (MSSQL) .....	35
Appendix B: CDR Data Structures .....	36
CDR.log File Data Structure .....	36
Local dnca_calls Database calls Table Data Structure and Samples .....	37
Local dnca_calls Database conferencedr Table Data Structure and Samples .....	39
Appendix C: Definitions, Acronyms and Abbreviations .....	40
Appendix D: Support Resources .....	43
Support Documentation .....	43
Web Support .....	43
Telephone Support .....	43
Email Support .....	43

***Tables List***

Table 1: Input CDR Data Format for Billing Adapters .....	10
---	----

### ***Figures List***

Figure 1: WYDE Bridge Standard and Custom Billing Adapters Architecture Samples ....	13
Figure 2: <i>wyde help billing-adapter-add</i> and <i>wyde billing-adapter-add</i> Commands Output Sample .....	17
Figure 3: <i>wyde help billing-adapter-set</i> and <i>wyde billing-adapter-set</i> Commands Output Sample .....	19
Figure 4: <i>wyde billing-adapter-show</i> Command Output Sample .....	19
Figure 5: <i>wyde help billing-rule-add</i> and <i>wyde billing-rule-add</i> Commands Output Sample .....	20
Figure 6: <i>wyde help billing-rule-set</i> and <i>wyde billing-rule-set</i> Commands Output Sample .....	22
Figure 7: <i>wyde billing-rule-show</i> Command Output Sample .....	22

## Chapter 1: Introduction

This is the Billing 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 calls information into your billing system, i.e. how to transmit and store the calls information in your specific data storage.

### *Section 1.1: Billing Overview*

For billing purposes the WYDE bridge software can store and transmit CDRs (Call Detail Records). Please 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.

### *Section 1.2: Assumed Skills*

This billing 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
- 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 another data storage.
- **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.

This document is devoted to billing adapters only. It explains how to develop your own billing adapters to store calls and conferences information into your database or other data storage. 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 D: Support Resources, Support Documentation section.

### ***Section 1.5: Internal Reporting***

Conference Bridge itself provides comprehensive reports allowing Administrator to verify what calls in what conferences took place. These reports are available through the standard Web Administration interface that comes with the bridge. They include following reports:

1. Calls report
  - ✓ gives Administrator a possibility to review individual CDRs
2. Conference Report
  - ✓ allows to browse conferences (that took place in the past). Additionally this report lets to playback conference recording (if it was recorded) and plot a Gantt chart of the calls belonging to this conference.
3. Disconnect report
  - ✓ allows to see calls distribution by different disconnect reasons. This is very helpful for troubleshooting.
4. DNIS report
  - ✓ allows to see calls distribution by different called numbers.
5. Load charts
  - ✓ shows actual port utilization over time for the desired interval.

To see detailed information about these reports please open chapter “*Web Report Management*” in document “*Web Administration Interface – User Guide*”. You can download this guide from the WYDE Voice documentation Web site as noted in Appendix D: Support Resources, Support Documentation section.

## Chapter 2: Billing

As it was previously mentioned WYDE bridge software allows you to store, process and transmit data that could be used for billing purposes. To do so you could either use standard billing adapters provided with the WYDE bridge software or write you own billing adapters.

WYDE bridge billing is being formed from CDR information generated by *MF*, *Billing Adapters* and *Billing Rules*.

WYDE bridge *MF* service writes CDR information about completed calls into internal journal; such CDRs journal is being created for each billing adapter. That journal could be accessed by the billing adapter to catch and store CDR information.

In terms of WYDE bridge software the *Billing Adapter* is the component (function) responsible for storing billing, i.e. CDR information. Billing adapter processes information from the internal CDRs journal created by *MF* service; in this journal *MF* keeps 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.

In terms of WYDE bridge software the *Billing Rule* is the set of billing adapters that are being used to store CDR information by specific call flow or DNIS. The billing rules are being used for the conference billing information storing. 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. *{adapter1[,adapter2[,adapter3[,...]]]}* (for example *file,localdb* or *file,localdb,myAdapter*). The billing rule could be defined either on call flow level or on DNIS level.

### Section 2.1: CDR Format

The input format for billing adapters, i.e. the CDR information about the completed calls that comes from *MF* to the journal used by billing adapter is shown in Table 1. This information goes in chronological order of when the call was completed, i.e. when CDR record was created for the call. The provided table contains names of the fields, description of the fields, and CDR data samples.

**Table 1: Input CDR Data Format for Billing Adapters**

Field	Description	Data Samples
access_code	Access code used	505052
addr_from	Full address FROM, i.e. full qualified SIP URI of caller's address	<sip:4024684432@192.168.1.5>
addr_to	Full address TO, i.e. full qualified SIP URI of callee's address	"8665080020" <sip:8665080020@192.168.1.5>
audio_key	Audio key assigned to this call or empty	690
bridge	Bridge name	WYDE5
call_created	Date and time when the call was created (started)	2010-11-09 17:39:07+02

Field	Description	Data Samples
call_dropped	Date and time when the call was dropped (ended)	2010-11-09 17:55:12+02
call_duration	Duration of the call in seconds	965
call_id	Call identifier	Integer call identifier
called_number	Called number, i.e. the number to which the caller had called	8665080020
callflow	Call flow name (for instance, <i>CONF</i> , <i>PLAYBACK</i> , <i>OPERATOR</i> , <i>SPECTEL</i> , etc.)	SPECTEL
calling_number	Incoming calling number, i.e. the number from which called the caller or empty	4024684432
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	2
conf_id	Conference identifier	Integer conference identifier
conf_number	Conference number	889900
connection_type	Connection type, i.e. call direction ( <i>In</i> for inbound calls, <i>Out</i> for outbound calls)	In
custom_call_id	Unique hash call identifier taken from <i>P-Charging-Vector</i> field of SIP header; this field is being used to join the calls on phone gateway and on the bridge	icid-value=651b96b70a;icid-generated-at=192.168.1.5
custom_call_type	Custom call type (for instance, <i>CONTROLLED</i> , <i>PSTN</i> , <i>RECORDING</i> , <i>VoIP</i> )	PSTN
custom_name	Custom caller name either set from the web or IVR (PIN) or empty	John Jr.
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, 16 – Normal Call Clearing, 18 – No User Responding, 34 – No Circuit/Channel Available, 38 – Network Out-of-Order, 127 – Interworking, Unspecified, etc.) <sup>1</sup>	16
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.)	Normal
disconnect_who	Who disconnected the call (for instance, <i>USER</i> , <i>BRIDGE</i> )	USER

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

- <http://en.wikipedia.org/wiki/Q.931>
- <http://networking.ringofsaturn.com/Routers/isdncausecodes.php>

Field	Description	Data Samples
job_code	Active billing (business) code	123
node	Node name	AST1
role	Role in the conference ( <i>Host, Participant, Listener</i> )	Host
subscriber_name	First and/or last names of the subscriber or empty	John

Additionally this information is being stored in local SQL database and in CSV text file by standard billing adapters provided with the WYDE bridge software. See next section of the guide for details.

## Section 2.2: Standard Billing Adapters and Rules

The following two predefined billing adapters are included and supported by standard WYDE bridge software installation:

- *localdb* – the adapter that saves CDR information into *dnca\_calls* local database;
  - *file* – the adapter that saves CDR information into */usr/local/DNCA/log/CDR.log* file.
- Note that both these billing adapters are standard adapters and included into standard WYDE bridge software installation. These billing adapter drivers are placed in the */usr/local/DNCA/lib/Billing/Adapter* folder, so this folder contains *LocalDb.pm* and *File.pm* files for these standard billing adapters. These two adapters are being added into the list of available billing adapters during the WYDE software installation and you can see them using *billing-adapter-show* command. They could be maintained by *billing-adapter-\*\*\** commands as we will describe later in this guide.

Let's review each of these two standard billing adapters.

The standard billing adapter named *localdb* saves CDR information into local SQL database (PostgreSQL).

- ✓ *dnca\_calls* database (the database name is defined using configuration parameter: *billing\_localdb\_name*), by default the database contains CDR data for the last 180 days (this period can be redefined using configuration parameter: *billing\_localdb\_storing\_period*); if you are going to use these billing data, they should be transferred to your external database within this time period.

Additionally this *localdb* billing adapter is configured using the following system settings configuration parameters:

- *billing\_localdb\_host* – denotes the server (its IP address) where the local billing database is being placed;
- *billing\_localdb\_user* – denotes the name of the user that is used to access the local billing database;
- *billing\_localdb\_passwd* – denotes the password of the user that is used to access the local billing database;
- *billing\_localdb\_driver* – denotes the billing database driver, the default value is *pgsql* – PostgreSQL, for other possible drivers (for example MySQL or Oracle) please contact WYDE Voice technical support.

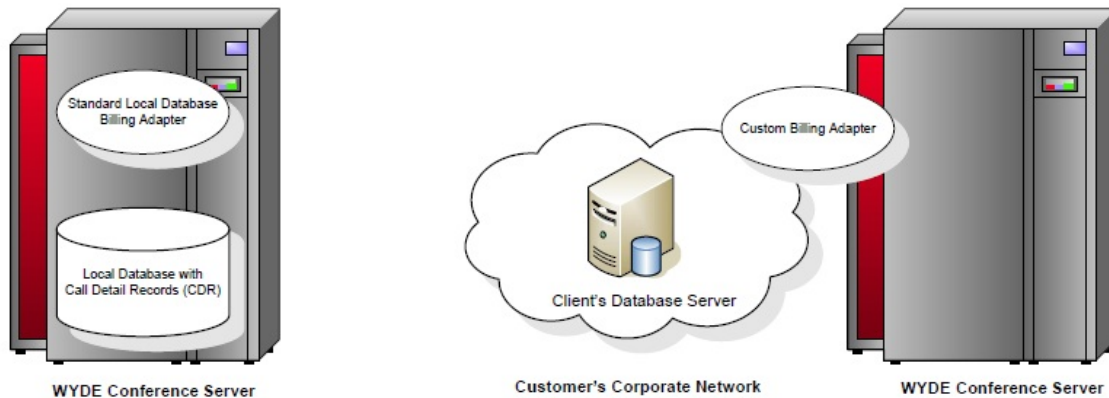
Local SQL database contains CDR information about the completed calls in the *calls* table. This table contains the data in chronological order of when the call was completed, i.e. when CDR record was created for the call. The table fields, description of the fields, and stored data samples are listed in Appendix B: CDR Data Structures, Local *dnca\_calls* Database *calls* Table Data Structure and Samples.

The *calls* table described above contains data about completed calls. The *conf\_flag* flag of the *calls* table shows if the CDR records represent the last call in the conference. Once the last call was completed in the conference, the conference data record is being stored into *conferencedr* table of local SQL database based on *calls* table data. The records in this table represent the completed conferences. This table fields, description of the fields, and stored data samples are listed in Appendix B: CDR Data Structures, Local *dnca\_calls* Database *conferencedr* Table Data Structure and Samples.

The standard billing adapter named *file* saves CDR information into local CSV 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 Appendix B: CDR Data Structures, CDR.log File Data Structure.

Of course, 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. This approach will be described in next sections of this guide.



**Figure 1: WYDE Bridge Standard and Custom Billing Adapters Architecture Samples**

There is one predefined standard billing rule with name *default* that is included and supported by standard WYDE bridge software installation. As it was previously mentioned the billing rule includes comma-separated list of billing adapters that should be used in the specific billing rule. This *default* billing rule is defined as *file,localdb* – that means that these two standard billing adapters are being consecutively used in this rule.

The billing rule name should be selected in `dnis_billingrule` (Billing rule) call flow attribute value either on call flow or on DNIS level.

### ***Section 2.3: Billing Integration***

CDRs can be delivered to the external billing system in one of the following ways:

1. External billing system pulls new CDRs from the internal database regularly in predefined intervals. In this case you use data created by standard *localdb* billing adapter in *dnca\_calls* database. 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. Using custom Billing Adapter bridge can push CDRs to the external billing system in the real-time. Usual push scenarios are:
  - i. The Billing Adapter inserts CDRs into external database.
  - ii. The Billing Adapter creates CSV files and makes them available on the FTP; the remote system takes them from FTP itself.
  - iii. The Billing Adapter sends CDRs to the external system using proprietary protocol (TCP based, UDP based, SOAP, etc).
3. 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.

### ***Section 2.4: Custom Billing Adapters and Rules***

In terms of WYDE bridge software the *Billing Adapter* is the component (function) responsible for storing billing, i.e. CDR information. Billing adapter processes information from the journal created by *MF* service; in this CDRs journal *MF* keeps information about completed calls; billing adapter receives this information, transforms it into required format and stores it in required data carrier.

In addition to standard *localdb* and *file* billing adapters that go with standard WYDE bridge software installation and that we described in previous sections of this guide you can create your own billing adapters that will process your CDR information on the fly in the real-time mode. Such custom billing adapters could store CDR information in your own database; they are 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 was previously described in Section 2.1: CDR Format.

The possibility to create the custom billing adapter is provided by the WYDE software. Custom billing adapters are routines, i.e. drivers, written in Perl that perform saving of CDR information using specific protocols and parameters used to run there routines (drivers).

The billing *Adapter* base class has been created as shown in Appendix A: Billing Adapters Code Samples, Billing Adapter Base Class (Adapter.pm). It has the following methods that

can be overridden in your custom billing adapter for the purpose of storing CDRs as required for your organization:

- `new` – the constructor of the class, that is used to parse billing adapter input parameters and for initialization purposes;
- `pollingInterval` – the interval in seconds of how often CDR information is being requested and processed by the adapter (default: *1* second);
- `onCDR` – is being called when CDR queue is not empty, the array of CDR hashes is being formed and this method is being called with this CDRs array transferred as its parameter, these CDR hashes are being processed by the custom billing adapter in this `onCDR` method; this method should return the integer actual number of CDRs that were successfully stored and processed by the adapter;
- `limitCDR` – the maximum size of the array of CDRs that is being transferred to the `onCDR` method as its parameter (default: *100* elements);
- `requireMDR` – the Boolean flag that shows should or should not the adapter process information when the last CDR in the conference has been created, i.e. the conference was dropped (MDR is information that determines if the call is the last in the conference and the conference was completed when this call ended):
  - if `requireMDR` returns *0* (default), that means that no MDR information should be processed and `onMDR` method should not be called;
  - if `requireMDR` returns *1*, that means that MDR information should be processed and stored by your custom billing adapter and `onMDR` method should be called to process such information;
- `onMDR` – is being called if `requireMDR` returns *1* and when the last CDR is being created for the conference, i.e. it is being called when the conference is over; the method as its parameters receives the array of all conference CDRs hashes that belong to one completed conference only; this method should return *1* if the conference CDRs were successfully stored and processed by the adapter or *0* if the CDRs processing was failed.

The custom billing adapter is the class inherited from this *Adapter* base class where any of the methods above could be overridden in order to maintain the logic required by your organization. When you write your custom billing adapter you should inherit it from the base *Adapter* class:

```
@ISA = ("Billing::Adapter");
```

and override the methods that you are going to use to store CDR as required for your organization.

Additionally your custom billing adapter should contain the method:

- `factory` – creates and returns the instance of your billing adapter class; the source code of this method could be the following:
 

```
sub factory {
    return new Billing::Adapter::<your_billing_adapter_name>(@_);
}
```

Note 1. It is up to you to decide if your organization wants to save CDRs using `onCDR` method, or `onMDR` method, or both.

Note 2. If `onCDR` or `onMDR` methods reported an error – the framework considers that these CDRs are not handled and tries to call and process them during the next call of this method.

Note 3. Because CDR and MDR information is being kept in internal billing journal, if your billing adapter was not working certain amount of time the billing information is not being lost; when your billing adapter starts working again all accumulated billing information will be transferred to your billing adapter in chronological order once it is available.

These billing adapter 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.

Billing rules are being used for the conference billing information storing. 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.

`{adapter1[,adapter2[,adapter3[,...]]]}`:

- *file* – denotes that CDR information should be stored into `/usr/local/DNCA/log/CDR.log` file using the billing adapter: *file*;
- *localdb* – denotes that CDR information should be stored into *dnca\_calls* local database using the billing adapter: *localdb*;
- `<adapter_name>[,...]` – denotes any other custom billing adapters comma-separated names that should be used to store CDR information.

For example billing rules could be defined as *file,localdb* or *file,localdb,myAdapter*.

As it was previously mentioned the billing rule name used for your custom billing adapter should be selected in `dnis_billingrule` (Billing rule) call flow attribute value either on call flow or on DNIS level.

## ***Section 2.5: WYDE Commands to Manage Billing Adapters and Rules***

Billing adapters and rules 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 billing adapters and billing rules.

### **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:



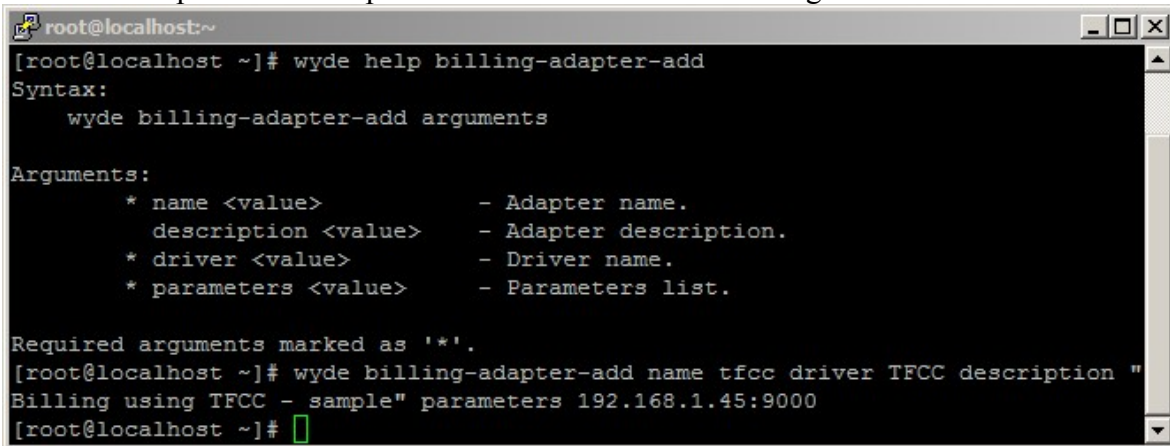
- `name <value>` – The name of the billing adapter that should be added. This name should be unique, i.e. there should no be any other billing adapter with the same name on the bridge.
- `description <value>` – The optional description of the billing adapter that should be added.
- `driver <value>` – 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.
- `parameters <value>` – The list of parameters for the billing adapter that should be added.

Arguments `name`, `driver` 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*, 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"
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 2.



```
root@localhost:~
[root@localhost ~]# wyde help billing-adapter-add
Syntax:
    wyde billing-adapter-add arguments

Arguments:
    * name <value>           - Adapter name.
    description <value>     - Adapter description.
    * driver <value>        - Driver name.
    * parameters <value>   - Parameters list.

Required arguments marked as '*'.
[root@localhost ~]# wyde billing-adapter-add name tfcc driver TFCC description "
Billing using TFCC - sample" parameters 192.168.1.45:9000
[root@localhost ~]#
```

Figure 2: *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 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:

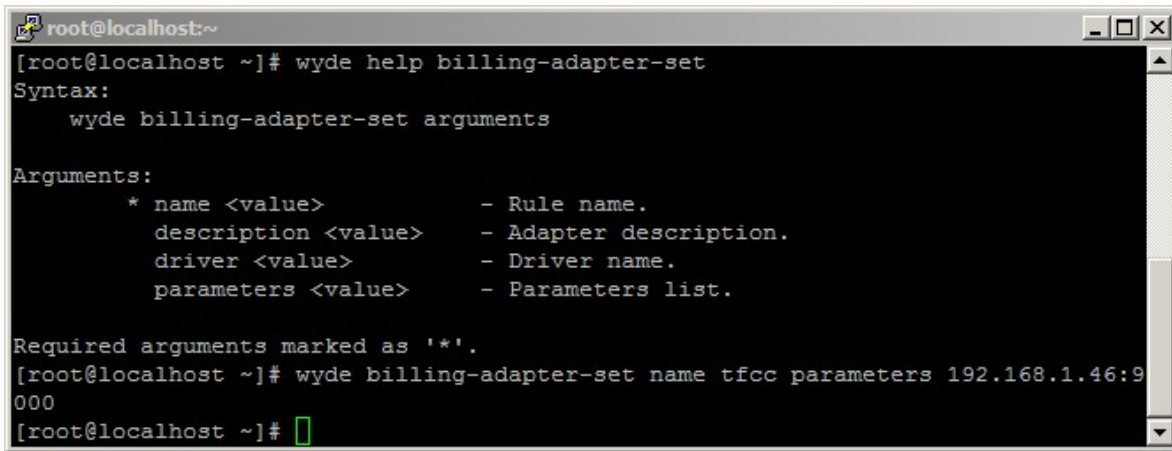
- name <value> – The name of the billing adapter that should be changed.
- description <value> – New description of the billing adapter that should be set.
- driver <value> – New driver name for the billing adapter that should be set.
- parameters <value> – New list of parameters for the billing adapter 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 3.



```

root@localhost:~
[root@localhost ~]# wyde help billing-adapter-set
Syntax:
    wyde billing-adapter-set arguments

Arguments:
    * name <value>          - Rule name.
    description <value>     - Adapter description.
    driver <value>          - Driver name.
    parameters <value>     - Parameters list.

Required arguments marked as '*'.
[root@localhost ~]# wyde billing-adapter-set name tfcc parameters 192.168.1.46:9000
[root@localhost ~]#

```

Figure 3: *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 4. 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, parameters and description.



```

root@ZILBER:~
[root@ZILBER ~]# wyde billing-adapter-show
Name      Driver      Parameters      Description
-----
file      File        file            Stores CDRs into the text files in the logs directory
localdb   LocalDb     localdb         Stores CDRs into the local billing database.
tfcc      TFCC        192.168.1.45:9000 Billing using TFCC - sample
[root@ZILBER ~]#

```

Figure 4: *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: {adapter1[, adapter2[, adapter3[, ...]]}:
  - file – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file using the billing adapter: file;

- o `localdb` – denotes that CDR information should be stored into *dnca\_calls* local database using the billing adapter: `localdb`;
- o `<adapter_name>[, ...]` – denotes any other custom billing adapters comma-separated names that should be used to store CDR information.

For example rule argument could be defined as *file,localdb* or *file,localdb,myAdapter*.

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,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 5.

```
root@ZILBER:~
[root@ZILBER ~]# wyde help billing-rule-add
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,tfcc"
[root@ZILBER ~]#
```

Figure 5: *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: {adapter1[, adapter2[, adapter3[, ...]]}:
  - file – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file using the billing adapter: *file*;
  - localdb – denotes that CDR information should be stored into *dnca\_calls* local database e using the billing adapter: *localdb*;
  - <adapter\_name>[, ...] – denotes any other custom billing adapters comma-separated names that should be used to store CDR information.

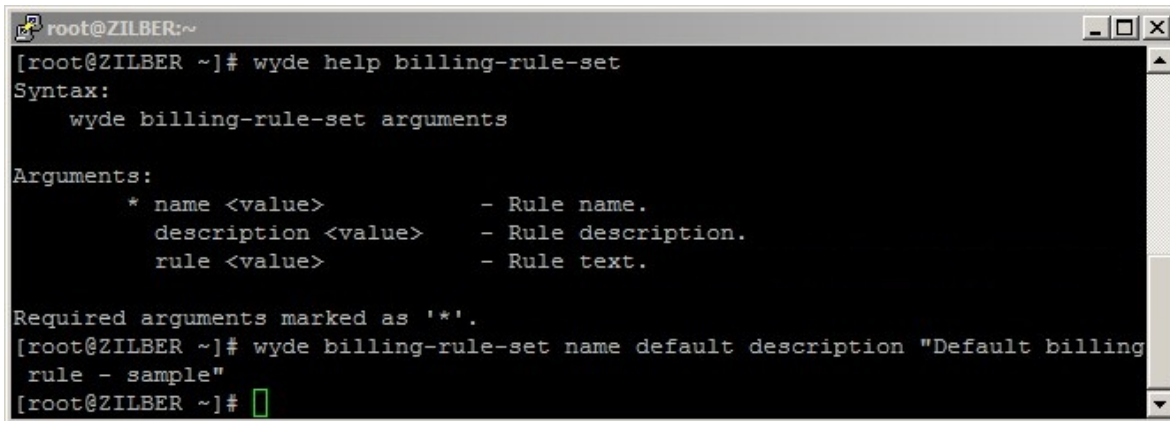
For example rule argument could be defined as *file, localdb* or *file, localdb, myAdapter*.

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 6.



```

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.

Required arguments marked as '*'.
[root@ZILBER ~]# wyde billing-rule-set name default description "Default billing
rule - sample"
[root@ZILBER ~]#

```

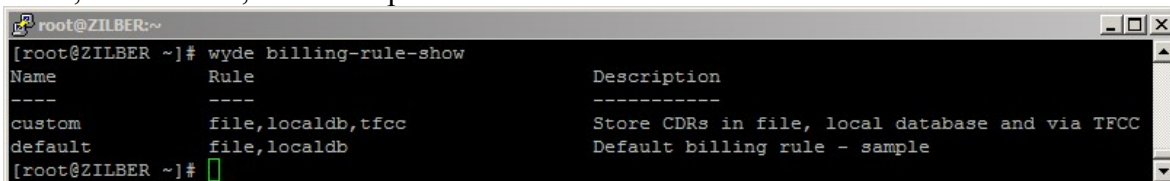
Figure 6: *wyde help billing-rule-set* and *wyde billing-rule-set* Commands Output Sample

## 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 7. 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,tfcc  Store CDRs in file, local database and via TFCC
default   file,localdb      Default billing rule - sample
[root@ZILBER ~]#

```

Figure 7: *wyde billing-rule-show* Command Output Sample

## Billing Configurations Reloading

You should reload your WYDE bridge billing configuration if you created new or updated existing billing adapter driver (\*.pm file) as well as if you added or updated billing adapter or rule definition using *wyde* commands with options: *billing-adapter-add*, *billing-adapter-set*, *billing-rule-add*, *billing-rule-set*.

To do so you should run the *wyde* command line utility with the *billing-reload* option. The syntax is as follows:

```
wyde billing-reload
```

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

## Chapter 3: 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 could have any of the following methods:

- `new` – that performs class initialization, for instance database initialization, socket initialization, etc.;
- `pollingInterval` – that determines the interval in seconds of how often CDR information is being requested and processed by the adapter;
- `onCDR` – that is being called when there are non-processed completed calls 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.;
- `limitCDR` – that determines the maximum size of the CDRs array that is being transferred to the `onCDR` method as its parameter;
- `requireMDR` – that determines should (if returns value `1`) or should not (if returns value `0`) the adapter process information when the conference is completed by `onMDR` method;
- `onMDR` – is being called if `requireMDR` returns `1` and when the last CDR is being created for the conference, i.e. when there are non-processed completed conferences; here you can also implement data saving that is necessary for your organization, but this method always processes the all calls that belong to the single conference only.

### Section 3.1: Sample of Calls Billing Adapter to Text File

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 specified text file.

Let's assume that the full file name is `/root/billing.csv` and it is being transferred for the billing adapter as its parameter.

[Click here to see sample of the billing adapter TEXTCSV source code that we developed to implement this request.](#)

### Sample of WYDE Bridge Configuration for Calls Text File Billing Adapter

When design of `TEXTCSV.pm` file is completed you should copy this file into `/usr/local/DNCA/lib/Billing/Adapter` folder and then you should run the `wyde` command line utility with the `billing-reload` option:

```
wyde billing-reload
```

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 textcsv driver TEXTCSV
  description "Billing to Calls Text File"
  parameters "/root/billing.csv"
```



```
wyde billing-rule-add name customfile description "Store CDRs
in file, local database and in custom text file" rule
"file,localdb,textcsv"
```

The billing rule *customfile* defines that the CDR data should be stored into CDR file, local database and using *textcsv* adapter into specified file.

Also note that after you add the billing adapter you also should run the *wyde* command line utility with the *billing-reload* option:

```
wyde billing-reload
```

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 *customfile*.

As soon as this has been done 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 custom text file. Here we provide few samples of your stored data:

```
subscriber_name=,conf_flag=0,calling_number=101,custom_call_type=PSTN,disconnect_reason=Normal,callflow=SPECTEL,node=AST1,conf_number=666011,call_dropped=2011-07-19
12:03:10,call_created=2011-07-19
11:53:10,job_code=,called_number=8665080012,custom_name=,audio_key=0,bridge=WYDE,call_id=16777555,call_duration=600,addr_to="8665080012"
<sip:8665080012@192.168.1.31>,access_code=6613,conf_id=134217884,disconnect_cause=16,connection_type=In,role=Listener,disconnect_who=USER,addr_from=<sip:101@127.0.0.1>
subscriber_name=,conf_flag=2,calling_number=100,custom_call_type=PSTN,disconnect_reason=Normal,callflow=SPECTEL,node=AST1,conf_number=666011,call_dropped=2011-07-19
12:03:11,call_created=2011-07-19
11:53:11,job_code=,called_number=8665080012,custom_name=,audio_key=0,bridge=WYDE,call_id=16777556,call_duration=600,addr_to="8665080012"
<sip:8665080012@192.168.1.31>,access_code=6611,conf_id=134217884,disconnect_cause=16,connection_type=In,role=Host,disconnect_who=USER,addr_from=<sip:100@127.0.0.1>
```

### Section 3.2: Sample of Conferences Billing Adapter to Text File

Let's review the following scenario:

- we need to create and configure the billing adapter that for all completed conferences will write the conference number, the conference identifier, the information when the conference was started, when it was ended, and how many calls were in the conference;
- this information should be written into specified comma-delimited text file.

Let's assume that the full file name is */root/billconf.csv* and it is being transferred for the billing adapter as its parameter.

[Click here to see sample of the billing adapter TEXTCONF source code that we developed to implement this request.](#)

Because the data need to be saved for entire conference, the *sub requireMDR* should be overridden and it should return *1*; the specific data saving mechanism is implemented in *sub onMDR* method that is being called when the conference is over; as its parameter it receives all conference CDRs data that are being processed and stored according to your needs (see sample in this guide appendix).



## Sample of WYDE Bridge Configuration for Conferences Text File Billing Adapter

When design of *TEXTCONF.pm* file is completed you should copy this file into */usr/local/DNCA/lib/Billing/Adapter* folder and then you should run the *wyde* command line utility with the *billing-reload* option:

```
wyde billing-reload
```

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 textconf driver TEXTCONF
    description "Billing to Conferences Text File"
    parameters "/root/billconf.csv"
wyde billing-rule-add name customfileconf description "Store
    CDRs in file, local database and in custom conf text
    file" rule "file,localdb,textconf"
```

The billing rule *customfileconf* defines that the CDR data should be stored into CDR file, local database and using *textconf* adapter into specified file.

Also note that after you add the billing adapter you also should run the *wyde* command line utility with the *billing-reload* option:

```
wyde billing-reload
```

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 *customfileconf*.

As soon as this has been done when the conferences are ended, their MDR information will be stored not only in traditional locations (CDR file and local database), but also in your custom text file. Here we provide few samples of your stored data:

```
666001,134217893,2011-07-19 18:59:34+03,2011-07-19 19:02:34+03,3
666011,134217894,2011-07-19 18:59:34+03,2011-07-19 19:09:34+03,2
666011,134217897,2011-07-19 19:11:34+03,2011-07-19 19:21:34+03,2
666001,134217896,2011-07-19 19:11:34+03,2011-07-19 19:21:34+03,3
666011,134217899,2011-07-19 19:23:34+03,2011-07-19 19:33:34+03,2
666001,134217898,2011-07-19 19:23:35+03,2011-07-19 19:33:34+03,3
```

## Section 3.3: Sample of Billing Adapter to Windows PostgreSQL Database

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.](#)

### PostgreSQL Database Access Configuration Sample

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 onCDR* method; it includes data formatting according to your needs and insert statement design and execution (see sample in this guide appendix).

### Sample of WYDE Bridge Configuration for PostgreSQL Billing Adapter

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 run the *wyde* command line utility with the *billing-reload* option:

```
wyde billing-reload
```

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"
    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,winpgsql"
```

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 run the *wyde* command line utility with the *billing-reload* option:

```
wyde billing-reload
```

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=101,custom_call_type=PSTN,disconnect_reason=Normal,callflow=SPECTEL,node=AST1,conf_number=666001,call_dropped=2011-07-19
16:26:07,call_created=2011-07-19
16:16:07,job_code=,called_number=8665080012,custom_name=,audio_key=0,bridge=WYDE,call_id=16777567,call_duration=600,addr_to="8665080012"
<sip:8665080012@192.168.1.31>,access_code=6602,conf_id=134217890,disconnect_cause=16,connection_type=In,role=Participant,disconnect_who=USER,addr_from=<sip:101@127.0.0.1>"
"subscriber_name=,conf_flag=2,calling_number=102,custom_call_type=PSTN,disconnect_reason=Normal,callflow=SPECTEL,node=AST1,conf_number=666001,call_dropped=2011-07-19
16:26:07,call_created=2011-07-19
16:16:07,job_code=,called_number=8665080012,custom_name=,audio_key=0,bridge=WYDE,call_id=16777568,call_duration=600,addr_to="8665080012"
<sip:8665080012@192.168.1.31>,access_code=6602,conf_id=134217890,disconnect_cause=16,connection_type=In,role=Participant,disconnect_who=USER,addr_from=<sip:102@127.0.0.1>"
```

### Section 3.4: Sample of Billing Adapter to Windows Microsoft SQL Database

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*.

### Microsoft SQL Server Installation and Configuration Sample

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.](#)

### Microsoft SQL Database Access Configuration Sample

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 onCDR* 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.

### Sample of WYDE Bridge Configuration for Microsoft SQL Billing Adapter

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 run the *wyde* command line utility with the *billing-reload* option 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"
    parameters MSSQL
wyde billing-rule-add name winms description "Store CDRs in
    file, local database and in Windows Microsoft SQL
    database" rule "file,localdb,mssql"
```

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 run the *wyde* command line utility with the *billing-reload* option:

```
wyde billing-reload
```

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.

## Chapter 4: wyde Billing Command Reference

### **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 (\*);  
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;  
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-reload (Reload Billing Configuration)**

*Syntax:*

```
wyde billing-reload
```

### **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: {adapter1[, adapter2[, adapter3[, ...]]]}:

- o **file** – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file using the billing adapter: *file*;
- o **localdb** – denotes that CDR information should be stored into *dnca\_calls* local database using the billing adapter: *localdb*;
- o <adapter\_name>[, ...] – denotes any other custom billing adapters comma-separated names that should be used to store CDR information.

For example rule argument could be defined as *file, localdb* or *file, localdb, myAdapter*.

### **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: {adapter1[, adapter2[, adapter3[, ...]]]}:

- o **file** – denotes that CDR information should be stored into */usr/local/DNCA/log/CDR.log* file using the billing adapter: *file*;
- o **localdb** – denotes that CDR information should be stored into *dnca\_calls* local database using the billing adapter: *localdb*;
- o <adapter\_name>[, ...] – denotes any other custom billing adapters comma-separated names that should be used to store CDR information.

For example rule argument could be defined as *file, localdb* or *file, localdb, myAdapter*.

### **billing-rule-show (Show Billing Rules)**

*Syntax:*

`wyde billing-rule-show`

## Appendix A: Billing Adapters Code Samples

### *Billing Adapter Base Class (Adapter.pm)*

```
package Billing::Adapter;

sub new {
    my $self = {};
    my $class = shift;

    return bless($self, $class);
}

sub pollingInterval {
    return 1; #1 sec
}

sub limitCDR {
    return 100;
}

sub onCDR {
    my ($self, $cdrs) = @_;
    return scalar(@$cdrs);
}

sub requireMDR {
    return 0;
}

sub onMDR {
    return 0;
}
```

### ***Sample of Calls Billing Adapter for Text File (TEXTCSV)***

```

package Billing::Adapter::TEXTCSV;

use IO::File;
use Misc::Logger;
use Billing::Adapter;
@ISA = ("Billing::Adapter");

sub factory {
    return new Billing::Adapter::TEXTCSV(@_);
}

sub new {
    my $self = {};
    my $class = shift;
    my $object = bless($self, $class);
    my $parameters = shift;

    $logger->info("Create adapter TEXTCSV : parameters=$parameters");

    $self->{FILE} = new IO::File;
    $self->{FILE}->open(">> $parameters");

    return $object;
}

sub onCDR {
    my $self = shift;
    my $cdrs = shift;
    my $sent = 0;

    foreach my $cdr (@$cdrs) {
        my @data_array = ();
        foreach my $k (keys(%$cdr)) {
            push(@data_array, $k."=".$cdr->{$k});
        }
        my $data_str = join(',', @data_array);

        $self->{FILE}->syswrite($data_str."\n");
        $sent++;
    }

    return $sent;
}

```



### *Sample of Conferences Billing Adapter for Text File (TEXTCONF)*

```

package Billing::Adapter::TEXTCONF;

use IO::File;
use Misc::Logger;
use Billing::Adapter;
@ISA = ("Billing::Adapter");

sub factory {
    return new Billing::Adapter::TEXTCONF(@_);
}

sub new {
    my $self = {};
    my $class = shift;
    my $object = bless($self, $class);
    my $parameters = shift;

    $logger->info("Create adapter TEXTCONF : parameters=$parameters");

    $self->{FILE} = new IO::File;
    $self->{FILE}->open(">> $parameters");

    return $object;
}

sub requireMDR {
    return 1;
}

sub onMDR {
    my $self = shift;
    my $cdrs = shift;
    my $data_str;
    my $conf_created;
    my $conf_dropped;
    my $calls = 0;
    my $isFirst = 1;

    foreach my $cdr (@$cdrs) {
        if ($isFirst==1) {
            $data_str = "$cdr->{conf_number},$cdr->{conf_id},";
            $conf_created = $cdr->{call_created};
            $conf_dropped = $cdr->{call_dropped};
            $isFirst = 0;
        }

        if ($conf_created lt $cdr->{call_created}) {
            $conf_created = $cdr->{call_created};
        }
        if ($conf_dropped gt $cdr->{call_dropped}) {
            $conf_dropped = $cdr->{call_dropped};
        }
        $calls++;
    }
    $data_str .= "$conf_created,$conf_dropped,$calls";

    $self->{FILE}->syswrite($data_str."\n");
    return 1;
}

```

## ***Sample of Billing Adapter for Windows PostgreSQL Database (WINPGSQL)***

```
package Billing::Adapter::WINPGSQL;

use Misc::Logger;
use Billing::Adapter;
use DBI;
@ISA = ("Billing::Adapter");

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 $host = shift;

    $logger->info("Create adapter WINPGSQL : parameters=$host");

    $self->{db} = DBI->connect("dbi:Pg:dbname=$database;host=$host", $user, $password)
        || proc_error("Connect: ".DBI::errstr);

    return $object;
}

sub onCDR {
    my $self = shift;
    my $cdrs = shift;
    my $sent = 0;
    my $query;
    my $sth;

    foreach my $cdr (@$cdrs) {
        $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");
        $sent++;
    }

    return $sent;
}
```

### *Sample of Billing Adapter for Microsoft SQL Database (MSSQL)*

```

package Billing::Adapter::MSSQL;

use Misc::Logger;
use Billing::Adapter;
use DBI;
use Misc::Database;
@ISA = ("Billing::Adapter");

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 $host = shift;

    $logger->info("Create adapter WINPGSQL : parameters=$host");

    $self->{db} = DBI->connect("dbi:Sybase:server=$host:database=$database", $user,
        $password) || die("Connect: ".DBI::errstr."\n");

    return $object;
}

sub onCDR {
    my $self = shift;
    my $cdrs = shift;
    my $sent = 0;
    my $query;
    my $sth;

    foreach my $cdr (@$cdrs) {
        $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");
        $sent++;
    }

    return $sent;
}

```

## Appendix B: CDR Data Structures

### *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 ( <i>In</i> for inbound calls, <i>Out</i> 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, <i>USER</i> , <i>BRIDGE</i> )
The reason why the call was disconnected (for instance <i>Normal</i> , <i>Error</i> )
Call flow name
Access code used
Role in the conference ( <i>Host</i> , <i>Participant</i> , <i>Listener</i> )
Custom call type (for instance, <i>CONTROLLED</i> , <i>PSTN</i> , <i>RECORDING</i> , <i>VoIP</i> )
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

***Local dnca\_calls Database calls Table Data Structure and Samples***

Field	Description	Data Samples
id	Internal serial identifier of <i>calls</i> table	Integer call identifier (counter)
bridge	Bridge name	WYDE5
node	Node name	AST1
call_id	Call identifier	Integer call identifier
connection_type	Connection type, i.e. call direction ( <i>In</i> for inbound calls, <i>Out</i> for outbound calls)	In
calling_number	Incoming calling number, i.e. the number from which called the caller or empty	4024684432
called_number	Called number, i.e. the number to which the caller had called	8665080020
addr_to	Full address TO, i.e. full qualified SIP URI of callee's address	"8665080020" <sip:8665080020@192.168.1.5>
addr_from	Full address FROM, i.e. full qualified SIP URI of caller's address	<sip:4024684432@192.168.1.5>
call_created	Date and time when the call was created (started)	2010-11-09 17:39:07+02
call_dropped	Date and time when the call was dropped (ended)	2010-11-09 17:55:12+02
duration	Duration of the call in seconds	965
disconnect_who	Who disconnected the call (for instance, <i>USER</i> , <i>BRIDGE</i> )	USER
disconnect_cause	Standard Q.931 (ISDN) Disconnect Cause Codes; see detail description for this field above in Table 1	16
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.)	Normal
conf_id	Conference identifier	Integer conference identifier
access_code	Access code used	505052
callflow	Call flow name (for instance, <i>CONF</i> , <i>PLAYBACK</i> , <i>OPERATOR</i> , <i>SPECTEL</i> , etc.)	SPECTEL
conf_number	Conference number	889900
conf_joined	Date and time when the call was joined to the conference	2010-11-09 17:39:13+02
conf_rejected	Date and time when the call was disconnected from the conference	2010-11-09 17:55:12+02
conf_duration	Total duration in seconds of how long the call was in joined to the conference	959
conf_mode	Role in the conference ( <i>Host</i> , <i>Participant</i> , <i>Listener</i> )	Host
custom_name	Custom caller name either set from the web or IVR (PIN) or empty	John Jr.
subscriber_name	Name of the subscriber assigned by this call or empty	John
custom_call_type	Custom call type or empty (for instance, <i>CONTROLLED</i> , <i>PSTN</i> , <i>RECORDING</i> , <i>VoIP</i> , etc.)	PSTN

Field	Description	Data Samples
custom_call_id	Unique hash call identifier taken from <i>P-Charging-Vector</i> field of SIP header; this field is being used to join the calls on phone gateway and on the bridge	icid-value=651b96b70a;icid-generated-at=192.168.1.5
audio_key	Audio key assigned to this call or empty	690
job_code	Active billing (business) code or empty if job cod was not defined	123
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	2

***Local dnca\_calls Database conferencedr Table Data Structure and Samples***

<b>Field</b>	<b>Description</b>	<b>Data Samples</b>
conferenceid	Conference identifier	Integer conference identifier
number	Conference number	889900
created	Date and time when the conference was created, i.e. the first caller arrived	2010-11-09 17:36:33+02
closed	Date and time when the conference was closed (ended)	2010-11-09 17:55:12+02
duration	Conference duration in seconds, number of seconds which have elapsed since the conference was created till the time when it was terminated	1119
confduration	Total duration in seconds of all calls that were joined to the conference, i.e. sum of all conference calls durations	1729
recduration	Conference recording duration in seconds – 0 (if there was no recording in the conference) or total recording duration	252
totalcount	Total number of conference calls (0, 1, 2, 3, etc.)	2
reccount	Number of the conference recording – 0 (if there was no recording in the conference), or 1, 2, 3, etc.	1
modcount	Number of hosts (moderators) that were joined to the conference (0, 1, 2, 3, etc.)	1
cdrid	Internal serial identifier of <i>conferencedr</i> table	Integer counter

## Appendix C: Definitions, Acronyms and Abbreviations

While we discussed the WYDE Bridge Billing 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.
- **Billing Adapter** – A component (function) responsible for storing billing, i.e. CDR information. Billing adapter processes information from the file created by *MF* service; in this file *MF* keeps information about completed calls; billing adapter receives this information, transforms it into required format and stores it in required data carrier.
- **Billing Rule** – A rule that is used to determine the specific billing adapters that are used to save CDR information. The billing rule could be defined either on call flow level or on DNIS level.

## **Appendix D: 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](mailto:support@wydevoice.com). We will respond to your question within 24 hours.