CGRateS Documentation

Release 0.9.1~rc8

Dan Christian Bogos

Dec 23, 2018

Contents

Full contents:

1.1 1. Introduction

CGRateS is a *very fast* and *easily scalable* **(charging, rating, accounting, lcr, mediation, billing, authorization)** *ENGINE* targeted especially for ISPs and Telecom Operators.

It is written in Go programming language and is accessible from any programming language via JSON RPC. The code is well documented (**go doc** compliant API docs) and heavily tested. (also **1300+** tests are part of the build system).

After testing various databases like Kyoto Cabinet, Apache Cassandra, Redis and MongoDB, the project focused on **Redis** as it delivers the best trade-off between speed, configuration and scalability.

Important: MongoDB **full** support is now added.

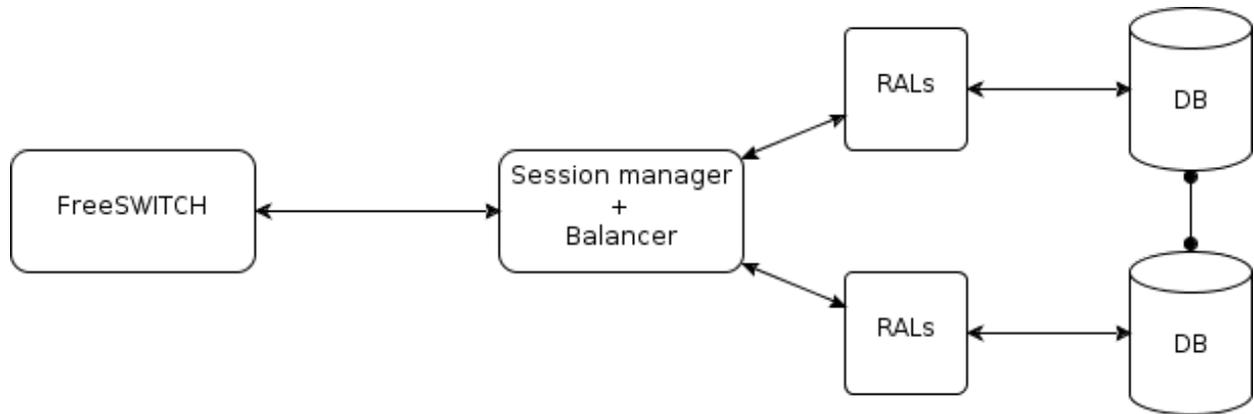
Thanks to CGRateS flexibility, connection to any database can be easily integrated by writing a simple adapter.

To better understand the CGRateS architecture, below are some logical configurations in which CGRateS can operate:

Note: RALs - is a CGRateS component and stands for RatingAccountingLCR service.
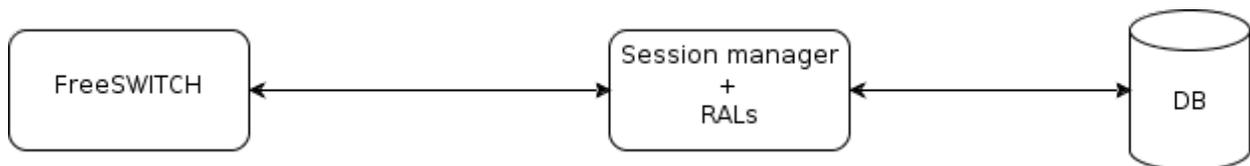
This scenario fits most of the simple installations.

While the network grows more **RALs** can be thrown into the stack to offer more requests per seconds workload. This implies the usage of the **Balancer** to distribute the requests to the **RALs** running on the *different machines*.
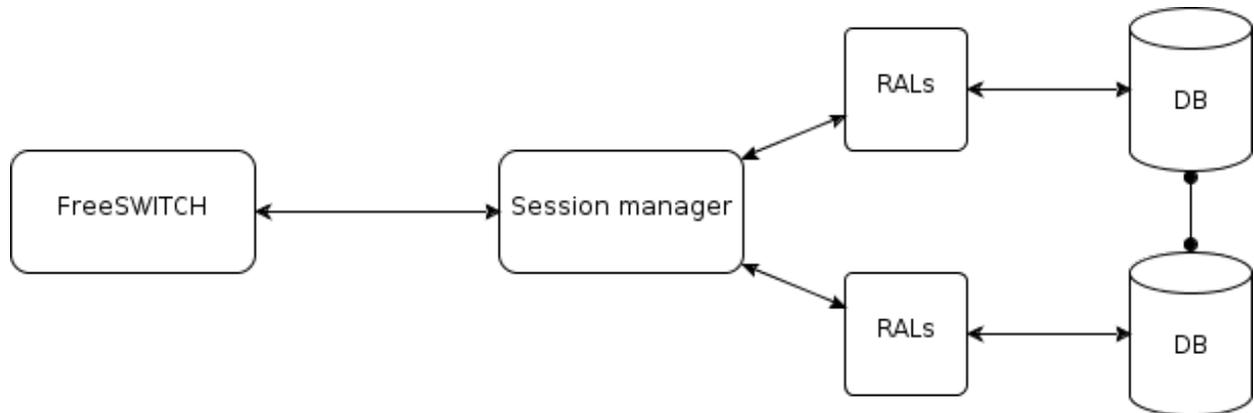
Without Balancer using HA (broadcast) . . . .
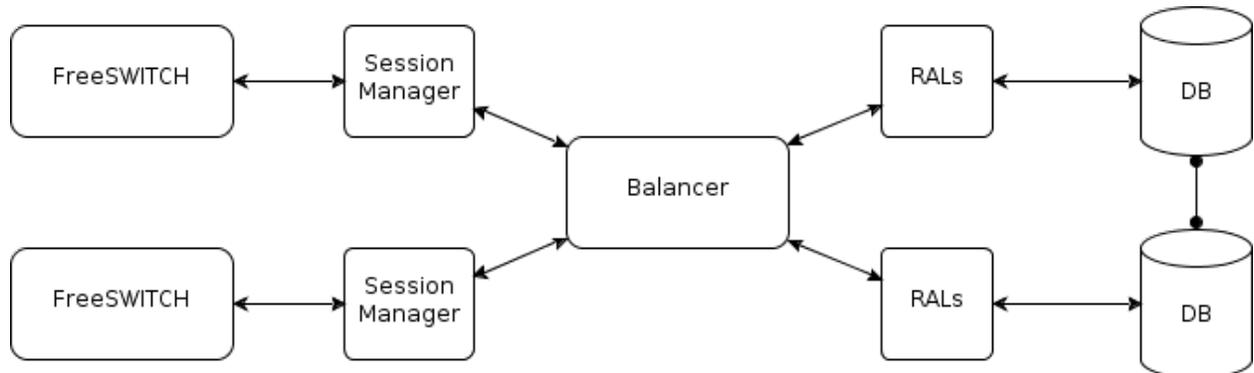
Of course more **SessionManagers** can serve *multiple Telecom Switches* and all of them are connected to the same **Balancer**.

Without Balancer using HA (broadcast) . . . .

Note: We are planning to support **multiple** *Balancers* for huge networks if the need arises.

1.1.1 1.1. CGRateS Features

- **Reliable and Fast (very fast ;)). To get an idea about speed, we have benchmarked 13000+ req/sec on a rather modest ma**

    - Using most modern programming concepts like multiprocessor support, asynchronous code execution within microthreads.

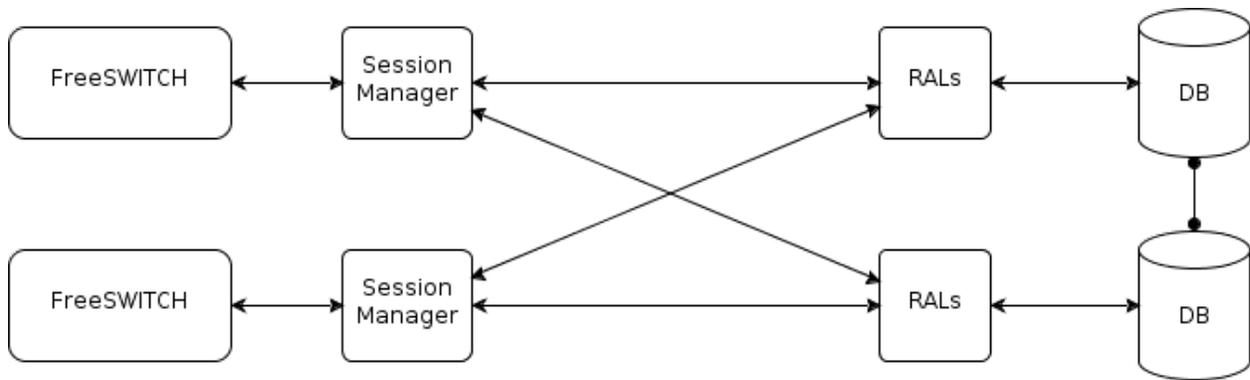   - Built-in data caching system per call duration.

    - In-Memory database with persistence over restarts.

    - Use of Linux enterprise ready tools to assure High-Availability of the Balancer where that is required (*Supervise* for Application level availability and *LinuxHA* for Host level availability).

    - High-Availability of main components is now part of CGRateS core.

- **Modular architecture**

    - Easy to enhance functionality by writing custom session managers or mediators.

    - Flexible API accessible via both **Gob** (Golang specific, increased performance) or **JSON** (platform independent, universally accessible).

- **Prepaid, Postpaid and Pseudo-Prepaid Controller.**

    - Mutiple Primary Balances per Account (eg: MONETARY, SMS, INTERNET_MINUTES, INTER-NET_TRAFFIC).

    - Multiple Auxiliary Balances per Account (eg: Free Minutes per Destination, Volume Rates, Volume Discounts).

    - Concurrent sessions per account sharing the same balance with configurable debit interval (starting with 1 second).

    - Built-in Task-Scheduler supporting both one-time as well as recurrent actions (eg: TOPUP_MINUTES_PER_DESTINATION, DEBIT_MONETARY, RESET_BALANCE).

    - ActionTriggers (useful for commercial offerings like receive amounts of monetary units if a specified number of minutes was charged in a month).

- **Highly configurable Rating.**

- – Connect Fees.

- – Priced Units definition.

- – Rate increments.

- – Millisecond timestaps.

- – Four decimal currencies.

- – Multiple TypeOfRecord rating (eg: standard vs. premium calls, SMSes, Internet Traffic).

- – Rating subject concatenations for combined records (eg: location based rating for same user).

- – Recurrent rates definition (per year, month, day, dayOfWeek, time).

- – Rating Profiles activation times (eg: rates becoming active at specific time in the future).

- Multi-Tenant for both Prepaid as well as Rating.

- Flexible Mediator able to run multiple mediation processes on the same CDR.

- Verbose action logging in persistent databases (eg: MongoDB/PostgreSQL/MySQL) to cope with country specific law requirements.

- Good documentation (that's me :).

- **"Free as in Beer"** with commercial support available on-demand.

1.1.2 1.2. Links

- CGRateS quick overview overview-main

- CGRateS home page http://www.cgrates.org

- Documentation http://cgrates.readthedocs.io

- API docs https://godoc.org/github.com/cgrates/cgrates/apier

- Source code https://github.com/cgrates/cgrates

- Travis CI https://travis-ci.org/cgrates/cgrates

- Google group https://groups.google.com/forum/#!forum/cgrates

- IRC irc.freenode.net #cgrates

1.1.3 1.3. License

CGRateS is released under the terms of the [GNU GENERAL PUBLIC LICENSE Version 3]. See **LICENSE.txt** file for details.

1.2 2. Architecture

The CGRateS suite consists of **five** software applications described below.

- cgr-engine
- cgr-loader
- cgr-console
- cgr-tester
- cgr-migrator

CGRateS has an internal cache.

```
"internal_cache" - cache
```

Operates with different external databases mentioned below.

```
"data_db"        - MongoDB, Redis
"stor_db"        - MongoDB, MySQL, PostgreSQL
```

- **data_db** - used to store runtime data (eg: accounts)
- **stor_db** - used to store offline tariff plan(s) and CDRs
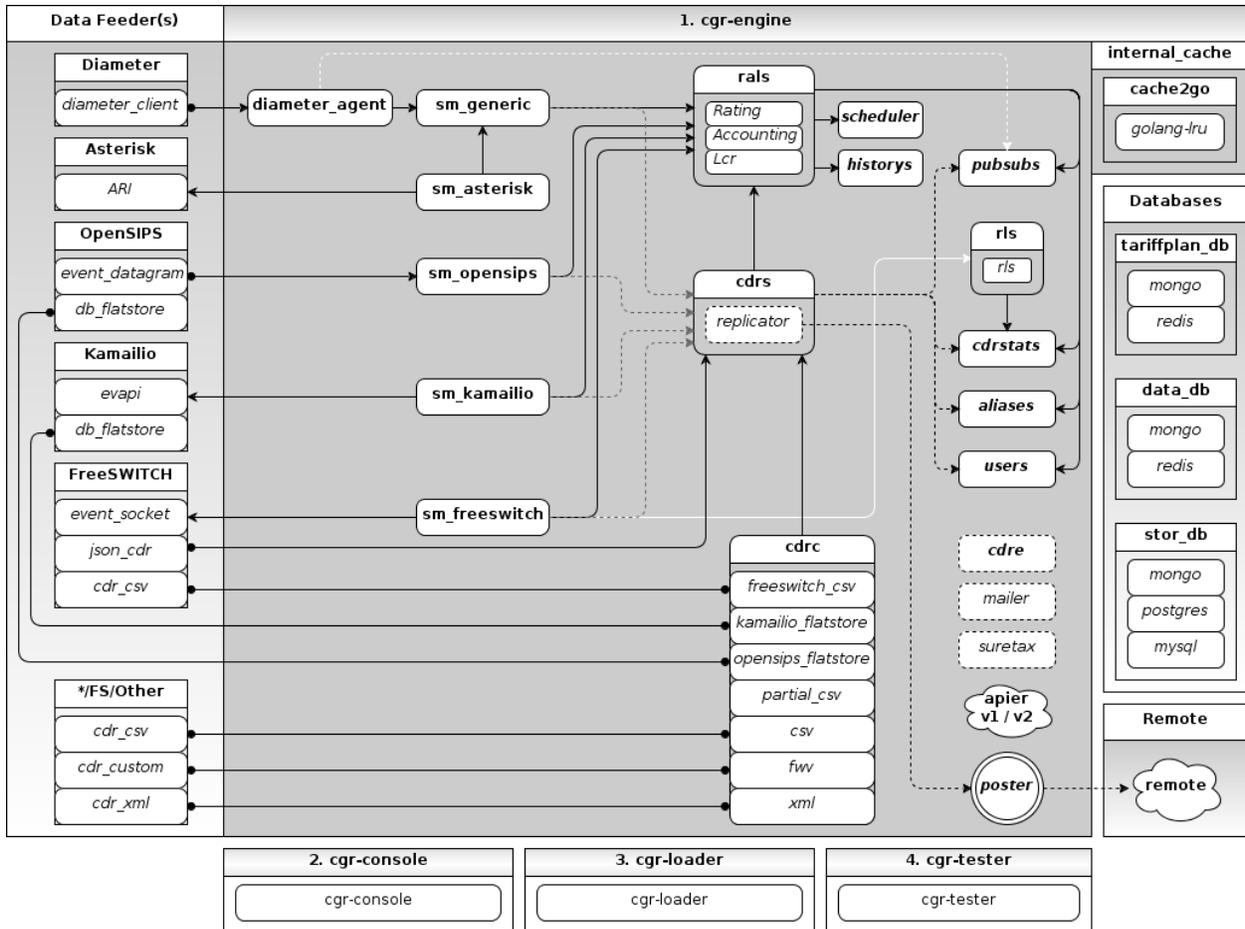

Fig. 1: CGRateS high level design

1.2.1 2.1. cgr-engine

Is the most important and complex component. Customisable through the use of *json* configuration file(s), it will start on demand **one or more** service(s), outlined below.

```
cgrates@OCS:~$ cgr-engine -help
Usage of cgr-engine:
  -cdrs
        Enforce starting of the cdrs daemon overwriting config
```

(continues on next page)

```
-config_dir string
      Configuration directory path. (default "/etc/cgrates/")
-cpuprofile string
      write cpu profile to file
-pid string
      Write pid file
-rater
      Enforce starting of the rater daemon overwriting config
-scheduler
      Enforce starting of the scheduler daemon .overwriting config
-scheduled_shutdown string
      shutdown the engine after this duration
-singlecpu
      Run on single CPU core
-version
      Prints the application version.
```

Hint: # cgr-engine -config_dir=/etc/cgrates

2.1.1. RALs service

Responsible with the following tasks:

- Operates on balances.

- Computes prices for rating subjects.

- Monitors and executes triggers.

- LCR functionality

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db"      – (dataDb)
"stor_db"      – (cdrDb, loadDb)
```

- **Config section in the CGRateS configuration file:**

    - "rals": {...}

2.1.2. Scheduler service

Used to execute periodic/scheduled tasks.

- **Communicates via:**

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db" - (dataDb)
```

- **Config section in the CGRateS configuration file:**

    - `"scheduler": {...}`

2.1.3. SessionManager service

Responsible with call control on the Telecommunication Switch side. Operates in two different modes (per call or globally):

- **PREPAID**

    - Monitors call start.

    - Checks balance availability for the call.

    - Enforces global timer for a call at call-start.

    - Executes routing commands for the call where that is necessary (eg call un-park in case of FreeSWITCH).

    - Periodically executes balance debits on call at the beginning of debit interval.

    - Enforce call disconnection on insufficient balance.

    - Refunds the balance taken in advance at the call stop.

- **POSTPAID**

    - Executes balance debit on call-stop.

All call actions are logged into CGRateS's LogDB.

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"stor_db" - (cdrDb)
```

2.1.4. DiameterAgent service

Responsible for the communication with Diameter server via diameter protocol. Despite the name it is a flexible **Diameter Server**.

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
- none
```

- **Config section in the CGRateS configuration file:**

    - `"diameter_agent": {...}`

2.1.5. CDR service

Centralized CDR server and CDR (raw or rated) **replicator**.

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"stor_db" - (cdrDb)
"data_db" - (accountDb)
```

- **Config section in the CGRateS configuration file:**

    - "cdrs": {...}

2.1.6. CDRStats service

Computes real-time CDR stats. Capable with real-time fraud detection and mitigation with actions triggered.

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db"      - (dataDb)
```

- **Config section in the CGRateS configuration file:**

    - "cdrstats": {...}

2.1.7. CDRC service

Gathers offline CDRs and post them to CDR Server - (CDRS component)

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
- none
```

- **Config section in the CGRateS configuration file:**

    - "cdrc": {...}

2.1.8. Aliases service

Generic purpose **aliasing** system.

Possible applications:

- Change destination name based on user or destination prefix matched.

- Change lcr supplier name based on the user calling.

- Locale specifics, ability to display specific tags in user defined language.

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db" - (accountDb)
```

- **Config section in the CGRateS configuration file:**

    - `"aliases": {...}`

2.1.9. User service

Generic purpose **user** system to maintain user profiles (LDAP similarity).

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db" - (accountDb)
```

- **Config section in the CGRateS configuration file:**

    - `"users": {...}`

2.1.10. PubSub service

PubSub service used to expose internal events to interested external components (eg: balance ops)

- **Communicates via:**

    - RPC

    - internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db" - (accountDb)
```

- **Config section in the CGRateS configuration file:**

    - `"pubsubs": {...}`

2.1.11. Resource Limiter service

Resource Limiter service used to limit resources during authorization (eg: maximum calls per destination for an account)

- **Communicates via:**

- RPC
- internal/in-process *within the same running* **cgr-engine** process.

- Operates with the following CGRateS database(s):

```
"data_db" - (accountDb)
```

- **Config section in the CGRateS configuration file:**

- `"rls":  {...}`

2.1.12. APIER RPC service

RPC service used to expose external access towards internal components.

- **Communicates via:**

- JSON/GOB over socket
- JSON over HTTP
- JSON over WebSocket

2.1.13. Cdre

Component to retrieve rated CDRs from internal CDRs database.

- Communicates via:
- Operates with the following CGRateS database(s):

```
"stor_db" - (cdrDb)
```

- **Config section in the CGRateS configuration file:**

- `"cdre":  {...}`

2.1.14. Mailer

TBD

- Communicates via:
- Operates with the following CGRateS database(s):
- **Config section in the CGRateS configuration file:**

- `"mailer":  {...}`

2.1.15. Suretax

TBD

- Communicates via:
- Operates with the following CGRateS database(s):
- **Config section in the CGRateS configuration file:**

- `"suretax":  {...}`

2.1.X Mediator service

Important: This service is not valid anymore. Its functionality is replaced by CDRC and CDRS services.

Responsible to mediate the CDRs generated by Telecommunication Switch.

Has the ability to combine CDR fields into rating subject and run multiple mediation processes on the same record.

On Linux machines, able to work with inotify kernel subsystem in order to process the records close to real-time after the Switch has released them.

1.2.2 2.2. cgr-loader

Used for importing the rating information into the CGRateS database system.

Can be used to:

- Import information from **csv files** to **data_db**.
- Import information from **csv files** to **stor_db**. `-to_stordb -tpid`
- Import information from **stor_db** to **data_db**. `-from_stordb -tpid`

```
cgrates@OCS:~$ cgr-loader -help
Usage of cgr-loader:
  -cdrstats_address string
        CDRStats service to contact for data reloads, empty to disable automatic data
→reloads (default "127.0.0.1:2013")
  -datadb_host string
        The DataDb host to connect to. (default "127.0.0.1")
  -datadb_name string
        The name/number of the DataDb to connect to. (default "11")
  -datadb_passwd string
        The DataDb user's password.
  -datadb_port string
        The DataDb port to bind to. (default "6379")
  -datadb_type string
        The type of the DataDb database <redis> (default "redis")
  -datadb_user string
        The DataDb user to sign in as.
  -dbdata_encoding string
        The encoding used to store object data in strings (default "msgpack")
  -disable_reverse_mappings
        Will disable reverse mappings rebuilding
  -dry_run
        When true will not save loaded data to dataDb but just parse it for
→consistency and errors.
  -flushdb
        Flush the database before importing
  -from_stordb
        Load the tariff plan from storDb to dataDb
  -load_history_size int
        Limit the number of records in the load history (default 10)
  -migrate_rc8 string
        Migrate Accounts, Actions, ActionTriggers, DerivedChargers, ActionPlans and
→SharedGroups to RC8 structures, possible values: *all,acc,atr,act,dcs,apl,shg
  -path string
```

(continues on next page)

```
         The path to folder containing the data files (default "./")
  -rater_address string
         Rater service to contact for cache reloads, empty to disable automatic cache␣
→reloads (default "127.0.0.1:2013")
  -runid string
         Uniquely identify an import/load, postpended to some automatic fields
  -stats
         Generates statsistics about given data.
  -stordb_host string
         The storDb host to connect to. (default "127.0.0.1")
  -stordb_name string
         The name/number of the storDb to connect to. (default "cgrates")
  -stordb_passwd string
         The storDb user's password. (default "CGRateS.org")
  -stordb_port string
         The storDb port to bind to. (default "3306")
  -stordb_type string
         The type of the storDb database <mysql> (default "mysql")
  -stordb_user string
         The storDb user to sign in as. (default "cgrates")
  -timezone string
         Timezone for timestamps where not specified <""|UTC|Local|$IANA_TZ_DB>␣
→(default "Local")
  -to_stordb
         Import the tariff plan from files to storDb
  -users_address string
         Users service to contact for data reloads, empty to disable automatic data␣
→reloads (default "127.0.0.1:2013")
  -validate
         When true will run various check on the loaded data to check for structural␣
→errors
  -verbose
         Enable detailed verbose logging output
  -version
         Prints the application version.
```

Hint: # cgr-loader -flushdb

Hint: # cgr-loader -verbose -datadb_port="27017" -datadb_type="mongo"

1.2.3 2.3. cgr-console

Command line tool used to interface with the RALs service. Able to execute **sub-commands**.

```
cgrates@OCS:~$ cgr-console –help
Usage of cgr-console:
  -rpc_encoding string
         RPC encoding used <gob|json> (default "json")
  -server string
         server address host:port (default "127.0.0.1:2012")
  -verbose
```

```
        Show extra info about command execution.
  -version
        Prints the application version.


rif@grace:~$ cgr-console help_more
2013/04/13 17:23:51
Usage: cgr-console [cfg_opts...{-h}] <status|get_balance>
```

Hint: # cgr-console status

1.2.4 2.4. cgr-tester

Command line stress testing tool.

```
cgrates@OCS:~$ cgr-tester --help
Usage of cgr-tester:
 -datadb_host string
        The DataDb host to connect to. (default "127.0.0.1")
 -datadb_name string
        The name/number of the DataDb to connect to. (default "11")
 -datatdb_passwd string
        The DataDb user's password.
 -datadb_port string
        The DataDb port to bind to. (default "6379")
 -datadb_type string
        The type of the DataDb database <redis> (default "redis")
 -datadb_user string
        The DataDb user to sign in as.
 -category string
        The Record category to test. (default "call")
 -cpuprofile string
        write cpu profile to file
 -dbdata_encoding string
        The encoding used to store object data in strings. (default "msgpack")
 -destination string
        The destination to use in queries. (default "1002")
 -json
        Use JSON RPC
 -load_history_size int
        Limit the number of records in the load history (default 10)
 -memprofile string
        write memory profile to this file
 -parallel int
        run n requests in parallel
 -rater_address string
        Rater address for remote tests. Empty for internal rater.
 -runs int
        stress cycle number (default 10000)
 -subject string
        The rating subject to use in queries. (default "1001")
 -tenant string
        The type of record to use in queries. (default "cgrates.org")
 -tor string
```

```
        The type of record to use in queries. (default "*voice")
```

Hint: # cgr-tester -runs=10000

1.2.5 2.5. cgr-migrator

Command line migration tool.

```
cgrates@OCS:~$ cgr-migrator --help
Usage of cgr-migrator:
 -datadb_host string
     The DataDb host to connect to. (default "192.168.100.40")
 -datadb_name string
     The name/number of the DataDb to connect to. (default "10")
 -datadb_passwd string
     The DataDb user's password.
 -datadb_port string
     The DataDb port to bind to. (default "6379")
 -datadb_type string
     The type of the DataDb database <redis> (default "redis")
 -datadb_user string
     The DataDb user to sign in as. (default "cgrates")
 -dbdata_encoding string
     The encoding used to store object data in strings (default "msgpack")
 -dry_run
     When true will not save loaded data to dataDb but just parse it for consistency␣
→and errors.(default "false")
 -load_history_size int
     Limit the number of records in the load history (default 10)
 -migrate string
     Fire up automatic migration *to use multiple values use ',' as separator
     <*set_versions|*cost_details|*accounts|*actions|*action_triggers|*action_
→plans|*shared_groups>
 -old_datadb_host string
     The DataDb host to connect to. (default "192.168.100.40")
 -old_datadb_name string
     The name/number of the DataDb to connect to. (default "10")
 -old_datadb_passwd string
     The DataDb user's password.
 -old_datadb_port string
     The DataDb port to bind to. (default "6379")
 -old_datadb_type string
     The type of the DataDb database <redis>
 -old_datadb_user string
     The DataDb user to sign in as. (default "cgrates")
 -old_dbdata_encoding string
     The encoding used to store object data in strings
 -old_load_history_size int
     Limit the number of records in the load history
 -old_stordb_host string
     The storDb host to connect to. (default "192.168.100.40")
 -old_stordb_name string
     The name/number of the storDb to connect to. (default "cgrates")
```

```
-old_stordb_passwd string
    The storDb user's password.
-old_stordb_port string
    The storDb port to bind to. (default "3306")
-old_stordb_type string
    The type of the storDb database <mysql|postgres>
-old_stordb_user string
    The storDb user to sign in as. (default "cgrates")
-stats
    Generates statsistics about given data.(default "false")
-stordb_host string
    The storDb host to connect to. (default "192.168.100.40")
-stordb_name string
    The name/number of the storDb to connect to. (default "cgrates")
-stordb_passwd string
    The storDb user's password.
-stordb_port string
    The storDb port to bind to. (default "3306")
-stordb_type string
    The type of the storDb database <mysql|postgres> (default "mysql")
-stordb_user string
    The storDb user to sign in as. (default "cgrates")
-verbose
    Enable detailed verbose logging output.(default "false")
-version
    Prints the application version.
```

1.3 3. Installation

CGRateS can be installed via packages as well as Go automated source install. We recommend using source installs for advanced users familiar with Go programming and packages for users not willing to be involved in the code building process.

1.3.1 3.1. Using packages

3.1.1. Debian

This is for the moment the only packaged and the most recommended to use method to install CGRateS.

On the server you want to install CGRateS, simply execute the following commands:

```
wget http://www.cgrates.org/tmp_pkg/cgrates_0.9.1~rc8_amd64.deb
dpkg -i cgrates_0.9.1~rc8_amd64.deb
```

Once the installation is completed, one should perform the *3.2.3 Create Debian / Ubuntu Packages from Source* section in order to have the CGRateS properly set and ready to run. After *post-install* actions are performed, CGRateS will be configured in **/etc/cgrates/cgrates.json** and enabled in **/etc/default/cgrates**.

1.3.2 3.2. Using source

For developing CGRateS and switching between its versions, we are using the **new vendor directory feature** introduced in go 1.6. In a nutshell all the dependencies are installed and used from a folder named *vendor* placed in the

root of the project.

3.2.1 Install GO Lang

First we have to setup the GO Lang to our OS. Feel free to download the latest GO binary release from https://golang.org/dl/ In this Tutorial we are going to install Go 1.11

```
rm -rf /usr/local/go
cd /tmp
wget https://dl.google.com/go/go1.11.linux-amd64.tar.gz
sudo tar -xvf go1.11.0.linux-amd64.tar.gz -C /usr/local/
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
```

3.2.2 Build CGRateS from Source

To manage this *vendor* folder we use a tool named glide which will download specific versions of the external packages used by CGRateS. To configure the project with glide use the following commands:

```
go get github.com/Masterminds/glide
go get github.com/cgrates/cgrates
cd $GOPATH/src/github.com/cgrates/cgrates
glide install
./build.sh
```

The **glide install** command will install the external dependencies versions, specified in the **glide.lock** file, in the vendor folder. There are different versions for each CGRateS branch, versions that are recorded in the **lock** file when the GCRateS releases are made (using **glide update** command).

Note: The *vendor* folder **should not** be registered with the VCS we are using.

For more information and command options use glide readme page.

3.2.3 Create Debian / Ubuntu Packages from Source

After compiling the source code you are ready to create the .deb packages for your Debian like OS. But First lets install some dependencies.

```
sudo apt-get install build-essential fakeroot dh-systemd
```

Finally we are ready to create the system package. Before creation we make sure that we delete the old one first.

```
cd $GOPATH/src/github.com/cgrates/cgrates/packages
rm -rf $GOPATH/src/github.com/cgrates/*.deb
make deb
```

After some time and maybe some console warnings, your CGRateS package will be ready.

3.2.4 Install Custom Debian / Ubuntu Package

```
cd $GOPATH/src/github.com/cgrates
sudo dpkg -i cgrates_*.deb
```

1.3.3 3.3. Post-install

3.3.1. Database setup

For its operation CGRateS uses **one or more** database types, depending on its nature, install and configuration being further necessary.

At present we support the following databases:

> • Redis

Can be used as `data_db`. Optimized for real-time information access. Once installed there should be no special requirements in terms of setup since no schema is necessary.

> • MySQL

Can be used as `stor_db`. Optimized for CDR archiving and offline Tariff Plan versioning. Once MySQL is installed, CGRateS database needs to be set-up out of provided scripts. (example for the paths set-up by debian package)

```
cd /usr/share/cgrates/storage/mysql/
./setup_cgr_db.sh root CGRateS.org localhost
```

> • PostgreSQL

Can be used as `stor_db`. Optimized for CDR archiving and offline Tariff Plan versioning. Once PostgreSQL is installed, CGRateS database needs to be set-up out of provided scripts (example for the paths set-up by debian package)

```
cd /usr/share/cgrates/storage/postgres/
./setup_cgr_db.sh
```

> • MongoDB

Can be used as `data_db` - `stor_db`. It is the first database that can be used to store all kinds of data stored from CGRateS from accounts, tariff plans to cdrs and logs. This is provided as an alternative to Redis and/or MySQL/PostgreSQL and right now there are NO plans to drop support for any of them soon.

Once MongoDB is installed, CGRateS database needs to be set-up out of provided scripts (example for the paths set-up by debian package)

```
cd /usr/share/cgrates/storage/mongo/
./setup_cgr_db.sh
```

3.3.2 Set versions data

Once database setup is completed, we need to write the versions data. To do this, run migrator tool with the parameters specific to your database.

Sample usage for MySQL:

```
cgr-migrator -stordb_passwd="CGRateS.org" -migrate="*set_versions"
```

3.3.3.Git

The **historys** (History Service) component will use Git to archive *tariff plan changes* in a local repository, hence Git installation is necessary if you want to use this service.

1.4 4. Configuration

The behaviour of **CGRateS** can be externally influenced by following means:

- **Engine configuration files**: usually located at */etc/cgrates/*. There can be one or multiple file(s)/folder(s) hierarchies behind configuration folder with support for automatic includes. The file(s)/folder(s) will be imported **in alphabetical order** into final configuration object.

- **Tariff Plans**: set of files used to import various data used in CGRateS subsystems (eg: Rating, Accounting, LCR, DerivedCharging, etc).

- **RPC APIs**: set of JSON/GOB encoded APIs remotely available for various operational/administrative tasks.

1.4.1 4.1. cgr-engine configuration file

Organized into configuration sections. All configuration options come with defaults and we have tried our best to choose the best ones for a minimum of efforts necessary when running.

Below is the default configuration file which comes hardcoded into **cgr-engine**.

```
1  {
2
3  // Real-time Online/Offline Charging System (OCS) for Telecom & ISP environments
4  // Copyright (C) ITsysCOM GmbH
5  //
6  // This file contains the default configuration hardcoded into CGRateS.
7  // This is what you get when you load CGRateS with an empty configuration file.
8
9  // "general": {
10 //        "node_id": "",
                                                                                      /
   // identifier of this instance in the cluster, if empty it will be autogenerated
11 //        "logger": "*syslog",
                                                                          // controls
   the destination of logs <*syslog|*stdout>
12 //        "log_level": 6,
                                                                                      /
   // control the level of messages logged (0-emerg to 7-debug)
13 //        "http_skip_tls_verify": false,
                                                             // if enabled Http Client
   will accept any TLS certificate
14 //        "rounding_decimals": 5,
                                                                          // system
   level precision for floats
15 //        "dbdata_encoding": "*msgpack",
                                                             // encoding used to store
   object data in strings: <*msgpack|*json>
16 //        "tpexport_dir": "/var/spool/cgrates/tpe",                  //
    path towards export folder for offline Tariff Plans
17 //        "poster_attempts": 3,
                                                                          // number
   of attempts before considering post request failed (eg: *call_url, CDR replication)
18 //        "failed_posts_dir": "/var/spool/cgrates/failed_posts",      // directory
   path where we store failed requests
19 //        "default_request_type": "*rated",
                                                          // default request type to consider
   when missing from requests: <""|*prepaid|*postpaid|*pseudoprepaid|*rated>
```

```
20  //          "default_category": "call",
    ↪                                                              // default category␣
    ↪to consider when missing from requests
21  //          "default_tenant": "cgrates.org",
    ↪                                                      // default tenant to consider when␣
    ↪missing from requests
22  //          "default_timezone": "Local",
    ↪                                                          // default timezone for␣
    ↪timestamps where not specified <""|UTC|Local|$IANA_TZ_DB>
23  //          "connect_attempts": 3,
    ↪                                                                  // initial␣
    ↪server connect attempts
24  //          "reconnects": -1,
    ↪                                                                     //␣
    ↪number of retries in case of connection lost
25  //          "connect_timeout": "1s",
    ↪                                                           // consider␣
    ↪connection unsuccessful on timeout, 0 to disable the feature
26  //          "reply_timeout": "2s",
    ↪                                                               // consider␣
    ↪connection down for replies taking longer than this value
27  //          "response_cache_ttl": "0s",
    ↪                                                           // the life span of␣
    ↪a cached response
28  //          "internal_ttl": "2m",
    ↪                                                                 // maximum␣
    ↪duration to wait for internal connections before giving up
29  //          "locking_timeout": "0",
    ↪                                                               // timeout␣
    ↪internal locks to avoid deadlocks
30  //          "digest_separator": ",",
31  //          "digest_equal": ":",
32  // },
33
34
35  // "data_db": {                                                      //␣
    ↪database used to store runtime data (eg: accounts, cdr stats)
36  //          "db_type": "redis",                                        // data_
    ↪db type: <*redis|*mongo|*internal>
37  //          "db_host": "127.0.0.1",                                    // data_db␣
    ↪host address
38  //          "db_port": 6379,                                           // data_db␣
    ↪port to reach the database
39  //          "db_name": "10",                                           // data_db␣
    ↪database name to connect to
40  //          "db_user": "cgrates",                                  // username␣
    ↪to use when connecting to data_db
41  //          "db_password": "",                                          //␣
    ↪password to use when connecting to data_db
42  //          "redis_sentinel": "",                                 // redis_
    ↪sentinel is the name of sentinel
43  // },
44
45
46  // "stor_db": {                                                      //␣
    ↪database used to store offline tariff plans and CDRs
47  //          "db_type": "mysql",                                        // stor_
    ↪database type to use: <*mongo|*mysql|*postgres|*internal>
```

```
48   //          "db_host": "127.0.0.1",                              // the host
     →to connect to
49   //          "db_port": 3306,                                     // the
     →port to reach the stordb
50   //          "db_name": "cgrates",                                // stor
     →database name
51   //          "db_user": "cgrates",                                // username
     →to use when connecting to stordb
52   //          "db_password": "",                                   //
     →password to use when connecting to stordb
53   //          "max_open_conns": 100,                               // maximum
     →database connections opened, not applying for mongo
54   //          "max_idle_conns": 10,                                // maximum
     →database connections idle, not applying for mongo
55   //          "conn_max_lifetime": 0,                              // maximum
     →amount of time in seconds a connection may be reused (0 for unlimited), not
     →applying for mongo
56   //          "cdrs_indexes": [],                                  //
     →indexes on cdrs table to speed up queries, used only in case of mongo
57   // },
58
59
60   // "listen": {
61   //          "rpc_json": "127.0.0.1:2012",                        // RPC JSON listening
     →address
62   //          "rpc_gob": "127.0.0.1:2013",                         // RPC GOB listening
     →address
63   //          "http": "127.0.0.1:2080",                            // HTTP listening
     →address
64   //          "rpc_json_tls": "127.0.0.1:2022",                    // RPC JSON TLS listening
     →address
65   //          "rpc_gob_tls": "127.0.0.1:2023",                     // RPC GOB TLS listening
     →address
66   //          "http_tls": "127.0.0.1:2280",                        // HTTP TLS listening
     →address
67   //          "tls_server_certificate": "",                        // path to server
     →certificate(must conatin server.crt + ca.crt)
68   //          "tls_server_key": "",                                // path to
     →server key
69   //          "tls_client_certificate": "",                        // path to client
     →certificate(must conatin client.crt + ca.crt)
70   //          "tls_client_key": "",                                // path to
     →client key
71   // },
72
73
74   // "tls": {
75   //          "server_certificate" : "",                           // path to server
     →certificate
76   //          "server_key":"",                                     // path to server
     →key
77   //          "client_certificate" : "",                           // path to client
     →certificate
78   //          "client_key":"",                                     // path to client
     →key
79   //          "ca_certificate":"",                                 // path to CA
     →certificate (populate if used self-sign certificate otherwise let it empty)
```

```
80  //          "server_policy":4,                                        // server_policy␣
    ↪determine the TLS Client Authentication (0-NoClientCert, 1-RequestClientCert, 2-
    ↪RequireAnyClientCert, 3-VerifyClientCertIfGiven, 4-RequireAndVerifyClientCert)
81  //          "server_name":"",
82  // },
83
84
85  // "http": {                                                               //␣
    ↪ HTTP server configuration
86  //          "json_rpc_url": "/jsonrpc",                                     // JSON␣
    ↪RPC relative URL ("" to disable)
87  //          "ws_url": "/ws",                                                //␣
    ↪WebSockets relative URL ("" to disable)
88  //          "freeswitch_cdrs_url": "/freeswitch_json",         // Freeswitch CDRS␣
    ↪relative URL ("" to disable)
89  //          "http_cdrs": "/cdr_http",                                    // CDRS␣
    ↪relative URL ("" to disable)
90  //          "use_basic_auth": false,                                       // use␣
    ↪basic authentication
91  //          "auth_users": {},                                               //␣
    ↪ basic authentication usernames and base64-encoded passwords (eg: { "username1":
    ↪"cGFzc3dvcmQ=", "username2": "cGFzc3dvcmQy "})
92  // },
93
94
95  // "scheduler": {
96  //          "enabled": false,                                   // start Scheduler␣
    ↪service: <true|false>
97  //          "cdrs_conns": [],                                   // address where to reach␣
    ↪CDR Server, empty to disable CDR capturing <*internal|x.y.z.y:1234>
98  // },
99
100
101 // "cache":{
102 //          "destinations": {"limit": -1, "ttl": "", "static_ttl": false, "precache":␣
    ↪false},                        // destination caching
103 //          "reverse_destinations": {"limit": -1, "ttl": "", "static_ttl": false,
    ↪"precache": false},        // reverse destinations index caching
104 //          "rating_plans": {"limit": -1, "ttl": "", "static_ttl": false, "precache":␣
    ↪false},                     // rating plans caching
105 //          "rating_profiles": {"limit": -1, "ttl": "", "static_ttl": false, "precache
    ↪": false},              // rating profiles caching
106 //          "lcr_rules": {"limit": -1, "ttl": "", "static_ttl": false, "precache":␣
    ↪false},                         // lcr rules caching
107 //          "cdr_stats": {"limit": -1, "ttl": "", "static_ttl": false, "precache":␣
    ↪false},                        // cdr stats queues caching
108 //          "actions": {"limit": -1, "ttl": "", "static_ttl": false, "precache": false}
    ↪,                         // actions caching
109 //          "action_plans": {"limit": -1, "ttl": "", "static_ttl": false, "precache":␣
    ↪false},                    // action plans caching
110 //          "account_action_plans": {"limit": -1, "ttl": "", "static_ttl": false,
    ↪"precache": false},        // account action plans index caching
111 //          "action_triggers": {"limit": -1, "ttl": "", "static_ttl": false, "precache
    ↪": false},              // action triggers caching
112 //          "shared_groups": {"limit": -1, "ttl": "", "static_ttl": false, "precache":␣
    ↪false},                   // shared groups caching
113 //          "aliases": {"limit": -1, "ttl": "", "static_ttl": false, "precache": false}
    ↪,                         // aliases caching
```

```
114   //          "reverse_aliases": {"limit": -1, "ttl": "", "static_ttl": false, "precache
      ↪": false},                    // reverse aliases index caching
115   //          "derived_chargers": {"limit": -1, "ttl": "", "static_ttl": false, "precache
      ↪": false},                    // derived charging rule caching
116   //          "timings": {"limit": -1, "ttl": "", "static_ttl": false, "precache": false}
      ↪,                              // timings caching
117   //          "resource_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
      ↪"precache": false},            // control resource profiles caching
118   //          "resources": {"limit": -1, "ttl": "", "static_ttl": false, "precache":
      ↪false},                        // control resources caching
119   //          "event_resources": {"limit": -1, "ttl": "1m", "static_ttl": false},
      ↪                                            // matching resources to
      ↪events
120   //          "statqueue_profiles": {"limit": -1, "ttl": "1m", "static_ttl": false,
      ↪"precache": false},        // statqueue profiles
121   //          "statqueues": {"limit": -1, "ttl": "1m", "static_ttl": false, "precache":
      ↪false},                    // statqueues with metrics
122   //          "threshold_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
      ↪"precache": false},            // control threshold profiles caching
123   //          "thresholds": {"limit": -1, "ttl": "", "static_ttl": false, "precache":
      ↪false},                        // control thresholds caching
124   //          "filters": {"limit": -1, "ttl": "", "static_ttl": false, "precache": false}
      ↪,                              // control filters caching
125   //          "supplier_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
      ↪"precache": false},            // control supplier profile caching
126   //          "attribute_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
      ↪"precache": false},            // control attribute profile caching
127   //          "charger_profiles": {"limit": -1, "ttl": "", "static_ttl": false, "precache
      ↪": false},                // control charger profile caching
128   //          "resource_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
      ↪                              // control resource filter indexes caching
129   //          "stat_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
      ↪                                            // control stat filter indexes
      ↪caching
130   //          "threshold_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
      ↪                              // control threshold filter indexes caching
131   //          "supplier_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
      ↪                                            // control supplier filter indexes caching
132   //          "attribute_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
      ↪                              // control attribute filter indexes caching
133   //          "charger_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
      ↪                                            // control charger filter indexes caching
134   // },
135
136
137   // "filters": {                                                              //
      ↪Filters configuration (*new)
138   //          "stats_conns": [],                                                //
      ↪address where to reach the stat service, empty to disable stats functionality: <"
      ↪"|*internal|x.y.z.y:1234>
139   //          "indexed_selects":true,                                          // enable
      ↪profile matching exclusively on indexes
140   // },
141
142
143   // "rals": {
144   //          "enabled": false,                                                // enable
      ↪Rater service: <true|false>
```

Chapter 1. Full contents:

```
145  //          "thresholds_conns": [],                                      // address
     →where to reach the thresholds service, empty to disable thresholds functionality: <"
     →"|*internal|x.y.z.y:1234>
146  //          "stats_conns": [],                                           //
     →address where to reach the stat service, empty to disable stats functionality: <"
     →"|*internal|x.y.z.y:1234>
147  //          "pubsubs_conns": [],                                         // address
     →where to reach the pubusb service, empty to disable pubsub functionality: <"
     →"|*internal|x.y.z.y:1234>
148  //          "users_conns": [],                                           //
     →address where to reach the user service, empty to disable user profile
     →functionality: <""|*internal|x.y.z.y:1234>
149  //          "aliases_conns": [],                                         // address
     →where to reach the aliases service, empty to disable aliases functionality: <"
     →"|*internal|x.y.z.y:1234>
150  //          "rp_subject_prefix_matching": false,       // enables prefix matching for
     →the rating profile subject
151  //          "lcr_subject_prefix_matching": false,      // enables prefix matching
     →for the lcr subject
152  //          "max_computed_usage": {                                      // do not
     →compute usage higher than this, prevents memory overload
153  //               "*any": "189h",
154  //               "*voice": "72h",
155  //               "*data": "107374182400",
156  //               "*sms": "10000"
157  //          },
158  // },
159
160
161  // "cdrs": {
162  //          "enabled": false,                                            // start
     →the CDR Server service:  <true|false>
163  //          "extra_fields": [],                                          //
     →extra fields to store in CDRs for non-generic CDRs
164  //          "store_cdrs": true,                                          //
     →store cdrs in storDb
165  //          "sessions_cost_retries": 5,                                  // number of
     →queries to sessions_costs before recalculating CDR
166  //          "chargers_conns": [],                                        // address
     →where to reach the charger service, empty to disable charger functionality: <"
     →"|*internal|x.y.z.y:1234>
167  //          "rals_conns": [
168  //               {"address": "*internal"}                                // address where
     →to reach the Rater for cost calculation, empty to disable functionality: <"
     →"|*internal|x.y.z.y:1234>
169  //          ],
170  //          "pubsubs_conns": [],                                         // address
     →where to reach the pubusb service, empty to disable pubsub functionality: <"
     →"|*internal|x.y.z.y:1234>
171  //          "attributes_conns": [],                                      // address
     →where to reach the attribute service, empty to disable attributes functionality: <"
     →"|*internal|x.y.z.y:1234>
172  //          "users_conns": [],                                           //
     →address where to reach the user service, empty to disable user profile
     →functionality: <""|*internal|x.y.z.y:1234>
173  //          "aliases_conns": [],                                         // address
     →where to reach the aliases service, empty to disable aliases functionality: <"
     →"|*internal|x.y.z.y:1234>
```

```
174  //        "thresholds_conns": [],                                      // address␣
    ↪where to reach the thresholds service, empty to disable thresholds functionality: <"
    ↪"|*internal|x.y.z.y:1234>
175  //        "stats_conns": [],                                           //␣
    ↪address where to reach the stat service, empty to disable stats functionality: <"
    ↪"|*internal|x.y.z.y:1234>
176  //        "online_cdr_exports": [],                                    // list of CDRE␣
    ↪profiles to use for real-time CDR exports
177  // },
178
179
180  // "cdre": {
181  //        "*default": {
182  //                "export_format": "*file_csv",
    ↪                                       // exported CDRs format <*file_csv|*file_
    ↪fwv|*http_post|*http_json_cdr|*http_json_map|*amqp_json_cdr|*amqp_json_map>
183  //                "export_path": "/var/spool/cgrates/cdre",            // path␣
    ↪where the exported CDRs will be placed
184  //                "filters" :[],
    ↪                                                                        // new␣
    ↪filters for cdre
185  //                "tenant": "cgrates.org",
    ↪                                         // tenant used in filterS.Pass
186  //                "synchronous": false,
    ↪                                            // block processing until␣
    ↪export has a result
187  //                "attempts": 1,
    ↪                                                        // Number␣
    ↪of attempts if not success
188  //                "field_separator": ",",
    ↪                                          // used field separator in␣
    ↪some export formats, eg: *file_csv
189  //                "usage_multiply_factor": {
190  //                        "*any":␣
    ↪1                                                                   //␣
    ↪multiply usage based on ToR field or *any for all
191  //                },
192  //                "cost_multiply_factor": 1,
    ↪                                       // multiply cost before export, eg:␣
    ↪add VAT
193  //                "header_fields": [],
    ↪                                             // template of the exported␣
    ↪header fields
194  //                "content_fields":␣
    ↪[                                                // template of the␣
    ↪exported content fields
195  //                        {"tag": "CGRID", "type": "*composed", "value": "~CGRID"},
196  //                        {"tag":"RunID", "type": "*composed", "value": "~RunID"},
197  //                        {"tag":"TOR", "type": "*composed", "value": "~ToR"},
198  //                        {"tag":"OriginID", "type": "*composed", "value": "~OriginID
    ↪"},
199  //                        {"tag":"RequestType", "type": "*composed", "value": "~
    ↪RequestType"},
200  //                        {"tag":"Tenant", "type": "*composed", "value": "~Tenant"},
201  //                        {"tag":"Category", "type": "*composed", "value": "~Category
    ↪"},
202  //                        {"tag":"Account", "type": "*composed", "value": "~Account"}
    ↪,
```

```
203  //                            {"tag":"Subject", "type": "*composed", "value": "~Subject"}
     ↪,
204  //                            {"tag":"Destination", "type": "*composed", "value": "~
     ↪Destination"},
205  //                            {"tag":"SetupTime", "type": "*composed", "value": "~
     ↪SetupTime", "layout": "2006-01-02T15:04:05Z07:00"},
206  //                            {"tag":"AnswerTime", "type": "*composed", "value": "~
     ↪AnswerTime", "layout": "2006-01-02T15:04:05Z07:00"},
207  //                            {"tag":"Usage", "type": "*composed", "value": "~Usage"},
208  //                            {"tag":"Cost", "type": "*composed", "value": "~Cost",
     ↪"rounding_decimals": 4},
209  //                ],
210  //                "trailer_fields": [],
     ↪                                                // template of the exported
     ↪trailer fields
211  //        },
212  // },
213
214
215  // "cdrc": [
216  //        {
217  //                "id": "*default",
     ↪                                                        // identifier of
     ↪the CDRC runner
218  //                "enabled": false,
     ↪                                                        // enable CDR
     ↪client functionality
219  //                "dry_run": false,
     ↪                                                        // do not send the
     ↪CDRs to CDRS, just parse them
220  //                "cdrs_conns": [
221  //                        {"address": "*internal"}
     ↪                                        // address where to reach CDR server.
     ↪<*internal|x.y.z.y:1234>
222  //                ],
223  //                "cdr_format": "csv",
     ↪                                                // CDR file format
     ↪<*csv|*freeswitch_csv|*fwv|*opensips_flatstore|*partial_csv>
224  //                "field_separator": ",",
     ↪                                                // separator used in case
     ↪of csv files
225  //                "timezone": "",
     ↪                                                        // timezone
     ↪for timestamps where not specified <""|UTC|Local|$IANA_TZ_DB>
226  //                "run_delay": 0,
     ↪                                                        // sleep
     ↪interval in seconds between consecutive runs, 0 to use automation via inotify
227  //                "max_open_files": 1024,
     ↪                                                // maximum simultaneous
     ↪files to process, 0 for unlimited
228  //                "data_usage_multiply_factor": 1024,
     ↪                        // conversion factor for data usage
229  //                "cdr_in_dir": "/var/spool/cgrates/cdrc/in",                  //
     ↪absolute path towards the directory where the CDRs are stored
230  //                "cdr_out_dir": "/var/spool/cgrates/cdrc/out",        // absolute
     ↪path towards the directory where processed CDRs will be moved
231  //                "failed_calls_prefix": "missed_calls",                       //
     ↪used in case of flatstore CDRs to avoid searching for BYE records
```

```
232  //                   "cdr_path": "",
     ↪                                                            // path␣
     ↪towards one CDR element in case of XML CDRs
233  //                   "cdr_source_id": "freeswitch_csv",                        /
     ↪/ free form field, tag identifying the source of the CDRs within CDRS database
234  //                   "filters": [],
     ↪                                                            // new␣
     ↪filters used in FilterS subsystem
235  //                   "tenant": "cgrates.org",
     ↪                                         // default tenant
236  //                   "continue_on_success": false,
     ↪                                    // continue to the next template if executed
237  //                   "partial_record_cache": "10s",
     ↪                                    // duration to cache partial records when␣
     ↪not pairing
238  //                   "partial_cache_expiry_action": "*dump_to_file",        // action␣
     ↪taken when cache when records in cache are timed-out <*dump_to_file|*post_cdr>
239  //                   "header_fields": [],
     ↪                                                            // template of the import␣
     ↪header fields
240  //                   "content_fields
     ↪":[                                                        // import␣
     ↪content_fields template, tag will match internally CDR field, in case of .csv value␣
     ↪will be represented by index of the field value
241  //                        {"tag": "TOR", "field_id": "ToR", "type": "*composed",
     ↪"value": "~2", "mandatory": true},
242  //                        {"tag": "OriginID", "field_id": "OriginID", "type":
     ↪"*composed", "value": "~3", "mandatory": true},
243  //                        {"tag": "RequestType", "field_id": "RequestType", "type":
     ↪"*composed", "value": "~4", "mandatory": true},
244  //                        {"tag": "Tenant", "field_id": "Tenant", "type": "*composed
     ↪", "value": "~6", "mandatory": true},
245  //                        {"tag": "Category", "field_id": "Category", "type":
     ↪"*composed", "value": "~7", "mandatory": true},
246  //                        {"tag": "Account", "field_id": "Account", "type":
     ↪"*composed", "value": "~8", "mandatory": true},
247  //                        {"tag": "Subject", "field_id": "Subject", "type":
     ↪"*composed", "value": "~9", "mandatory": true},
248  //                        {"tag": "Destination", "field_id": "Destination", "type":
     ↪"*composed", "value": "~10", "mandatory": true},
249  //                        {"tag": "SetupTime", "field_id": "SetupTime", "type":
     ↪"*composed", "value": "~11", "mandatory": true},
250  //                        {"tag": "AnswerTime", "field_id": "AnswerTime", "type":
     ↪"*composed", "value": "~12", "mandatory": true},
251  //                        {"tag": "Usage", "field_id": "Usage", "type": "*composed",
     ↪"value": "~13", "mandatory": true},
252  //                   ],
253  //                   "trailer_fields": [],
     ↪                                                    // template of the import␣
     ↪trailer fields
254  //                   "cache_dump_fields":␣
     ↪[                                                    // template used when␣
     ↪dumping cached CDR, eg: partial CDRs
255  //                        {"tag": "CGRID", "type": "*composed", "value": "~CGRID"},
256  //                        {"tag": "RunID", "type": "*composed", "value": "~RunID"},
257  //                        {"tag": "TOR", "type": "*composed", "value": "~ToR"},
258  //                        {"tag": "OriginID", "type": "*composed", "value": "~
     ↪OriginID"},
```

```
259  //                         {"tag": "RequestType", "type": "*composed", "value": "~
     ↪RequestType"},
260  //                         {"tag": "Tenant", "type": "*composed", "value": "~Tenant"},
261  //                         {"tag": "Category", "type": "*composed", "value": "~
     ↪Category"},
262  //                         {"tag": "Account", "type": "*composed", "value": "~Account
     ↪"},
263  //                         {"tag": "Subject", "type": "*composed", "value": "~Subject
     ↪"},
264  //                         {"tag": "Destination", "type": "*composed", "value": "~
     ↪Destination"},
265  //                         {"tag": "SetupTime", "type": "*composed", "value": "~
     ↪SetupTime", "layout": "2006-01-02T15:04:05Z07:00"},
266  //                         {"tag": "AnswerTime", "type": "*composed", "value": "~
     ↪AnswerTime", "layout": "2006-01-02T15:04:05Z07:00"},
267  //                         {"tag": "Usage", "type": "*composed", "value": "~Usage"},
268  //                         {"tag": "Cost", "type": "*composed", "value": "~Cost"},
269  //                 ],
270  //         },
271  // ],
272
273
274  // "sessions": {
275  //         "enabled": false,                                       // starts␣
     ↪session manager service: <true|false>
276  //         "listen_bijson": "127.0.0.1:2014",               // address where to␣
     ↪listen for bidirectional JSON-RPC requests
277  //         "chargers_conns": [],                                  // address␣
     ↪where to reach the charger service, empty to disable charger functionality: <"
     ↪"|*internal|x.y.z.y:1234>
278  //         "rals_conns": [
279  //                 {"address": "*internal"}                 // address where␣
     ↪to reach the RALs <""|*internal|127.0.0.1:2013>
280  //         ],
281  //         "cdrs_conns": [
282  //                 {"address": "*internal"}                 // address where␣
     ↪to reach CDR Server, empty to disable CDR capturing <*internal|x.y.z.y:1234>
283  //         ],
284  //         "resources_conns": [],                                 // address␣
     ↪where to reach the ResourceS <""|*internal|127.0.0.1:2013>
285  //         "thresholds_conns": [],                                // address␣
     ↪where to reach the ThresholdS <""|*internal|127.0.0.1:2013>
286  //         "stats_conns": [],                                     //␣
     ↪address where to reach the StatS <""|*internal|127.0.0.1:2013>
287  //         "suppliers_conns": [],                                 // address␣
     ↪where to reach the SupplierS <""|*internal|127.0.0.1:2013>
288  //         "attributes_conns": [],                                // address␣
     ↪where to reach the AttributeS <""|*internal|127.0.0.1:2013>
289  //         "session_replication_conns": [],                // replicate sessions␣
     ↪towards these session services
290  //         "debit_interval": "0s",                                 // interval␣
     ↪to perform debits on.
291  //         "min_call_duration": "0s",                           // only␣
     ↪authorize calls with allowed duration higher than this
292  //         "max_call_duration": "3h",                           // maximum call␣
     ↪duration a prepaid call can last
293  //         "session_ttl": "0s",                                  // time after␣
     ↪a session with no updates is terminated, not defined by default
```

```
294  //          //"session_ttl_max_delay": "",                      // activates session_
     →ttl randomization and limits the maximum possible delay
295  //          //"session_ttl_last_used": "",                      // tweak LastUsed_
     →for sessions timing-out, not defined by default
296  //          //"session_ttl_usage": "",                          // tweak Usage_
     →for sessions timing-out, not defined by default
297  //          "session_indexes": [],                              // index_
     →sessions based on these fields for GetActiveSessions API
298  //          "client_protocol": 1.0,                             // version_
     →of protocol to use when acting as JSON-PRC client <"0","1.0">
299  //          "channel_sync_interval": "0",                       // sync channels_
     →regularly (0 to disable sync session)
300  // },


303  // "asterisk_agent": {
304  //          "enabled": false,                                   // starts_
     →the Asterisk agent: <true|false>
305  //          "sessions_conns": [
306  //                  {"address": "*internal"}                    // connection_
     →towards session service: <*internal>
307  //          ],
308  //          "create_cdr": false,                                // create CDR_
     →out of events and sends it to CDRS component
309  //          "asterisk_conns":[                                  //_
     →instantiate connections to multiple Asterisk servers
310  //                  {"address": "127.0.0.1:8088", "user": "cgrates", "password":
     →"CGRateS.org", "connect_attempts": 3,"reconnects": 5}
311  //          ],
312  // },


315  // "freeswitch_agent": {
316  //          "enabled": false,                                   // starts_
     →the FreeSWITCH agent: <true|false>
317  //          "sessions_conns": [
318  //                  {"address": "*internal"}                    // connection_
     →towards session service: <*internal>
319  //          ],
320  //          "subscribe_park": true,                             //_
     →subscribe via fsock to receive park events
321  //          "create_cdr": false,                                // create CDR_
     →out of events and sends them to CDRS component
322  //          "extra_fields": [],                                 //_
     →extra fields to store in auth/CDRs when creating them
323  //          //"min_dur_low_balance": "5s",                      // threshold which_
     →will trigger low balance warnings for prepaid calls (needs to be lower than debit_
     →interval)
324  //          //"low_balance_ann_file": "",                       // file to be played_
     →when low balance is reached for prepaid calls
325  //          "empty_balance_context": "",                        // if defined, prepaid_
     →calls will be transferred to this context on empty balance
326  //          "empty_balance_ann_file": "",                       // file to be played_
     →before disconnecting prepaid calls on empty balance (applies only if no context_
     →defined)
327  //          "max_wait_connection": "2s",                        // maximum duration to_
     →wait for a connection to be retrieved from the pool
```

```
328   //          "event_socket_conns": [                                       //␣
      ↪instantiate connections to multiple FreeSWITCH servers
329   //                {"address": "127.0.0.1:8021", "password": "ClueCon", "reconnects":␣
      ↪5,"alias":""}
330   //          ],
331   // },


334   // "kamailio_agent": {
335   //          "enabled": false,                                              // starts␣
      ↪SessionManager service: <true|false>
336   //          "sessions_conns": [
337   //                {"address": "*internal"}                        // connection␣
      ↪towards session service: <*internal>
338   //          ],
339   //          "create_cdr": false,                                   // create CDR␣
      ↪out of events and sends them to CDRS component
340   //          "timezone": "",                                                //␣
      ↪timezone of the Kamailio server
341   //          "evapi_conns": [                                            //␣
      ↪instantiate connections to multiple Kamailio servers
342   //                {"address": "127.0.0.1:8448", "reconnects": 5}
343   //          ],
344   // },


347   // "diameter_agent": {
348   //          "enabled": false,
      ↪                                                                        /
      ↪/ enables the diameter agent: <true|false>
349   //          "listen": "127.0.0.1:3868",
      ↪                                                            // address␣
      ↪where to listen for diameter requests <x.y.z.y:1234>
350   //          "dictionaries_path": "/usr/share/cgrates/diameter/dict/",        // path␣
      ↪towards directory holding additional dictionaries to load
351   //          "sessions_conns": [
352   //                {"address": "*internal"}
      ↪                                              // connection␣
      ↪towards SessionService
353   //          ],
354   //          "origin_host": "CGR-DA",
      ↪                                                            // diameter␣
      ↪Origin-Host AVP used in replies
355   //          "origin_realm": "cgrates.org",
      ↪                                                      // diameter Origin-
      ↪Realm AVP used in replies
356   //          "vendor_id": 0,
      ↪                                                                        /
      ↪/ diameter Vendor-Id AVP used in replies
357   //          "product_name": "CGRateS",
      ↪                                                      // diameter␣
      ↪Product-Name AVP used in replies
358   //          "templates": {
359   //                "*cca": [
360   //                          {"tag": "SessionId", "field_id": "Session-Id",
      ↪"type": "*composed",
361   //                          "value": "~*req.Session-Id", "mandatory":␣
      ↪true},
```

```
362   //                                    {"tag": "OriginHost", "field_id": "Origin-Host",
      →"type": "*composed",
363   //                                        "value": "~*vars.OriginHost", "mandatory":
      →true},
364   //                                    {"tag": "OriginRealm", "field_id": "Origin-Realm",
      →"type": "*composed",
365   //                                        "value": "~*vars.OriginRealm", "mandatory
      →": true},
366   //                                    {"tag": "AuthApplicationId", "field_id": "Auth-
      →Application-Id", "type": "*composed",
367   //                                        "value": "~*vars.*appid", "mandatory":
      →true},
368   //                                    {"tag": "CCRequestType", "field_id": "CC-Request-
      →Type", "type": "*composed",
369   //                                        "value": "~*req.CC-Request-Type",
      →"mandatory": true},
370   //                                    {"tag": "CCRequestNumber", "field_id": "CC-Request-
      →Number", "type": "*composed",
371   //                                        "value": "~*req.CC-Request-Number",
      →"mandatory": true},
372   //                        ]
373   //          },
374   //          "request_processors": [],
375   // },


378   // "radius_agent": {
379   //          "enabled": false,
      →                                                                                /
      →/ enables the radius agent: <true|false>
380   //          "listen_net": "udp",
      →                                                                        //
      →network to listen on <udp|tcp>
381   //          "listen_auth": "127.0.0.1:1812",
      →                                                      // address where to listen
      →for radius authentication requests <x.y.z.y:1234>
382   //          "listen_acct": "127.0.0.1:1813",
      →                                                      // address where to listen
      →for radius accounting requests <x.y.z.y:1234>
383   //          "client_secrets":
      →{                                                                              /
      →/ hash containing secrets for clients connecting here <*default|$client_ip>
384   //                  "*default": "CGRateS.org"
385   //          },
386   //          "client_dictionaries":
      →{                                                                  // per
      →client path towards directory holding additional dictionaries to load (extra to RFC)
387   //                  "*default": "/usr/share/cgrates/radius/dict/",
      →                         // key represents the client IP or catch-all <*default|
      →$client_ip>
388   //          },
389   //          "sessions_conns": [
390   //                  {"address": "*internal"}
      →                                                              // connection
      →towards SessionService
391   //          ],
392   //          "cdr_requires_session": false,
      →                                                              // only create CDR
      →if there is an active session at terminate
```

```
393     //          "request_processors": [],
394     // },
395
396
397     // "http_agent": [],                                      // HTTP Agents, ie towards cnc.to
        ↪MVNE platform
398
399
400     // "pubsubs": {
401     //          "enabled": false,                            // starts PubSub service:
        ↪<true|false>.
402     // },
403
404
405     // "aliases": {
406     //          "enabled": false,                            // starts Aliases
        ↪service: <true|false>.
407     // },
408
409
410     // "users": {
411     //          "enabled": false,                            // starts User service:
        ↪<true|false>.
412     //          "indexes": [],                               // user profile
        ↪field indexes
413     // },
414
415
416     // "attributes": {                                                      //
        ↪Attribute service
417     //          "enabled": false,                                            // starts
        ↪attribute service: <true|false>.
418     //          //"string_indexed_fields": [],                     // query indexes
        ↪based on these fields for faster processing
419     //          "prefix_indexed_fields": [],                     // query indexes based
        ↪on these fields for faster processing
420     //          "process_runs": 1,                                           //
        ↪number of run loops when processing event
421     // },
422
423
424     // "chargers": {                                                // Charger
        ↪service
425     //          "enabled": false,                                    // starts
        ↪charger service: <true|false>.
426     //          "attributes_conns": [],                              // address
        ↪where to reach the AttributeS <""|127.0.0.1:2013>
427     //          //"string_indexed_fields": [],                     // query indexes
        ↪based on these fields for faster processing
428     //          "prefix_indexed_fields": [],                     // query indexes based
        ↪on these fields for faster processing
429     // },
430
431
432     // "resources": {                                               // Resource
        ↪service (*new)
433     //          "enabled": false,                                    // starts
        ↪ResourceLimiter service: <true|false>.
```

```
434  //          "store_interval": "",                              // dump cache␣
     ↪regularly to dataDB, 0 - dump at start/shutdown: <""|$dur>
435  //          "thresholds_conns": [],                                  // address␣
     ↪where to reach the thresholds service, empty to disable thresholds functionality: <"
     ↪"|*internal|x.y.z.y:1234>
436  //          //"string_indexed_fields": [],                        // query indexes␣
     ↪based on these fields for faster processing
437  //          "prefix_indexed_fields": [],                       // query indexes based␣
     ↪on these fields for faster processing
438  // },
439
440
441  // "stats": {                                                          // Stat␣
     ↪service (*new)
442  //          "enabled": false,                                        // starts␣
     ↪Stat service: <true|false>.
443  //          "store_interval": "",                              // dump cache␣
     ↪regularly to dataDB, 0 - dump at start/shutdown: <""|$dur>
444  //          "thresholds_conns": [],                                  // address␣
     ↪where to reach the thresholds service, empty to disable thresholds functionality: <"
     ↪"|*internal|x.y.z.y:1234>
445  //          //"string_indexed_fields": [],                        // query indexes␣
     ↪based on these fields for faster processing
446  //          "prefix_indexed_fields": [],                       // query indexes based␣
     ↪on these fields for faster processing
447  // },
448
449
450  // "thresholds": {                                                     //␣
     ↪Threshold service (*new)
451  //          "enabled": false,                                        // starts␣
     ↪ThresholdS service: <true|false>.
452  //          "store_interval": "",                              // dump cache␣
     ↪regularly to dataDB, 0 - dump at start/shutdown: <""|$dur>
453  //          //"string_indexed_fields": [],                        // query indexes␣
     ↪based on these fields for faster processing
454  //          "prefix_indexed_fields": [],                       // query indexes based␣
     ↪on these fields for faster processing
455  // },
456
457
458  // "suppliers": {                                                      // Supplier␣
     ↪service (*new)
459  //          "enabled": false,                                        // starts␣
     ↪SupplierS service: <true|false>.
460  //          //"string_indexed_fields": [],                        // query indexes␣
     ↪based on these fields for faster processing
461  //          "prefix_indexed_fields": [],                       // query indexes based␣
     ↪on these fields for faster processing
462  //          "attributes_conns": [],                                  // address␣
     ↪where to reach the AttributeS <""|127.0.0.1:2013>
463  //          "rals_conns": [
464  //                  {"address": "*internal"},                      // address where␣
     ↪to reach the RALs for cost/accounting  <*internal>
465  //          ],
466  //          "resources_conns": [],                                   // address␣
     ↪where to reach the Resource service, empty to disable functionality: <"
     ↪"|*internal|x.y.z.y:1234>
```

```
467  //          "stats_conns": [],                                              //␣
     ↪address where to reach the Stat service, empty to disable stats functionality: <"
     ↪"|*internal|x.y.z.y:1234>
468  // },

469

470

471  // "loaders": [
472  //          {
473  //                  "id": "*default",
     ↪                                                                           //␣
     ↪identifier of the Loader
474  //                  "enabled": false,
     ↪                                                                           // starts␣
     ↪as service: <true|false>.
475  //                  "tenant": "cgrates.org",
     ↪                                                         // tenant used in filterS.
     ↪Pass
476  //                  "dry_run": false,
     ↪                                                                           // do not␣
     ↪send the CDRs to CDRS, just parse them
477  //                  "run_delay": 0,
     ↪                                                                           //␣
     ↪sleep interval in seconds between consecutive runs, 0 to use automation via inotify
478  //                  "lock_filename": ".cgr.lck",
     ↪                                              // Filename containing concurrency␣
     ↪lock in case of delayed processing
479  //                  "caches_conns": [
480  //                          {"address": "*internal"},
     ↪                                              // address where to reach the␣
     ↪CacheS for data reload, empty for no reloads  <""|*internal|x.y.z.y:1234>
481  //                  ],
482  //                  "field_separator": ",",
     ↪                                                                 // separator used␣
     ↪in case of csv files
483  //                  "tp_in_dir": "/var/spool/cgrates/loader/in",              //␣
     ↪absolute path towards the directory where the CDRs are stored
484  //                  "tp_out_dir": "/var/spool/cgrates/loader/out",            //␣
     ↪absolute path towards the directory where processed CDRs will be moved
485  //                  "data":␣
     ↪[                                                                          /
     ↪/ data profiles to load
486  //                          {
487  //                                  "type": "*attributes",
     ↪                                              // data source type
488  //                                  "file_name": "Attributes.csv",
     ↪                                      // file name in the tp_in_dir
489  //                                  "fields": [
490  //                                          {"tag": "TenantID", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
491  //                                          {"tag": "ProfileID", "field_id": "ID",
     ↪"type": "*composed", "value": "~1", "mandatory": true},
492  //                                          {"tag": "Contexts", "field_id": "Contexts",
     ↪ "type": "*composed", "value": "~2"},
493  //                                          {"tag": "FilterIDs", "field_id": "FilterIDs
     ↪", "type": "*composed", "value": "~3"},
494  //                                          {"tag": "ActivationInterval", "field_id":
     ↪"ActivationInterval", "type": "*composed", "value": "~4"},
```

```
495  //                                         {"tag": "FieldName", "field_id": "FieldName
     ↪", "type": "*composed", "value": "~5"},
496  //                                         {"tag": "Initial", "field_id": "Initial",
     ↪"type": "*composed", "value": "~6"},
497  //                                         {"tag": "Substitute", "field_id":
     ↪"Substitute", "type": "*composed", "value": "~7"},
498  //                                         {"tag": "Append", "field_id": "Append",
     ↪"type": "*composed", "value": "~8"},
499  //                                         {"tag": "Weight", "field_id": "Weight",
     ↪"type": "*composed", "value": "~9"},
500  //                             ],
501  //                         },
502  //                         {
503  //                             "type": "*filters",
     ↪                                 // data source type
504  //                             "file_name": "Filters.csv",
     ↪                                 // file name in the tp_in_dir
505  //                             "fields": [
506  //                                 {"tag": "Tenant", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
507  //                                 {"tag": "ID", "field_id": "ID", "type":
     ↪"*composed", "value": "~1", "mandatory": true},
508  //                                 {"tag": "FilterType", "field_id":
     ↪"FilterType", "type": "*composed", "value": "~2"},
509  //                                 {"tag": "FilterFieldName", "field_id":
     ↪"FilterFieldName", "type": "*composed", "value": "~3"},
510  //                                 {"tag": "FilterFieldValues", "field_id":
     ↪"FilterFieldValues", "type": "*composed", "value": "~4"},
511  //                                 {"tag": "ActivationInterval", "field_id":
     ↪"ActivationInterval", "type": "*composed", "value": "~5"},
512  //                             ],
513  //                         },
514  //                         {
515  //                             "type": "*resources",
     ↪                                 // data source type
516  //                             "file_name": "Resources.csv",
     ↪                             // file name in the tp_in_dir
517  //                             "fields": [
518  //                                 {"tag": "Tenant", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
519  //                                 {"tag": "ID", "field_id": "ID", "type":
     ↪"*composed", "value": "~1", "mandatory": true},
520  //                                 {"tag": "FilterIDs", "field_id": "FilterIDs
     ↪", "type": "*composed", "value": "~2"},
521  //                                 {"tag": "ActivationInterval", "field_id":
     ↪"ActivationInterval", "type": "*composed", "value": "~3"},
522  //                                 {"tag": "TTL", "field_id": "UsageTTL",
     ↪"type": "*composed", "value": "~4"},
523  //                                 {"tag": "Limit", "field_id": "Limit", "type
     ↪": "*composed", "value": "~5"},
524  //                                 {"tag": "AllocationMessage", "field_id":
     ↪"AllocationMessage", "type": "*composed", "value": "~6"},
525  //                                 {"tag": "Blocker", "field_id": "Blocker",
     ↪"type": "*composed", "value": "~7"},
526  //                                 {"tag": "Stored", "field_id": "Stored",
     ↪"type": "*composed", "value": "~8"},
527  //                                 {"tag": "Weight", "field_id": "Weight",
     ↪"type": "*composed", "value": "~9"},
```

```
528  //                                          {"tag": "ThresholdIDs", "field_id":
     ↪"ThresholdIDs", "type": "*composed", "value": "~10"},
529  //                              ],
530  //                      },
531  //                      {
532  //                              "type": "*stats",
     ↪                                    // data source type
533  //                              "file_name": "Stats.csv",
     ↪                                 // file name in the tp_in_dir
534  //                              "fields": [
535  //                                      {"tag": "Tenant", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
536  //                                      {"tag": "ID", "field_id": "ID", "type":
     ↪"*composed", "value": "~1", "mandatory": true},
537  //                                      {"tag": "FilterIDs", "field_id": "FilterIDs
     ↪", "type": "*composed", "value": "~2"},
538  //                                      {"tag": "ActivationInterval", "field_id":
     ↪"ActivationInterval", "type": "*composed", "value": "~3"},
539  //                                      {"tag": "QueueLength", "field_id":
     ↪"QueueLength", "type": "*composed", "value": "~4"},
540  //                                      {"tag": "TTL", "field_id": "TTL", "type":
     ↪"*composed", "value": "~5"},
541  //                                      {"tag": "Metrics", "field_id": "Metrics",
     ↪"type": "*composed", "value": "~6"},
542  //                                      {"tag": "MetricParams", "field_id":
     ↪"Parameters", "type": "*composed", "value": "~7"},
543  //                                      {"tag": "Blocker", "field_id": "Blocker",
     ↪"type": "*composed", "value": "~8"},
544  //                                      {"tag": "Stored", "field_id": "Stored",
     ↪"type": "*composed", "value": "~9"},
545  //                                      {"tag": "Weight", "field_id": "Weight",
     ↪"type": "*composed", "value": "~10"},
546  //                                      {"tag": "MinItems", "field_id": "MinItems",
     ↪ "type": "*composed", "value": "~11"},
547  //                                      {"tag": "ThresholdIDs", "field_id":
     ↪"ThresholdIDs", "type": "*composed", "value": "~12"},
548  //                              ],
549  //                      },
550  //                      {
551  //                              "type": "*thresholds",
     ↪                                    // data source type
552  //                              "file_name": "Thresholds.csv",
     ↪                                 // file name in the tp_in_dir
553  //                              "fields": [
554  //                                      {"tag": "Tenant", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
555  //                                      {"tag": "ID", "field_id": "ID", "type":
     ↪"*composed", "value": "~1", "mandatory": true},
556  //                                      {"tag": "FilterIDs", "field_id": "FilterIDs
     ↪", "type": "*composed", "value": "~2"},
557  //                                      {"tag": "ActivationInterval", "field_id":
     ↪"ActivationInterval", "type": "*composed", "value": "~3"},
558  //                                      {"tag": "MaxHits", "field_id": "MaxHits",
     ↪"type": "*composed", "value": "~4"},
559  //                                      {"tag": "MinHits", "field_id": "MinHits",
     ↪"type": "*composed", "value": "~5"},
560  //                                      {"tag": "MinSleep", "field_id": "MinSleep",
     ↪ "type": "*composed", "value": "~6"},
```

```
561  //                                        {"tag": "Blocker", "field_id": "Blocker",
     ↪"type": "*composed", "value": "~7"},
562  //                                        {"tag": "Weight", "field_id": "Weight",
     ↪"type": "*composed", "value": "~8"},
563  //                                        {"tag": "ActionIDs", "field_id": "ActionIDs
     ↪", "type": "*composed", "value": "~9"},
564  //                                        {"tag": "Async", "field_id": "Async", "type
     ↪": "*composed", "value": "~10"},
565  //                              ],
566  //                      },
567  //                      {
568  //                              "type": "*suppliers",
     ↪                               // data source type
569  //                              "file_name": "Suppliers.csv",
     ↪                       // file name in the tp_in_dir
570  //                              "fields": [
571  //                                        {"tag": "Tenant", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
572  //                                        {"tag": "ID", "field_id": "ID", "type":
     ↪"*composed", "value": "~1", "mandatory": true},
573  //                                        {"tag": "FilterIDs", "field_id": "FilterIDs
     ↪", "type": "*composed", "value": "~2"},
574  //                                        {"tag": "ActivationInterval", "field_id":
     ↪"ActivationInterval", "type": "*composed", "value": "~3"},
575  //                                        {"tag": "Sorting", "field_id": "Sorting",
     ↪"type": "*composed", "value": "~4"},
576  //                                        {"tag": "SortingParamameters", "field_id":
     ↪"SortingParamameters", "type": "*composed", "value": "~5"},
577  //                                        {"tag": "SupplierID", "field_id":
     ↪"SupplierID", "type": "*composed", "value": "~6"},
578  //                                        {"tag": "SupplierFilterIDs", "field_id":
     ↪"SupplierFilterIDs", "type": "*composed", "value": "~7"},
579  //                                        {"tag": "SupplierAccountIDs", "field_id":
     ↪"SupplierAccountIDs", "type": "*composed", "value": "~8"},
580  //                                        {"tag": "SupplierRatingPlanIDs", "field_id
     ↪": "SupplierRatingPlanIDs", "type": "*composed", "value": "~9"},
581  //                                        {"tag": "SupplierResourceIDs", "field_id":
     ↪"SupplierResourceIDs", "type": "*composed", "value": "~10"},
582  //                                        {"tag": "SupplierStatIDs", "field_id":
     ↪"SupplierStatIDs", "type": "*composed", "value": "~11"},
583  //                                        {"tag": "SupplierWeight", "field_id":
     ↪"SupplierWeight", "type": "*composed", "value": "~12"},
584  //                                        {"tag": "SupplierBlocker", "field_id":
     ↪"SupplierBlocker", "type": "*composed", "value": "~13"},
585  //                                        {"tag": "SupplierParameters", "field_id":
     ↪"SupplierParameters", "type": "*composed", "value": "~14"},
586  //                                        {"tag": "Weight", "field_id": "Weight",
     ↪"type": "*composed", "value": "~15"},
587  //                              ],
588  //                      },
589  //                      {
590  //                              "type": "*chargers",
     ↪                               // data source type
591  //                              "file_name": "Chargers.csv",
     ↪                       // file name in the tp_in_dir
592  //                              "fields": [
593  //                                        {"tag": "Tenant", "field_id": "Tenant",
     ↪"type": "*composed", "value": "~0", "mandatory": true},
```

```
594  //                                            {"tag": "ID", "field_id": "ID", "type":
     →"*composed", "value": "~1", "mandatory": true},
595  //                                            {"tag": "FilterIDs", "field_id": "FilterIDs
     →", "type": "*composed", "value": "~2"},
596  //                                            {"tag": "ActivationInterval", "field_id":
     →"ActivationInterval", "type": "*composed", "value": "~3"},
597  //                                            {"tag": "RunID", "field_id": "RunID", "type
     →": "*composed", "value": "~4"},
598  //                                            {"tag": "AttributeIDs", "field_id":
     →"AttributeIDs", "type": "*composed", "value": "~5"},
599  //                                            {"tag": "Weight", "field_id": "Weight",
     →"type": "*composed", "value": "~6"},
600  //                              ],
601  //                      },
602  //              ],
603  //          },
604  // ],


607  // "mailer": {
608  //          "server": "localhost",
     →                                                    // the server to
     →use when sending emails out
609  //          "auth_user": "cgrates",
     →                                                    // authenticate to
     →email server using this user
610  //          "auth_password": "CGRateS.org",
     →                                              // authenticate to email server
     →with this password
611  //          "from_address": "cgr-mailer@localhost.localdomain"        // from address
     →used when sending emails out
612  // },


615  // "suretax": {
616  //          "url": "",                                                             /
     →/ API url
617  //          "client_number": "",                                          // client
     →number, provided by SureTax
618  //          "validation_key": "",                                         // validation
     →key provided by SureTax
619  //          "business_unit": "",                                          // client's
     →Business Unit
620  //          "timezone": "Local",                                          // convert the
     →time of the events to this timezone before sending request out <UTC|Local|$IANA_TZ_
     →DB>
621  //          "include_local_cost": false,                          // sum local
     →calculated cost with tax one in final cost
622  //          "return_file_code": "0",                              // default or
     →Quote purposes <0|Q>
623  //          "response_group": "03",                                       //
     →determines how taxes are grouped for the response <03|13>
624  //          "response_type": "D4",                                        //
     →determines the granularity of taxes and (optionally) the decimal precision for the
     →tax calculations and amounts in the response
625  //          "regulatory_code": "03",                              // provider type
626  //          "client_tracking": "CGRID",                                   // template
     →extracting client information out of StoredCdr; <$RSRFields>
```

```
627  //          "customer_number": "Subject",                          // template␣
     ↪extracting customer number out of StoredCdr; <$RSRFields>
628  //          "orig_number":  "Subject",                             // template␣
     ↪extracting origination number out of StoredCdr; <$RSRFields>
629  //          "term_number": "Destination",                          // template␣
     ↪extracting termination number out of StoredCdr; <$RSRFields>
630  //          "bill_to_number": "",                                  // template␣
     ↪extracting billed to number out of StoredCdr; <$RSRFields>
631  //          "zipcode": "",                                         //␣
     ↪template extracting billing zip code out of StoredCdr; <$RSRFields>
632  //          "plus4": "",                                           //␣
     ↪template extracting billing zip code extension out of StoredCdr; <$RSRFields>
633  //          "p2pzipcode": "",                                      //␣
     ↪template extracting secondary zip code out of StoredCdr; <$RSRFields>
634  //          "p2pplus4": "",                                        //␣
     ↪template extracting secondary zip code extension out of StoredCdr; <$RSRFields>
635  //          "units": "^1",                                         //␣
     ↪template extracting number of "lines" or unique charges contained within the␣
     ↪revenue out of StoredCdr; <$RSRFields>
636  //          "unit_type": "^00",                                    //␣
     ↪template extracting number of unique access lines out of StoredCdr; <$RSRFields>
637  //          "tax_included": "^0",                                  // template␣
     ↪extracting tax included in revenue out of StoredCdr; <$RSRFields>
638  //          "tax_situs_rule": "^04",                               // template␣
     ↪extracting tax situs rule out of StoredCdr; <$RSRFields>
639  //          "trans_type_code": "^010101",                          // template␣
     ↪extracting transaction type indicator out of StoredCdr; <$RSRFields>
640  //          "sales_type_code": "^R",                               // template␣
     ↪extracting sales type code out of StoredCdr; <$RSRFields>
641  //          "tax_exemption_code_list": "",                         // template␣
     ↪extracting tax exemption code list out of StoredCdr; <$RSRFields>
642  // },
643
644
645  // "loader": {                                                      //␣
     ↪loader for tariff plans out of .csv files
646  //          "tpid": "",
     ↪                                                    // tariff plan␣
     ↪identificator
647  //          "data_path": "",                                       // path␣
     ↪towards tariff plan files
648  //          "disable_reverse": false,                              // disable␣
     ↪reverse computing
649  //          "caches_conns": [                                      //␣
     ↪addresses towards cacheS components for reloads
650  //                  {"address": "127.0.0.1:2012", "transport": "*json"}
651  //          ],
652  //          "scheduler_conns": [
653  //                  {"address": "127.0.0.1:2012"}
654  //          ],
655  // },
656
657
658  // "migrator": {
659  //          "out_datadb_type": "redis",
660  //          "out_datadb_host": "127.0.0.1",
661  //          "out_datadb_port": "6379",
```

```
662  //          "out_datadb_name": "10",
663  //          "out_datadb_user": "cgrates",
664  //          "out_datadb_password": "",
665  //          "out_datadb_encoding": "msgpack",
666  //          "out_stordb_type": "mysql",
667  //          "out_stordb_host": "127.0.0.1",
668  //          "out_stordb_port": "3306",
669  //          "out_stordb_name": "cgrates",
670  //          "out_stordb_user": "cgrates",
671  //          "out_stordb_password": "",
672  // },
673
674
675  // "dispatcher":{
676  //          "enabled": false,                                        // starts␣
     ↪DispatcherS service: <true|false>.
677  //          "rals_conns": [],                                        //␣
     ↪address where to reach the RALs for dispatcherS  <*internal>
678  //          "resources_conns": [],                                   // address␣
     ↪where to reach the ResourceS <""|127.0.0.1:2013>
679  //          "thresholds_conns": [],                                  // address␣
     ↪where to reach the ThresholdS <""|127.0.0.1:2013>
680  //          "stats_conns": [],                                       //␣
     ↪address where to reach the StatS <""|127.0.0.1:2013>
681  //          "suppliers_conns": [],                                   // address␣
     ↪where to reach the SupplierS <""|127.0.0.1:2013>
682  //          "attributes_conns": [],                                  // address␣
     ↪where to reach the AttributeS <""|127.0.0.1:2013>
683  //          "sessions_conns": [],                                    // connection␣
     ↪towards SessionService
684  //          "chargers_conns": [],                                    // address␣
     ↪where to reach the ChargerS <""|127.0.0.1:2013>
685  //          "dispatching_strategy": "*first",              // strategy for␣
     ↪dispatching <*first|*random|*next|*broadcast>
686  // },
687
688
689  // "analyzers":{
690  //          "enabled":false                                          //␣
     ↪starts AnalyzerS service: <true|false>.
691  // },
692
693
694  }
```

1.4.2 4.2. Tariff Plans

Major concept within CGRateS architecture, implement mechanisms to load rating as well as account data into CGRateS. For importing the data into CGRateS database(s) we are using **csv** *files*. The import process can be started as many times it is desired with one ore more csv files and the existing values are overwritten.

Important: If **-flushdb** option is used when importing data with cgr-loader, then the database **is cleaned** before importing.

For more details see the **cgr-loader** tool from the tutorial chapter.

The rest of this section we will describe the content of every csv file.

4.2.1. Destinations

The destinations are binding together various prefixes / caller ids to define a logical destination group. A prefix can appear in multiple destination groups.

```
"Destinations.csv" - csv
"tp_destinations"  - stor_db
```

#Id	Prefix
DST_1002	1002
DST_1001	1001
DST_FS	10
DST_1003	1003

[0] - Id: Destination Id, a string by which this destination will be referenced in other places by.

[1] - Prefix: Prefix(es) attached to this destination. The prefix or caller id to be added to the specified destination.

4.2.2. Timings

Holds time related definitions. Describes the time periods that have different rates attached to them.

```
"Timings.csv" - csv
"tp_timings"  - stor_db
```

[0] - Tag: String by which this timing will be referenced in other places by.

[1] - Years: Integers separated by semicolons (;) specifying the years for this time period.

 *any in case of always.

[2] - Months: Integers from 1=January to 12=December separated by semicolons (;) specifying the months for this time period.

 *any in case of always (equivalent to 1;2;3;4;5;6;7;8;9;10;11;12).

[3] - MonthDays: Integers from 1 to 31 separated by semicolons (;) specifying the month days for this time period.

 *any in case of always.

[4] - WeekDays: Integers from 1=Monday to 7=Sunday separated by semicolons (;) specifying the week days for this time period.

 *any in case of always.

[5] - Time: The start time for this time period.

 If you set it to ***asap** (was ***now**) it will be replaced with the time of the data importing.

4.2.3. Rates

Defines price groups for various destinations which will be associated to various timings.

```
"Rates.csv"  - csv
"tp_rates"   - stor_db
```

#Id	ConnectFee	Rate	RateUnit	RateIncrement	GroupIntervalStart
RT_10CNT	0.2	0.1	60s	60s	0s
RT_10CNT	0	0.05	60s	1s	60s
RT_20CNT	0.4	0.2	60s	60s	0s
RT_20CNT	0	0.1	60s	1s	60s
RT_40CNT	0.8	0.4	60s	30s	0s
RT_40CNT	0	0.2	60s	10s	60s
RT_1CNT	0	0.01	60s	60s	0s
RT_1CNT_PER_SEC	0	0.01	1s	1s	0s
RT_SMS	0	0.01	1	1	0

[0] - Id: Rate Id, a string by which this *rate* will be referenced in other places by.

[1] - ConnectFee: ConnectFee applied once the call is answered. The price to be charged once at the beginning of the call to the specified destination.

[2] - Rate: Number of billing units this rate applies to. The price for the billing unit expressed in cents.

[3] - RateUnit: The billing unit expressed in seconds.

[4] - RateIncrement: This rate will apply in increments of duration. The time gap for the rate

[5] - GroupIntervalStart: When the rate starts

See also:

Rateincrement and GroupIntervalStart are when the calls has different rates in the timeframe. For example, the first 30 seconds of the calls has a rate of €0.1 and after that €0.2. The rate for this will the same TAG with two RateIncrements

4.2.4. Destination Rates

Attach rates to destinations.

```
"DestinationRates.csv" - csv
"tp_destination_rates" - stor_db
```

#Id	Destina-tionId	RatesTag	Rounding-Method	Round-ingDecimals	Max-Cost	MaxCost-Strategy
DR_1001_20CNT	DST_1001	RT_20CNT	*up	4	0	
DR_1002_20CNT	DST_1002	RT_20CNT	*up	4	0	
DR_1003_MAXCOST_DISC	DST_1003	RT_1CNT_PER_SEC	*up	4	0.12	*disconnect
DR_1001_10CNT	DST_1001	RT_10CNT	*up	4	0	
DR_SMS	*any	RT_SMS	*up	4	0	

[0] - Id: tbd

[1] - DestinationId: tbd

[2] - RatesTag: tbd

[3] - RoundingMethod: tbd

[4] - RoundingDecimals: tbd

[5] - MaxCost: tbd

[6] - MaxCostStrategy: tbd

4.2.5. Rating Plans

The *rating plan* makes the links between **Rating Profiles**, **Timings** and **Destination Rates** so each of them can be described once and various combinations are made possible.

```
"RatingPlans.csv" - csv
"tp_rating_plans" - stor_db
```

#Id	DestinationRatesId	TimingTag	Weight
RP_1001	DR_1002_20CNT	*any	10
RP_1001	DR_1003_MAXCOST_DISC	*any	10
RP_1002	DR_1001_20CNT	*any	10
RP_1002_LOW	DR_1001_10CNT	*any	10
RP_1003	DR_1001_10CNT	*any	10
RP_SMS	DR_SMS	*any	0

[0] - Id: A string by which this *rating plan* will be referenced in other places by.

[1] - DestinationRatesId: The rating id/tag described in the **Destination rates** file. (*DestinationRates.csv* - **Id**)

[2] - TimingTag: The timing tag described in the **Timings** file. (*Timings.csv* - **Tag**)

[3] - Weight: If multiple timings cab be applied to a call the one with the lower weight wins. An example here can be the Christmas day: we can have a special timing for this day but the regular day of the week timing can also be applied to this day. The weight will differentiate between the two timings.

4.2.6. Rating profiles

The *rating profile* **describes** the prices to be applied for various calls to various destinations in various time frames. When a call is made the CGRateS system will locate the rates to be applied to the call using the rating profiles.

```
"RatingProfiles.csv" - csv
"tp_rating_profiles" - stor_db
```

#Direction	Tenant	Category	Subject	ActivationTime	RatingPlanId	RatesFallbackSubject	CdrStatQueueIds
*out	cgrates.org	call	1001	2014-01-14T00:00:00Z	RP_1001		
*out	cgrates.org	call	1002	2014-01-14T00:00:00Z	RP_1002		
*out	cgrates.org	call	1003	2014-01-14T00:00:00Z	RP_1003		
*out	cgrates.org	sms	*any	2014-01-14T00:00:00Z	RP_SMS		

[0] - Direction: Can be ***in** or ***out** for the INBOUND and OUTBOUND calls.

[1] - Tenant: Used to distinguish between carriers if more than one share the same database in the CGRates system.

[2] - Category: Type of record specifies the kind of transmission this rate profile applies to.

[3] - Subject: The client/user for who this profile is detailing the rates.

[4] - ActivationTime: Multiple rates timings/prices can be created for one profile with different activation times. When a call is made the appropriate profile(s) will be used to rate the call. So future prices can be defined here and the activation time can be set as appropriate.

[5] - RatingPlanId: The rating plan id/tag described in the **Rating Plans** file. (*RatingPlans.csv* - **Id**)

> This specifies the profile to be used in case the call destination.

[6] - RatesFallbackSubject: This specifies another profile to be used in case the call destination will not be found in the current profile. The same tenant, tor and direction will be used.

[7] - CdrStatQueueIds: The cdr stats id described in the **Cdr Stats** file. (*CdrStats.csv* - **Id**)

> Stat Queue associated with this account.

4.2.7. Account actions

Describes the actions to be applied to the clients/users accounts. There are two kinds of actions: timed and triggered. For the timed actions there is a scheduler application that reads them from the database and executes them at the appropriate timings. The triggered actions are executed when the specified balance counters reach certain thresholds.

The accounts hold the various balances and counters to activate the triggered actions for each the client.

Balance types are: MONETARY, SMS, INTERNET, INTERNET_TIME, MINUTES.

```
"AccountActions.csv" – csv
"tp_account_actions" – stor_db
```

#Tenant	Account	ActionPlanId	ActionTriggersId	AllowNegative	Disabled
cgrates.org	1001	AP_PACKAGE_10			
cgrates.org	1002	AP_PACKAGE_10			
cgrates.org	1003	AP_PACKAGE_10			

[0] - Tenant: Used to distinguish between carriers if more than one share the same database in the CGRates system.

[1] - Account: The identifier for the user's account.

[2] - Direction: Can be ***in** or ***out** for the INBOUND and OUTBOUND calls.

[3] - ActionPlanId: The action plan id/tag described in the **Action plans** file. (*ActionPlans.csv* - **Id**)

> Forwards to a timed action group that will be used on this account.

[4] - ActionTriggersId: The action trigger id/tag described in the **Action triggers** file. (*ActionTriggers.csv* - **Tag**)

> Forwards to a triggered action group that will be applied to this account.

[5] - AllowNegative: TBD

[6] - Disabled: TBD

4.2.8 Action triggers

For each account there are counters that record the activity on various balances. Action triggers allow when a counter reaches a threshold to activate a group of actions. After the execution the action trigger is marked as used and will no longer be evaluated until the triggers are reset. See actions for action trigger resetting.

```
"ActionTriggers.csv" – csv
"tp_action_triggers" – stor_db
```

[0] - Tag: A string by which this action trigger will be referenced in other places by.

[1] - UniqueID: Unique id for the trigger in multiple ActionTriggers

[2] - ThresholdType: The threshold type. Can have one of the following:

- ***min_counter**: Fire when counter is less than ThresholdValue

- ***max_counter**: Fire when counter is greater than ThresholdValue

- ***min_balance**: Fire when balance is less than ThresholdValue

- ***max_balance**: Fire when balances is greater than ThresholdValue

- ***min_asr**: Fire when ASR(Average success Ratio) is less than ThresholdValue

- ***max_asr**: Fire when ASR is greater than ThresholdValue

- ***min_acd**: Fire when ACD(Average call Duration) is less than ThresholdValue

- ***max_acd**: Fire when ACD is greater than ThresholdValue

- ***min_acc**: Fire when ACC(Average call cost) is less than ThresholdValue

- ***max_acc**: Fire when ACC is greater than ThresholdValue

- ***min_tcc**: Fire when TCC(Total call cost) is less than ThresholdValue

- ***max_tcc**: Fire when TCC is greater than ThresholdValue

- ***min_tcd**: fire when TCD(total call duration) is less than thresholdvalue

- ***max_tcd**: fire when TCD is greater than thresholdvalue

- ***min_pdd**: Fire when PDD(Post Dial Delay) is less than ThresholdValue

- ***max_pdd**: Fire when PDD is greater than ThresholdValue

[3] - ThresholdValue: The value of the balance counter that will trigger this action.

[4] - Recurrent(Boolean): In case of trigger we can fire recurrent while it's active, or only the first time.

[5] - MinSleep: When Threshold is triggered we can sleep for the time specified.

[6] - ExpiryTime TBD

[7] - ActivationTime TBD

[8] - BalanceTag: Specifies the balance counter by which this action will be triggered. Can be:

- **MONETARY**

- **SMS**

- **INTERNET**

- **INTERNET_TIME**

- **MINUTES**

[9] - BalanceType: Specifies the balance type for this action:

- ***voice**: units of call minutes

- ***sms**: units of SMS

- ***data**: units of data

- ***monetary**: units of money

[10] - BalanceDirections: Can be ***in** or ***out** for the INBOUND and OUTBOUND calls.

[11] - BalanceCategories: Category of the call/trigger

[12] - BalanceDestinationIds: The destination id/tag described in the **Destinations** file. (*Destinations.csv* - **Id**) - rinor: need verification

Destination of the call/trigger

[13] - BalanceRatingSubject: TBD

[14] - BalanceSharedGroup: Shared Group of the call/trigger

[15] - BalanceExpiryTime: TBD

[16] - BalanceTimingIds: TBD

[17] - BalanceWeight: TBD

[18] - BalanceBlocker TBD

[19] - BalanceDisabled: TBD

[20] - StatsMinQueuedItems: Min of items that need to have a queue to reach this Trigger. Trigger actions only if this number is hit (stats only).

[21] - ActionsId: The actions id/tag described in the **Actions** file. (*Actions.csv* - **ActionsId**)

Forwards to an action group to be executed when the threshold is reached.

[22] - Weight: Specifies the order for these triggers to be evaluated. If there are multiple triggers are fired in the same time the ones with the lower weight will be executed first.

4.2.9. Action Plans

TBD

```
"ActionPlans.csv"  – csv
"tp_action_plans" – stor_db
```

#Id	ActionsId	TimingId	Weight
AP_PACKAGE_10	ACT_TOPUP_RST_10	*asap	10

[0] - Id: A string by which this action timing will be referenced in other places by.

[1] - ActionsId: Forwards to an action group to be executed when the timing is right.

[2] - TimingId: A timing (one time or recurrent) at which the action group will be executed

[3] - Weight: Specifies the order for these timings to be evaluated. If there are multiple action timings set to be execute on the same time the ones with the lower weight will be executed first.

4.2.10. Actions

TBD

```
"Actions.csv"  - csv
"tp_actions"   - stor_db
```

#ActionsId[0]	Action[1]	ExtraParameters[2]	Filter[3]	BalanceId[4]	BalanceType[5]	Directions[6]	Categories[7]	DestinationIds[8]	RatingSubject[9]	SharedGroup[10]	ExpiryTime[11]	TimingTag[12]	Units[13]	BalanceWeight[14]	BalanceBlocker[15]	BalanceDisabled[16]	Weight[17]
ACT_TOPUP_RST_10	*topup_reset				*monetary	*out		*any			*unlimited		10	10	false	false	10
ACT_LOG_WARNING	*log														false	false	10

[0] - **ActionsId:** A string by which this action will be referenced in other places by.

[1] - **Action:** The action type. Can have one of the following:

- ***allow_negative**: Allow to the account to have negative balance
- ***call_url**: Send a http request to the following url
- ***call_url_async**: Send a http request to the following url Asynchronous
- ***cdrlog**: Log the current action in the storeDB
- ***debit**: Debit account balance.
- ***deny_negative**: Deny to the account to have negative balance
- ***disable_account**: Disable account in the platform
- ***enable_account**: Enable account in the platform
- ***log**: Logs the other action values (for debugging purposes).
- ***mail_async**: Send a email to the direction
- ***reset_account**: Sets all counters to 0
- ***reset_counter**: Sets the counter for the BalanceTag to 0
- ***reset_counters**: Sets *all* the counters for the BalanceTag to 0
- ***reset_triggers**: reset all the triggers for this account
- ***set_recurrent**: (pending)
- ***topup**: Add account balance. If the specific balance is not defined, define it (example: minutes per destination).
- ***topup_reset**: Add account balance. If previous balance found of the same type, reset it before adding.
- ***unset_recurrent**: (pending)
- ***unlimited**: (pending)

[2] - **ExtraParameters:** In Extra Parameter field you can define an argument for the action. In case of call_url Action, extraParameter will be the url action. In case of mail_async the email that you want to receive.

---

[3] - Filter TBD

[4] - BalanceId: The balance on which the action will operate

[5] - BalanceType: Specifies the balance type for this action:

> - ***voice**: units of call minutes
>
> - ***sms**: units of SMS
>
> - ***data**: units of data
>
> - ***monetary**: units of money

[6] - Directions: Can be ***in** or ***out** for the INBOUND and OUTBOUND calls.

[7] - Categories: TBD

[8] - DestinationIds: The destination id/tag described in the **Destinations** file. (*Destinations.csv* - Id)

> This field is used only if the BalanceId is MINUTES. Specifies the destination of the minutes to be operated.

[9] - RatingSubject: The ratingSubject of the Actions

[10] - SharedGroup: In case of the account uses any shared group for the balances.

[11] - ExpiryTime: TBD

[12] - TimingIds: Timming tag when the action can be executed. Default ALL.

[13] - Units: Number of units for decrease the balance. Only use if BalanceType is voice.

[14] - BalanceWeight: TBD

[15] - BalanceBlocker TBD

[16] - BalanceDisabled: TBD

[17] - Weight: If there are multiple actions in a group, they will be executed in the order of their weight (**smaller** first).

4.2.11. Derived Chargers

For each call we can bill more than one time, for that we need to use the following options:

```
"DerivedChargers.csv" - csv
"tp_derived_chargers" - stor_db
```

In derived charges we have 2 different kind of options, **FILTERS** and **ACTIONS** :

Filters: With the following fields we filter the calls that need to run a extra billing parameter.

[0] - Direction: TBD

[1] - Tenant: TBD

[2] - Category: TBD

[3] - Account: TBD

[4] - Subject: TBD

[5] - DestinationIds: TBD

Actions: In case of the filter options match, platform creates extra runid with the fields that we want to modify.

[6] - RunId: TBD

[7] - RunFilter: TBD

[8] - ReqTypeField: TBD

[9] - DirectionField: TBD

[10] - TenantField: TBD

[11] - CategoryField: TBD

[12] - AccountField: TBD

[13] - SubjectField: TBD

[14] - DestinationField: TBD

[15] - SetupTimeField: TBD

[16] - PddField: TBD

[17] - AnswerTimeField: TBD

[18] - UsageField: TBD

[19] - SupplierField: TBD

[20] - DisconnectCause: TBD

[21] - RatedField: TBD

[22] - CostField: TBD

In the example, all the calls with direction=out, tenant=cgrates.org, category="call" and account and subject equal 1001. Will be created a new cdr in the table *rated_cdrs* with the runID derived_run1, and the subject 1002.

This feature it's useful in the case that you want to rated the calls 2 times, for example rated for different tenants or resellers.

4.2.12. CDR Stats

CDR Stats enables some realtime statistics in your platform for multiple purposes, you can read more, see *CDR Stats Server*

```
"CdrStats.csv" - csv
"tp_cdr_stats" - stor_db
```

[0] - Id: Tag name for the Queue id

[1] - QueueLength: Maximum number of calls in this queue

[2] - TimeWindow: Window frame to store the calls

[3] - SaveInterval: Each interval queue stats will save in the stordb

[4] - Metric: Type of metric see *Metrics Types*

[5] - SetupInterval: TBD

[6] - TOR: TBD

[7] - CdrHost TBD

[8] - CdrSource: TBD

[9] - ReqType: Filter by reqtype

[10] - Direction: TBD

[11] - Tenant: Used to distinguish between carriers if more than one share the same database in the CGRates system.

[12] - Category: Type of record specifies the kind of transmission this rate profile applies to.

[13] - Account: The identifier for the user's account.

[14] - Subject: The client/user for who this profile is detailing the rates.

[15] - DestinationIds: Filter only by destinations prefix. Can be multiple separated with **;**

[16] - PddInterval: TBD

[17] - UsageInterval: TBD

[18] - Supplier: TBD

[19] - DisconnectCause: TBD

[20] - RunIds: TBD

[21] - RatedAccount: Filter by rated account

[22] - RatedSubject: Filter by rated subject

[23] - CostInterval: Filter by cost

[24] - ActionTriggers: ActionTriggers associated with this queue

4.2.13. Shared groups

TBD

```
"SharedGroups.csv" - csv
"tp_shared_groups" - stor_db
```

[0] - Id: TBD

[1] - Account: TBD

[2] - Strategy: TBD

[3] - RatingSubject: TBD

4.2.14. LCR rules

TBD

```
"LcrRules.csv" - csv
"tp_lcr_rules" - stor_db
```

[0] - Direction: TBD

[1] - Tenant: TBD

[2] - Category: TBD

[3] - Account: TBD

[4] - Subject: TBD

[5] - DestinationTag: TBD

[6] - RpCategory: TBD

[7] - Strategy: TBD

[8] - StrategyParams: TBD

[9] - ActivationTime: TBD

[10] - Weight: TBD

4.2.15. Users

TBD

```
"Users.csv" – csv
"tp_users"  – stor_db
```

[0] - Tenant: TBD

[1] - UserName: TBD

[2] - Masked: TBD

[3] - AttributeName: TBD

[4] - AttributeValue: TBD

[5] - Weight: TBD

4.2.16. Aliases

TBD

```
"Aliases.csv" – csv
"tp_aliases"  – stor_db
```

[0] - Direction: TBD

[1] - Tenant: TBD

[2] - Category: TBD

[3] - Account: TBD

[4] - Subject: TBD

[5] - DestinationId: TBD

[6] - Context: TBD

[7] - Target: TBD

[8] - Original: TBD

[9] - Alias: TBD

[10] - Weight: TBD

4.2.17. Resource Limits

TBD

```
"Resources.csv" – csv
"tp_resources"  – stor_db
```

[0] - Tag TBD

[1] - FilterType TBD

[2] - FilterFieldName TBD

[3] - FilterFieldValues TBD

[4] - ActivationTime TBD

[5] - Weight TBD

[6] - Limit TBD

[7] - ActionTriggerIds TBD

1.5 5. Administration

The general steps to get CGRateS operational are:

1. Create CSV files containing the initial data for CGRateS.

2. Load the data in the databases using the Loader application.

3. Start a Rater.

4. Start the SessionManager talking to your VoIP Switch or directly make API calls to the Rater.

5. Make API calls to the Rater or just let the SessionManager do the work.

1.6 6. Advanced Topics

1.6.1 API Calls

API calls are documented in the following GoDoc

1.6.2 CDR Server

An important component of every rating system is represented by the CDR Server. CGRateS includes an out of the box CDR Server component, controlable in the configuration file and supporting multiple interfaces for CDR feeds. This component makes the CDRs real-time accessible (influenced by the time of receiving them) to CGRateS subsystems.

Following interfaces are supported:

CDR-CGR

Available as handler within http server.

To feed CDRs in via this interface, one must use url of the form: <http://\protect\T1\textdollarip_configured:\protect\T1\textdollarport_configured/cdr_http>.

The CDR fields are received via http form (although for simplicity we support inserting them within query parameters as well) and are expected to be urlencoded in order to transport special characters reliably. All fields are expected by CGRateS as string, particular conversions being done on processing each CDR. The fields received are split into two different categories based on CGRateS interest in them:

Primary fields: the fields which CGRateS needs for it's own operations and are stored into cdrs_primary table of storDb.

- ToR: type of record, meta-field, should map to one of the TORs hardcoded inside the server <*voice|*data|*sms>
- OriginID: represents the unique accounting id given by the telecom switch generating the CDR
- OrderID: Stor order id used as export order id
- OriginHost: represents the IP address of the host generating the CDR (automatically populated by the server)
- Source: formally identifies the source of the CDR (free form field)
- RequestType: matching the supported request types by the **CGRateS**, accepted values are hardcoded in the server <prepaid|postpaid|pseudoprepaid|rated>.
- Category: free-form filter for this record, matching the category defined in rating profiles.
- Tenant: tenant whom this record belongs
- Account: account id (accounting subsystem) the record should be attached to
- Subject: rating subject (rating subsystem) this record should be attached to
- Destination: destination to be charged
- SetupTime: set-up time of the event. Supported formats: datetime RFC3339 compatible, SQL datetime (eg: MySQL), unix timestamp.
- AnswerTime: answer time of the event. Supported formats: datetime RFC3339 compatible, SQL datetime (eg: MySQL), unix timestamp.
- Usage: event usage information (eg: in case of tor=*voice this will represent the total duration of a call)
- CostSource: The source of this cost
- Rated: Mark the CDR as rated so we do not process it during rating

Extra fields: any field coming in via the http request and not a member of primary fields list. These fields are stored as json encoded into *cdrs_extra* table of storDb.

Example of sample CDR generated simply using curl:

```
curl --data "ToR=*voice \
 &Source=curl_cdr \
 &OrderID=abcde \
 &OriginHost=192.168.1.2 \
 &Source=sbc1 \
 &OriginID=qwerty3234567 \
 &ToR=*voice \
 &RequestType=*raw \
 &Tenant=192.168.56.66 \
 &Category=call \
 &Account=1004 \
 &Subject=1004 \
 &Destination=%2B4986517174963 \
 &SetupTime=2018-05-21T12:32:50Z \
 &AnswerTime=2018-05-21T12:32:56Z \
 &Usage=306 \
 &CostSource=*cdrs" http://127.0.0.1:2080/cdr_http
```

CDR-FS_JSON

Available as handler within http server, it implements the mechanism to store CDRs received from FreeSWITCH mod_json_cdr.

This interface is available at url: <http://\protect\T1\textdollarip_configured:\protect\T1\textdollarport_configured/freeswitch_json>.

This handler has a different implementation logic than the previous CDR-CGR, filtering fields received in the CDR from FreeSWITCH based on predefined configuration. The mechanism of extracting CDR information out of JSON encoded CDR received from FreeSWITCH is the following:

- When receiving the CDR from FreeSWITCH, CGRateS will extract the content of ''variables'' object.

- **Content of the ''variables'' will be filtered out and the following information will be stored into an internal CDR object:**

    - **Fields used by CGRateS in primary mediation, known as primary fields. These are:**

        * uuid: internally generated uuid by FreeSWITCH for the call

        * sip_local_network_addr: IP address of the FreeSWITCH box generating the CDR

        * sip_call_id: call id out of SIP protocol

        * cgr_reqtype: request type as understood by the CGRateS

        * cgr_category: call category (optional)

        * cgr_tenant: tenant this call belongs to (optional)

        * cgr_account: account id in CGRateS (optional)

        * cgr_subject: rating subject in CGRateS (optional)

        * cgr_destination: destination being rated (optional)

        * user_name: username as seen by FreeSWITCH (considered if cgr_subject or cgr_account not present)

        * dialed_extension: destination number considered if cgr_destination is missing

    - Fields stored at request in cdr_extra and definable in configuration file under *extra_fields*.

- Once the content will be filtered, the real CDR object will be processed, stored into storDb under *cdrs_primary* and *cdrs_extra* tables and, if configured, it will be passed further for mediation.

CDR-RPC

Available as RPC handler on top of CGR APIs exposed (in-process as well as GOB-RPC and JSON-RPC). This interface is used for example by CGR-SM component capturing the CDRs over event interface (eg: OpenSIPS or FreeSWITCH-ZeroConfig scenario)

The RPC function signature looks like this:

```
CDRSV1.ProcessCdr(cdr *utils.StoredCdr, reply *string) error
```

The simplified StoredCdr object is represented by following:

```
type StoredCdr struct {
  CgrId        string
  OrderId      int64           // Stor order id used as export order id
  TOR          string          // type of record, meta-field, should map to one
→of the TORs hardcoded inside the server <*voice|*data|*sms>
```
(continues on next page)

```
 AccId           string              // represents the unique accounting id given by␣
↪the telecom switch generating the CDR
 CdrHost         string              // represents the IP address of the host␣
↪generating the CDR (automatically populated by the server)
 CdrSource       string              // formally identifies the source of the CDR (free␣
↪form field)
 ReqType         string              // matching the supported request types by the␣
↪**CGRateS**, accepted values are hardcoded in the server
↪<prepaid|postpaid|pseudoprepaid|rated>.
 Direction       string              // matching the supported direction identifiers of␣
↪the CGRateS <*out>
 Tenant          string              // tenant whom this record belongs
 Category        string              // free-form filter for this record, matching the␣
↪category defined in rating profiles.
 Account         string              // account id (accounting subsystem) the record␣
↪should be attached to
 Subject         string              // rating subject (rating subsystem) this record␣
↪should be attached to
 Destination     string              // destination to be charged
 SetupTime       time.Time           // set-up time of the event. Supported formats:␣
↪datetime RFC3339 compatible, SQL datetime (eg: MySQL), unix timestamp.
 AnswerTime      time.Time           // answer time of the event. Supported formats:␣
↪datetime RFC3339 compatible, SQL datetime (eg: MySQL), unix timestamp.
 Usage           time.Duration       // event usage information (eg: in case of␣
↪tor=*voice this will represent the total duration of a call)
 ExtraFields     map[string]string // Extra fields to be stored in CDR
}
```

1.6.3 CDR Client (cdrc)

It's role is to gather offline CDRs and post them to CDR Server(CDRS) component.

Part of the *cgr-engine*, can be started on a remote server as standalone component.

Controlled within *cdrc* section of the configuration file.

Has two modes of operation:

- Automated: CDR file processing is triggered on file creation/move.

- Periodic: CDR file processing will be triggered at configured time interval (delay/sleep between processes) and it will be performed on all files present in the folder (IN) at run time.

Principles behind functionality:

- Monitor/process a CDR folder (IN) as outlined above.

- For every file processed, extract the information based on configuration and post it via configured mechanism to CDRS.

- The fields extracted out of each CDR row are the same ones depicted in the CDRS documentation (following primary and extra fields concept).

- Once the file processing completes, move it in it's original format in another folder (OUT) in order to avoid re-processing. Here it's worth mentioning the auto-detection of duplicated CDRs at server side based on accid and host fields.

- Advanced configuration like forking a number of simultaneous client instances monitoring different folders possible through the use of *.xml* configuration.

Import Templates

To specifiy custom imports (for various sources) one can specify *Import Templates*. These are definable within both *.cfg* as well as *.xml* advanced configuration files. For increased flexibility the Import Template can be defined using CGR-RSR fields capturing both ReGexp as well as static rules. The static values will be way faster in processing but limited in functionality.

CGR-RSR Regexp Rule

Format:

```
~field_id:s/regexp_search_and_capture_rule/output_teplate/
```

Example of usage:

```
Input CDR field:
        {
        "account": "First-Account123"
        }
Capture Rule:
        ~account:s/^*+(Account123)$/$1-processed/
Result after processing:
        {
        "account": "Account123-processed"
        }
```

CGR-RSR Static Rule

Format:

```
^field_id:static_value
```

Example of usage:

Input CDR field:

> { "account": "First-Account123" }

> **Capture Rule:** ^account:MasterAccount

> **Result after processing:** { "account": "MasterAccount" }

CDR Formats supported:

CDR .CSV

Most widely used format by Telecom Switches.

Light to read and generic to process. CDRC should be able to process in this way any .csv CDR, independent of the Telecom Switch generating them. Incompatibilities here can come out of answer time and duration formats which can vary between CDR writer implementations. As answer time we support a number of formats already - rfc3339, SQL/MySQL, unix timestamp. As duration we support nanoseconds granularity in our code. Time unit can be specified (eg: ms, s, m, h), or if missing, will default to nanoseconds.

In case of *.csv* files the Import Template will contain indexes for the possition where primary fields are located (0 representing the first field) and fieldname/position format for extra fields which need

not only to be extracted by row index but also to be named since .csv format does not save field names/labels. CDRC uses the following convention for extra fields in the configuration: *<label_extrafield_1>:<index_extrafield_1>[...,<label_extrafield_n>:<index_extrafield_n>]....*

1.6.4 CDR Exporter

Component to retrieve rated CDRs from internal CDRs database.

Although nowadays it is custom to read a storage/database with tools, we do not recommend doing it so due to possibility that reads can slow down complete rating system. For this purpose we have created exporter plugins which are meant to work in tight relationship with CGRateS internal components in order to best optimize performance and avoid system locks.

Export Templates

For advanced needs CGRateS Export Templates are configurable via *.cfg*, *.xml* as well as directly within RPC call requesting the export to be performed. Inside each Export Template one can either specify simple CDR field ids or use CGR-RSR fields capturing both Regexp as well as static rules.

CGR-RSR Regexp Rule

Format:

```
~field_id:s/regexp_search_and_capture_rule/output_teplate/
```

Example of usage:

```
Input CDR field:
  {
  "account": "First-Account123"
  }
Capture Rule:
  ~account:s/^*+(Account123)$/$1-processed/
Result after processing:
  {
  "account": "Account123-processed"
  }
```

CGR-RSR Static Rule

Format:

```
^field_id:static_value
```

Example of usage:

Input CDR field:

>   { "account": "First-Account123" }

>   **Capture Rule:** ^account:MasterAccount

>   **Result after processing:** { "account": "MasterAccount" }

Export interfaces implemented:

CGR-CSV

Simplest way to export CDRs in a format internally defined (with parts like *CDRExtraFields* configurable in main configuration file).

Principles behind exports:

- Exports are to be manually requested (although automated is planned for the future through the used of built-in scheduled actions) via exposed JSON-RPC api. Example of api call from python call provided as sample script:

```
rpc.call("ApierV1.ExportCsvCdrs",{"TimeStart":"1383823746","TimeEnd":
→"1383833746"} )
```

- On each export call there will be a .csv format file generated using configured separator. Location of the export folder is definable inside *cgrates.cfg*.

- File name of the export will appear like: *cdrs_$(timestamp).csv* where $(timestamp) will be replaced by unix timestamp of the server running the export process or requested via API call.

- Each exported file will have as content all the CDRs inside time interval defined in the API call. Both TimeStart and TimeEnd are optional, hence being able to obtain a full export of the available CDRs with one API call.

- To be noted here that CGRateS does not keep anywhere a history of exports, hence it is the responsibility of the system administrator to make sure that his exports are not doubled.

- If not otherwise defined, each line within the exported file will follow an internally predefined template:

cgrid,mediation_runid,tor,accid,reqtype,direction,tenant,category,account,subject,destination,setup_time,answer_time,usage,co

```
$(cgrid),$(mediation_runid),$(tor),$(accid),$(reqtype),$(direction),$(direction),
→$(tenant),$(category),$(account),$(subject),$(destination),$(setup_time),
→$(answer_time),$(usage),$(cost)
```

The significance of the fields exported:

- tor: type of record, meta-field, should map to one of the TORs hardcoded inside the server <*voice|*data|*sms>

- accid: represents the unique accounting id given by the telecom switch generating the CDR

- cdrhost: represents the IP address of the host generating the CDR (automatically populated by the server)

- cdrsource: formally identifies the source of the CDR (free form field)

- reqtype: matching the supported request types by the **CGRateS**, accepted values are hardcoded in the server <prepaid|postpaid|pseudoprepaid|rated>.

- direction: matching the supported direction identifiers of the CGRateS <*out>

- tenant: tenant whom this record belongs

- category: free-form filter for this record, matching the category defined in rating profiles.

- account: account id (accounting subsystem) the record should be attached to

- subject: rating subject (rating subsystem) this record should be attached to

- destination: destination to be charged

- setup_time: set-up time of the event. Supported formats: datetime RFC3339 compatible, SQL datetime (eg: MySQL), unix timestamp.

- answer_time: answer time of the event. Supported formats: datetime RFC3339 compatible, SQL datetime (eg: MySQL), unix timestamp.

- usage: event usage information (eg: in case of tor=*voice this will represent the total duration of a call)

- **extra_cdr_fields:**

  - selected list of cdr_extra fields via *cgrates.cfg* configuration or

  - alphabetical order of the cdr extra fields stored in cdr_extra table

Sample CDR export file content which was made available at path: */var/log/cgrates/cdr/out/cgr/csv/cdrs_1384104724.csv*

```
dbafe9c8614c785a65aabd116dd3959c3c56f7f6,default,*voice,dsafdsaf,rated,*out,cgrates.
→org,call,1001,1001,1002,2013-11-07T08:42:25Z,2013-11-07T08:42:26Z,10000000000,1.0100
```

CGR-FWV

Fixed width form of export CDR. Advanced template configuration available via *.xml* configuration file.

Hybrid CSV-FWV

For advanced needs **CGRateS** supports exporting the CDRs as combination between *.csv* and *.fwv* formats.

1.6.5 CDR Stats Server

Collects CDRs from various sources (eg: CGR-CDRS, CGR-Mediator, CGR-SM, third-party CDR source via RPC) and builds real-time stats based on them. Each StatsQueue has attached *ActionTriggers* with monitoring and actions capabilities.

Principles of functionality:

- Standalone component (can be started individually on remote hardware, isolated form other CGRateS compoenents).

- Performance oriented. Should be able to process tens of thousands of CDRs per second.

- Cache driven technology. But SaveInterval can be set to store this information on redis.

- Stats are build within *StatsQueues* a CDR Stats Server being able to support unlimited number of StatsQueues. Each CDR will be passed to all of StatsQueues available and will be processed by individual StatsQueue based on configuration.

- Stats will be build inside Metrics (eg: ASR, ACD, ACC, TCC) and attached to specific StatsQueue.

- Each StatsQueue will have attached one *ActionTriggers* profile which will monitor Metrics values and react on thresholds reached (unlimited number of thresholds and reactions configurable).

- CDRs are processed by StatsQueues if they pass CDR field filters.

- CDRs are auto-removed from StatsQueues in a *fifo* manner if the QueueLength is reached or if they do not longer fit within TimeWindow defined.

Configuration

Individual StatsQueue configurations are loaded inside TariffPlan defitions, one configuration object is internally represented as:

```
type CdrStats struct {
    Id              string          // Config id, unique per config instance
    QueueLength     int             // Number of items in the stats buffer
    TimeWindow      time.Duration   // Will only keep the CDRs who's call setup
→time is not older than time.Now()-TimeWindow
    SaveInterval    time.Duration   // Interval to store the info into database
    Metrics         []string        // ASR, ACD, ACC, TCC, TCD, PDD
    SetupInterval   []time.Time     // CDRFieldFilter on SetupInterval, 2 or less
→items (>= start interval,< stop_interval)
    TOR             []string        // CDRFieldFilter on TORs
    CdrHost         []string        // CDRFieldFilter on CdrHosts
    CdrSource       []string        // CDRFieldFilter on CdrSources
    ReqType         []string        // CDRFieldFilter on ReqTypes
    Direction       []string        // CDRFieldFilter on Directions
    Tenant          []string        // CDRFieldFilter on Tenants
    Category        []string        // CDRFieldFilter on Categories
    Account         []string        // CDRFieldFilter on Accounts
    Subject         []string        // CDRFieldFilter on Subjects
    DestinationPrefix []string      // CDRFieldFilter on DestinationPrefixes
    UsageInterval   []time.Duration // CDRFieldFilter on UsageInterval, 2 or less
→items (>= Usage, <Usage)
    PddInterval     []time.Duration // CDRFieldFilter on PddInterval, 2 or less
→items (>= Pdd, <Pdd)
    Supplier        []string        // CDRFieldFilter on Suppliers
    DisconnectCause []string        // Filter on DisconnectCause
    MediationRunIds []string        // CDRFieldFilter on MediationRunIds
    RatedAccount    []string        // CDRFieldFilter on RatedAccounts
    RatedSubject    []string        // CDRFieldFilter on RatedSubjects
    CostInterval    []float64       // CDRFieldFilter on CostInterval, 2 or less
→items, (>=Cost, <Cost)
    Triggers        ActionTriggerPriotityList
}
```

Metrics Types

- ACC *(Average Call Cost)*: Queue with the average call cost

- ACD *(Average Call Duration)*: Queue with the average call duration

- ASR *(Answer-Seizure Ratio)*: Queue with the answer ratio

- PDD *(Post Dial Delay)*: Queue with the average Post Dial Delay in seconds

- TCC *(Total Call Cost)*: Queue with the Total cost for the time frame.

- TCD *(Total Call Duration)*: Queue with the total call duration for the

time frame

ExternalQueries

The Metrics calculated are available to be real-time queried via RPC methods.

To facilitate interaction there are four commands built in the provided *cgr-console* tool:

- *cdrstats_queueids*: returns the queue ids processing CDR Stats.

- *cdrstats_metrics*: returns metrics calculated within specific CDRStatsQueue.

- *cdrstats_reload*: reloads the CdrStats configurations out of DataDb.

- *cdrstats_reset*: resets calculated metrics for one specific or all StatsQueues.

Example use

When you work with balance maybe you want to keep a eye in your users, so you can add a new queue for the last 5 hours to check that your customer it's not hacked, this is an example:

CDR stats:

```
"result":{
    "CdrStats": [
        {
            "Accounts": "my_account",
            "ActionTriggers": "FRAUD_CHECK",
            "Categories": "",
            "CdrHosts": "",
            "CdrSources": "",
            "CostInterval": "",
            "DestinationPrefixes": "",
            "Directions": "",
            "DisconnectCauses": "",
            "MediationRunIds": "",
            "Metrics": "TCC",
            "PddInterval": "",
            "QueueLength": "0",
            "RatedAccounts": "",
            "RatedSubjects": "",
            "ReqTypes": "",
            "SaveInterval": "15s",
            "SetupInterval": "",
            "Subjects": "",
            "Suppliers": "",
            "TORs": "",
            "Tenants": "foehn",
            "TimeWindow": "5h",
            "UsageInterval": ""
        }
    ],
    "CdrStatsId": "FRAUD_ACCOUNT",
    "TPid": "test"
}
```

Action Trigger:

```
"result": {
    "ActionTriggers": [
        {
            "ActionsId": "LOG_WARNING",
            "BalanceCategory": "",
            "BalanceDestinationIds": "",
            "BalanceDirection": "",
```

```
            "BalanceExpirationDate": "",
            "BalanceId": "",
            "BalanceRatingSubject": "",
            "BalanceSharedGroup": "",
            "BalanceTimingTags": "",
            "BalanceType": "",
            "BalanceWeight": 0,
            "Id": "",
            "MinQueuedItems": 0,
            "MinSleep": "3h",
            "Recurrent": true,
            "ThresholdType": "\\*max_tcc",
            "ThresholdValue": 150,
            "Weight": 10
        }
    ],
    "ActionTriggersId": "FRAUD_CHECK",
    "TPid": "test"
}
```

Using *cgr-console* you can check the status of the queue anytime:

```
cgr-console 'cdrstats_metrics StatsQueueId="FRAUD_ACCOUNT"'
```

1.6.6 DerivedCharging

DerivedCharging is the process of forking original request into a number (configured) of emulated ones, derived from the original parameters. This mechanism used in combination with multi-tenancy supported by default by **CGRateS** can give out complex charging scenarios, needed for example in case of whitelabel-ing.

DerivedCharging occurs in two separate places:

- SessionManager: necessary to handle each derived (emulated) session in it's individuall loop (eg: individual resellers will have their own charging policies implemented, some paying per minute, others per second and so on) and keep them in sync (eg: one reseller is left out of money, original call should be disconnected and all emulated sessions should end their debit loops).

- Mediator: necessary to fork the CDRs into a number of derived ones influenced by the derived charging configuration and rate them individually.

Configuration

DerivedCharging is configured in two places:

- Platform level configured within *cgrates.cfg* file.

- Account level configured as part of TarrifPlans defition or interactively via RPC methods.

One DerivedCharger object will be configured by an internal object like:

```
type DerivedCharger struct {
      RunId                string      // Unique runId in the chain
      RunFilters           string      // Only run the charger if all the filters
→match
      ReqTypeField         string      // Field containing request type info, number
→in case of csv source, '^' as prefix in case of static values
```

```
        DirectionField     string     // Field containing direction info
        TenantField        string     // Field containing tenant info
        CategoryField      string     // Field containing tor info
        AccountField       string     // Field containing account information
        SubjectField       string     // Field containing subject information
        DestinationField   string     // Field containing destination information
        SetupTimeField     string     // Field containing setup time information
        AnswerTimeField    string     // Field containing answer time information
        UsageField         string     // Field containing usage information
}
```

CGRateS is able to attach an unlimited number of DerivedChargers to a single request, based on configuration.

1.6.7 Rating logic

Let's start with the most important function: finding the cost of a certain call.

The call information comes to CGRateS having the following vital information like subject, destination, start time and end time. The engine will look up the database for the rates applicable to the received subject and destination.

```
type CallDescriptor struct {
        Direction
        TOR
        Tenant, Subject, Account, Destination
        TimeStart, TimeEnd
        LoopIndex       // indicates the position of this segment in a cost request
→loop
        CallDuration    // the call duration so far (partial or final)
        FallbackSubject // the subject to check for destination if not found on
→primary subject
        RatingPlans
}
```

When the session manager receives a call start event it will first check if the call is prepaid or postpaid. If the call is postpaid than the cost will be determined only once at the end of the call but if the call is prepaid there will be a debit operation every X seconds (X is configurable).

In prepaid case the rating engine will have to set rates for multiple parts of the call so the *LoopIndex* in the above structure will help the engine add the connect fee only to the first part. The *CallDuration* attribute is used to set the right rate in case the rates database has different costs for the different parts of a call e.g. first minute is more expensive (we can also define the minimum rate unit).

The **FallbackSubject** is used in case the initial call subject is not found in the rating profiles list (more on this later in this chapter).

What are the activation periods?

> At one given time there is a set of prices that applay to different time intervals when a call can be made. In CGRateS one can define multiple such sets that will become active in various point of time called activation time. The activation period is a structure describing different prices for a call on different intervals of time. This structure has an activation time, which specifies the active prices for a period of time by one ore more (usually more than one) rate intervals.

```
type RateInterval struct {
        Years
        Months
```

```
        MonthDays
        WeekDays
        StartTime, EndTime
        Weight, ConnectFee
        Prices
        RoundingMethod
        RoundingDecimals
}

type Price struct {
        GroupIntervalStart
        Value
        RateIncrement
        RateUnit
}
```

An **RateInterval** specifies the Month, the MonthDay, the WeekDays, the StartTime and the EndTime when the RateInterval's price profile is in effect.

> **Example** The RateInterval {"Month": [1], "WeekDays":[1,2,3,4,5], "StartTime":"18:00:00"} specifies the *Price* for the first month of each year from Monday to Friday starting 18:00. Most structure elements are optional and they can be combined in any way it makes sense. If an element is omitted it means it is zero or any.

The *ConnectFee* specifies the connection price for the call if this interval is the first one of the call.

The *Weight* will establish which interval will set the price for a call segment if more then one applies to it.

> **Example** Let's assume there is an interval defining price for the weekdays and another interval that defines a special holiday rates. As that holiday is also one of the regular weekdays than both intervals are applicable to a call made on that day so the interval with the smaller Weight will give the price for the call in question. If both intervals have the same Weight than the interval with the smaller price wins. It is, however, a good practice to set the Weight for the defined intervals.

The *RoundingMethod* and the *RoundingDecimals* will adjust the price using the specified function and number of decimals (more on this in the rates definition chapter).

The **Price** structure defines the start (*GroupIntervalStart*) of a section of a call with a specified rate *Value* per *RateUnit* diving and rounding the section in *RateIncrement* subsections.

So when there is a need to define new sets of prices just define new RatingPlans with the activation time set to the moment when it becomes active.

Let's get back to the engine. When a GetCost or Debit call comes to the engine it will try to match the best rating profile for the given *Direction*, *Tenant*, *TOR* and *Subject* using the longest *Subject* prefix method or using the *FallbackSubject* if not found. The rating profile contains the activation periods that might apply to the call in question.

At this point in rating process the engine will start splitting the call into various time spans using the following criterias:

1. Minute Balances: first it will handle the call information to the originator user acount to be split by available minute balances. If the user has free or special price minutes for the call destination they will be consumed by the call.

2. Activation periods: if there were not enough special price minutes available than the engine will check if the call spans over multiple activation periods (the call starts in initial rates period and continues in another).

3. RateIntervals: for each activation period that apply to the call the engine will select the best rate intervals that apply.

```
type TimeSpan struct {
      TimeStart, TimeEnd
      Cost
      RatingPlan
      RateInterval
      MinuteInfo
      CallDuration  // the call duration so far till TimeEnd
}
```

The result of this splitting will be a list of *TimeSpan* structures each having attached the MinuteInfo or the RateInterval that gave the price for it. The *CallDuration* attribute will select the right *Price* from the *RateInterval Prices* list. The final cost for the call will be the sum of the prices of these times spans plus the *ConnectionFee* from the first time span of the call.

User balances

The user account contains a map of various balances like money, sms, internet traffic, internet time, etc. Each of these lists contains one or more Balance structure that have a wheight and a possible expiration date.

```
type UserBalance struct {
      Type           // prepaid-postpaid
      BalanceMap
      UnitCounters
      ActionTriggers
}

type Balance struct {
      Value
      ExpirationDate
      Weight
}
```

CGRateS treats special priced or free minutes different from the rest of balances. They will be called free minutes further on but they can have a special price.

The free minutes must be handled a little differently because usually they are grouped by specific destinations (e.g. national minutes, ore minutes in the same network). So they are grouped in balances and when a call is made the engine checks all applicable balances to consume minutes according to that call.

When a call cost needs to be debited these minute balances will be queried for call destination first. If the user has special minutes for the specific destination those minutes will be consumed according to call duration.

A standard debit operation consist of selecting a certaing balance type and taking all balances from that list in the weight order to be debited till the total amount is consumed.

CGRateS provide api for adding/substracting user's money credit. The prepaid and postpaid are uniformly treated except that the prepaid is checked to be always greater than zero and the postpaid can go bellow zero.

Both prepaid and postpaid can have a limited number of free SMS and Internet traffic per month and this budget is replenished at regular intervals based on the user tariff plan or as the user buys more free SMSs (for example).

Another special feature allows user to get a better price as the call volume increases each month. This can be added on one ore more thresholds so the more he/she talks the cheaper the calls.

Finally bonuses can be rewarded to users who received a certain volume of calls.

1.6.8 FilterS

FilterS are code blocks applied to generic events (hashmaps) in order to allow/deny further processing.

A Tenant will define multiple Filter profiles via .csv or API calls. The Filter profile ID is unique within a tenant but it can be repeated over multiple Tenants.

In order to be used in event processing a Filter profile will be attached inside another subsystem profile definition, otherwise Filter profile will have no effect on it's own. A Filter profile can be shared between multiple subsystem profile definitions.

Filter profile

Definition:

```
type Filter struct {
      Tenant             string
      ID                 string
      Rules              []*FilterRule
      ActivationInterval *utils.ActivationInterval
}
```

A Filter profile can contain any number of Filter rules and all of them must pass in order for the filter profile to pass.

A Filter profile can be activated on specific interval, if multiple filters are used within a subsystem profile at least one needs to be active and passing in order for the subsystem profile to pass the event.

Filter rule

Definition:

```
type FilterRule struct {
      Type            string              // Filter type (*string, *timing, *rsr,
→*stats, *lt, *lte, *gt, *gte)
      FieldName       string              // Name of the field providing us the
→Values to check (used in case of some )
      Values          []string            // Filter definition
}
```

The matching logic of each FilterRule is given by it's type.

The following types are implemented:

- *string* will match in full the *FieldName* with at least one value defined inside *Values*. Any of them matching will match the FilterRule. It is indexed for performance and, in order to be enabled, the subsystem configuration where the Filter profile is used needs to have the parameter *string_indexed_fields* nil or contain the Filter profile ID inside.

- *prefix* will match at beginning of *FieldName* one of the values defined inside *Values*. It is indexed for performance and, in order to be enabled, the subsystem configuration where the Filter profile is used needs to have the parameter *prefix_indexed_fields* nil or contain the Filter profile ID inside.

- *timings* will compare the time contained in *FieldName* with one of the TimingIDs defined in Values.

- *destinations* will make sure that the *FieldName* is a prefix contained inside one of the destination IDs as *Values*.

- *rsr* will match the *RSRRules* defined in Values. The field name is taken out of *RSRRule.ID* and matching logic is done against *RSRRule.Filters*

- **lt* (less than), **lte* (less than or equal), **gt* (greather than), **gte* (greather than or equal) are comparison operators and they pass if at least one of the values defined in *Values* are passing for the *FieldName* of event. The operators are able to compare string, float, int, time.Time, time.Duration, however both types need to be the same, otherwise the filter will raise *incomparable* as error.

1.7 7. Tutorials

1.7.1 Asterisk Integration Tutorials

In these tutorials we exemplify a few cases of integration between Asterisk and CGRateS. We start with common steps, installation and postinstall processes, then we dive into particular configurations, depending on the case we run.

Software installation

We have chosen Debian Jessie as operating system.

Asterisk

We got Asterisk14 installed via following commands:

```
apt-get install autoconf build-essential openssl libssl-dev libsrtp-dev libxml2-dev↵
→libncurses5-dev uuid-dev sqlite3 libsqlite3-dev pkg-config libedit-dev
cd /tmp
wget --no-check-certificate https://raw.githubusercontent.com/asterisk/third-party/
→master/pjproject/2.7.2/pjproject-2.7.2.tar.bz2
wget --no-check-certificate https://raw.githubusercontent.com/asterisk/third-party/
→master/jansson/2.11/jansson-2.11.tar.bz2
wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-16-current.tar.gz
tar xzvf asterisk-16-current.tar.gz
cd asterisk-16.1.0/
./configure --with-jansson-bundled
make
make install
adduser --quiet --system --group --disabled-password --shell /bin/false --gecos
→"Asterisk" asterisk || true
```

Once installed we proceed with loading the configuration out of specific tutorial cases bellow.

CGRateS Installation

We have chosen Debian Jessie as operating system, since all the software components we use provide packaging for it.

Prerequisites

Some components of **CGRateS** (whether enabled or not, is up to the administrator) depend on external software like:

- Git used by **CGRateS** History Server as archiver.
- Redis to serve as Rating and Accounting DB for **CGRateS**.
- MySQL to serve as StorDB for **CGRateS**.

We will install them in one shot using the command bellow.

```
apt-get install git redis-server mysql-server
```

Note: We will use this [MySQL](#) root password when asked: *CGRateS.org*.

Installation

Installation steps are provided within the **CGRateS** [install documentation](#).

Since this tutorial is for master version of **CGRateS**, we will install CGRateS out of temporary .deb packages built out of master code:

```
wget http://www.cgrates.org/tmp_pkg/cgrates_0.9.1~rc8_amd64.deb
dpkg -i cgrates_0.9.1~rc8_amd64.deb
```

As described in post-install section, we will need to set up the [MySQL](#) database (using *CGRateS.org* as our root password):

```
cd /usr/share/cgrates/storage/mysql/
./setup_cgr_db.sh root CGRateS.org localhost
```

Once the database is in place, we can now set versions:

```
cgr-migrator -stordb_passwd="CGRateS.org" -migrate="*set_versions"
```

At this point we have **CGRateS** installed but not yet configured. To facilitate understanding and speed up the process, **CGRateS** has the configurations used in these tutorials available in the */usr/share/cgrates/tutorials* folder.

SIP UA - Jitsi

On our ubuntu desktop host, we have installed [Jitsi](#) to be used as SIP UA, out of stable provided packages on [Jitsi download](#) and had [Jitsi](#) configured with 4 accounts: 1001/CGRateS.org, 1002/CGRateS.org, 1003/CGRateS.org and 1004/CGRateS.org.

Asterisk interaction via *ARI*

Scenario

- Asterisk out of *basic-pbx* configuration samples.

- Considering the following users: 1001-prepaid, 1002-postpaid, 1003-pseudoprepaid, 1004-rated, 1007-rated.

- **CGRateS** with following components:

- CGR-SM started as translator between [Asterisk](#) and **CGR-RALs** for both authorization events (prepaid/pseudoprepaid) as well as postpaid ones.

- CGR-CDRS component processing raw CDRs from CGR-SM component and storing them inside CGR StorDB.

- CGR-CDRE exporting rated CDRs from CGR StorDB (export path: */tmp*).

- CGR-History component keeping the archive of the rates modifications (path browsable with git client at */tmp/cgr_history*).

Starting Asterisk with custom configuration

```
/usr/share/cgrates/tutorials/asterisk_ari/asterisk/etc/init.d/asterisk start
```

To verify that Asterisk is running we run the console command:

```
asterisk -r -s /tmp/cgr_asterisk_ari/asterisk/run/asterisk.ctl

  ari show status
```

Starting CGRateS with custom configuration

```
/usr/share/cgrates/tutorials/asterisk_ari/cgrates/etc/init.d/cgrates start
```

Make sure that cgrates is running

```
cgr-console status
```

CDR processing

At the end of each call Asterisk will generate an CDR event and due to automatic handler registration built in **CGRateS-SM** component, this will be directed towards the port configured inside *cgrates.json*. This event will reach inside **CGRateS** through the *SM* component (close to real-time). Once in-there it will be instantly rated and be ready for export.

CGRateS Usage

Since it is common to most of the tutorials, the example for **CGRateS** usage is provided in a separate page here

CGRateS Usage

Loading CGRateS Tariff Plans

Before proceeding to this step, you should have **CGRateS** installed and started with custom configuration, depending on the tutorial you have followed.

For our tutorial we load again prepared data out of shared folder, containing following rules:

- Create the necessary timings (always, asap, peak, offpeak).

- Configure 3 destinations (1002, 1003 and 10 used as catch all rule).

- As rating we configure the following:

- Rate id: *RT_10CNT* with connect fee of 20cents, 10cents per minute for the first 60s in 60s increments followed by 5cents per minute in 1s increments.

- Rate id: *RT_20CNT* with connect fee of 40cents, 20cents per minute for the first 60s in 60s increments, followed by 10 cents per minute charged in 1s increments.

- Rate id: *RT_40CNT* with connect fee of 80cents, 40cents per minute for the first 60s in 60s increments, follwed by 20cents per minute charged in 10s increments.

- Rate id: *RT_1CNT* having no connect fee and a rate of 1 cent per minute, chargeable in 1 minute increments.

- Rate id: *RT_1CNT_PER_SEC* having no connect fee and a rate of 1 cent per second, chargeable in 1 second increments.

- Accounting part will have following configured:

- Create 3 accounts: 1001, 1002, 1003.

- 1001, 1002 will receive 10units of **monetary* balance.

```
cgr-loader -verbose -path=/usr/share/cgrates/tariffplans/tutorial
```

To verify that all actions successfully performed, we use following *cgr-console* commands:

- Make sure all our balances were topped-up:

```
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1001"]'
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1002"]'
```

- Query call costs so we can see our calls will have expected costs (final cost will result as sum of *ConnectFee* and *Cost* fields):

```
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1002" AnswerTime="2014-08-04T13:00:00Z" Usage="20s"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1002" AnswerTime="2014-08-04T13:00:00Z" Usage="1m25s"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1003" AnswerTime="2014-08-04T13:00:00Z" Usage="20s"'
```

Test calls

1001 -> 1002

Since the user 1001 is marked as *prepaid* inside the telecom switch, calling between 1001 and 1002 should generate pre-auth and prepaid debits which can be checked with *get_account* command integrated within *cgr-console* tool. Charging will be done based on time of day as described in the tariff plan definition above.

Note: An important particularity to note here is the ability of **CGRateS** SessionManager to refund units booked in advance (eg: if debit occurs every 10s and rate increments are set to 1s, the SessionManager will be smart enough to refund pre-booked credits for calls stoped in the middle of debit interval).

Check that 1001 balance is properly deducted, during the call, and moreover considering that general balance has priority over the shared one debits for this call should take place at first out of general balance.

```
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1001"]'
```

1002 -> 1001

The user 1002 is marked as *postpaid* inside the telecom switch hence his calls will be debited at the end of the call instead of during a call and his balance will be able to go on negative without influencing his new calls (no pre-auth).

To check that we had debits we use again console command, this time not during the call but at the end of it:

```
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1002"]'
```

1001 -> 1003

The user 1001 call user 1003 and after 12 seconds the call will be disconnected.

CDR Exporting

Once the CDRs are mediated, they are available to be exported. One can use available RPC APIs for that or directly call exports from console:

```
cgr-console 'cdrs_export CdrFormat="csv" ExportPath="/tmp"'
```

Fraud detection

Since we have configured some action triggers (more than 20 units of balance topped-up or less than 2 and more than 5 units spent on *FS_USERS* we should be notified over syslog when things like unexpected events happen (eg: fraud with more than 20 units topped-up). Most important is the monitor for 100 units topped-up which will also trigger an account disable together with killing it's calls if prepaid debits are used.

To verify this mechanism simply add some random units into one account's balance:

```
cgr-console 'balance_set Tenant="cgrates.org" Account="1003" Direction="*out" Value=23
↪'
tail -f /var/log/syslog -n 20

cgr-console 'balance_set Tenant="cgrates.org" Account="1001" Direction="*out"␣
↪Value=101'
tail -f /var/log/syslog -n 20
```

On the CDRs side we will be able to integrate CdrStats monitors as part of our Fraud Detection system (eg: the increase of average cost for 1001 and 1002 accounts will signal us abnormalities, hence we will be notified via syslog).

1.7.2 FreeSWITCH Integration Tutorials

In these tutorials we exemplify a few cases of integration between FreeSWITCH and **CGRateS**. We start with common steps, installation and postinstall processes, then we dive into particular configurations.

Software installation

As operating system we have chosen Debian Jessie, since all the software components we use provide packaging for it.

FreeSWITCH

More information regarding the installation of FreeSWITCH on Debian can be found on it's official installation wiki.

To get FreeSWITCH installed and configured, we have choosen the simplest method, out of *vanilla* packages, plus one individual module we need: *mod-json-cdr*.

We will install FreeSWITCH via following commands:

```
wget -O - http://files.freeswitch.org/repo/deb/freeswitch-1.6/key.gpg |apt-key add -
echo "deb http://files.freeswitch.org/repo/deb/freeswitch-1.6/ jessie main" > /etc/
→apt/sources.list.d/freeswitch.list
apt-get update
apt-get install freeswitch-meta-vanilla freeswitch-mod-json-cdr libyuv-dev
```

Once installed, we will proceed with loading the configuration out of specific tutorial cases bellow.

FreeSWITCH generating *http-json* CDRs

Scenario

- FreeSWITCH with *vanilla* configuration adding *mod_json_cdr* for CDR generation.

- Modified following users (with configs in *etc/freeswitch/directory/default*): 1001-prepaid, 1002-postpaid, 1003-pseudoprepaid, 1004-rated, 1006-prepaid, 1007-rated.

- Have added inside default dialplan CGR own extensions just before routing towards users (*etc/freeswitch/dialplan/default.xml*).

- FreeSWITCH configured to generate default *http-json* CDRs.

- **CGRateS** with following components:

- CGR-SM started as prepaid controller, with debits taking place at 5s intervals.

- CGR-CDRS component receiving raw CDRs from FreeSWITCH, storing them and attaching costs inside CGR StorDB.

- CGR-CDRE exporting processed CDRs from CGR StorDB (export path: */tmp*).

- CGR-History component keeping the archive of the rates modifications (path browsable with git client at */tmp/cgr_history*).

Starting FreeSWITCH with custom configuration

```
/usr/share/cgrates/tutorials/fs_evsock/freeswitch/etc/init.d/freeswitch start
```

To verify that FreeSWITCH is running we run the console command:

```
fs_cli -x status
```

Starting CGRateS with custom configuration

```
/usr/share/cgrates/tutorials/fs_evsock/cgrates/etc/init.d/cgrates start
```

Check that cgrates is running

```
cgr-console status
```

CDR processing

At the end of each call FreeSWITCH will issue a http post with the CDR. This will reach inside **CGRateS** through the *CDRS* component (close to real-time). Once in-there it will be instantly rated and it is ready to be exported:

```
cgr-console 'cdrs_export CdrFormat="csv" ExportPath="/tmp"'
```

CGRateS Usage

Since it is common to most of the tutorials, the example for **CGRateS** usage is provided in a separate page here

1.7.3 Kamailio Integration Tutorials

In these tutorials we exemplify a few cases of integration between Kamailio and CGRateS. We start with common steps, installation and postinstall processes, then we dive into particular configurations, depending on the case we run.

Software installation

We have chosen Debian Jessie as operating system, since all the software components we use provide packaging for it.

Kamailio

We got Kamailio installed via following commands:

```
wget -O- http://deb.kamailio.org/kamailiodebkey.gpg | sudo apt-key add -
echo "deb http://deb.kamailio.org/kamailio52 stretch main" > /etc/apt/sources.list.d/
→kamailio.list
apt-get update
apt-get install kamailio kamailio-extra-modules kamailio-json-modules
```

Once installed we proceed with loading the configuration out of specific tutorial cases bellow.

Kamailio interaction via *evapi* module

Scenario

- Kamailio default configuration modified for **CGRateS** interaction. For script maintainability and simplicity we have separated CGRateS specific routes in *kamailio-cgrates.cfg* file which is included in main *kamailio.cfg* via include directive.
- Considering the following users (with configs hardcoded in the *kamailio.cfg* configuration script and loaded in htable): 1001-prepaid, 1002-postpaid, 1003-pseudoprepaid, 1004-rated, 1005-rated, 1006-prepaid, 1007-prepaid.
- **CGRateS** with following components:
- CGR-SM started as translator between Kamailio and CGR-Rater for both authorization events as well as accounting ones.
- CGR-CDRS component processing raw CDRs from CGR-SM component and storing them inside CGR StorDB.

- CGR-CDRE exporting rated CDRs from CGR StorDB (export path: */tmp*).

- CGR-History component keeping the archive of the rates modifications (path browsable with git client at */tmp/cgr_history*).

Starting Kamailio with custom configuration

```
/usr/share/cgrates/tutorials/kamevapi/kamailio/etc/init.d/kamailio start
```

To verify that Kamailio is running we run the console command:

```
kamctl moni
```

Starting CGRateS with custom configuration

```
/usr/share/cgrates/tutorials/kamevapi/cgrates/etc/init.d/cgrates start
```

Make sure that cgrates is running

```
cgr-console status
```

CDR processing

At the end of each call Kamailio will generate an CDR event via *evapi* and this will be directed towards the port configured inside *cgrates.json*. This event will reach inside **CGRateS** through the *SM* component (close to real-time). Once in-there it will be instantly rated and be ready for export.

CGRateS Usage

Since it is common to most of the tutorials, the example for **CGRateS** usage is provided in a separate page here

1.7.4 OpenSIPS Integration Tutorials

In these tutorials we exemplify a few cases of integration between OpenSIPS and CGRateS. We start with common steps, installation and postinstall processes, then we dive into particular configurations, depending on the case we run.

Software installation

We have chosen Debian Jessie as operating system, since all the software components we use provide packaging for it.

OpenSIPS

We got OpenSIPS installed via following commands:

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 049AD65B
echo "deb http://apt.opensips.org jessie 2.4-nightly" >/etc/apt/sources.list.d/
→opensips.list
apt-get update
apt-get install opensips opensips-cgrates-module
```

Once installed we proceed with loading the configuration out of specific tutorial cases bellow.

OpenSIPS interaction via *event_datagram*

Scenario

- OpenSIPS out of *residential* configuration generated.

- Considering the following users (with configs hardcoded in the *opensips.cfg* configuration script): 1002-postpaid, 1003-pseudoprepaid, 1004-rated, 1007-rated.

- For simplicity we configure no authentication (WARNING: Not for production usage).

- **CGRateS** with following components:

- CGR-SM started as translator between OpenSIPS and **cgr-rater** for both authorization events (pseudoprepaid) as well as CDR ones.

- CGR-CDRS component processing raw CDRs from CGR-SM component and storing them inside CGR StorDB.

- CGR-CDRE exporting rated CDRs from CGR StorDB (export path: */tmp*).

- CGR-History component keeping the archive of the rates modifications (path browsable with git client at */tmp/cgr_history*).

Starting OpenSIPS with custom configuration

```
/usr/share/cgrates/tutorials/osips_native/opensips/etc/init.d/opensips start
```

To verify that OpenSIPS is running we run the console command:

```
opensipsctl moni
```

Starting CGRateS with custom configuration

```
/usr/share/cgrates/tutorials/osips_native/cgrates/etc/init.d/cgrates start
```

Make sure that cgrates is running

```
cgr-console status
```

CDR processing

At the end of each call OpenSIPS will generate an CDR event and due to automatic handler registration built in **CGRateS-SM** component, this will be directed towards the port configured inside *cgrates.json*. This event will reach inside **CGRateS** through the *SM* component (close to real-time). Once in-there it will be instantly rated and be ready for export.

CGRateS Usage

Since it is common to most of the tutorials, the example for **CGRateS** usage is provided in a separate page here

1.8 8. Miscellaneous

1.8.1 8.1. FreeSWITCH integration

Being the original platform supported by CGRateS, FreeSWITCH has the advantage of support for complete set of CGRateS features. When used as Telecom Switch it fully supports all rating modes: **prepaid/postpaid/pseudoprepaid/rated**. A typical use case would be like the one in the diagram below:
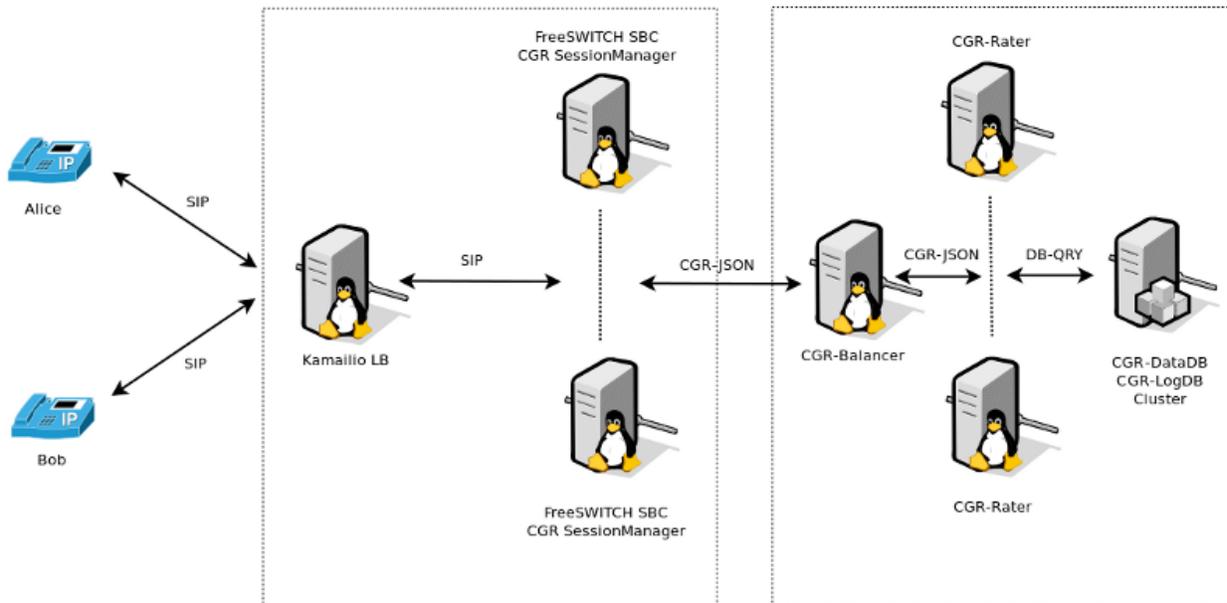
The process of rating is decoupled into two different components:

8.1.1. SessionManager

TODO - update and add CDRs and CDRc.

- Attached to FreeSWITCH via the socket library, enhancing CGRateS with real-time call monitoring and call control functions.

- **In Prepaid mode implements the following behaviour:**

    – **On *CHANNEL_PARK* event received from FreeSWITCH:**

        * Authorize the call by calling *GetMaxSessionTime* on the Rater.

        * **Sets the channel variable *cgr_notify* via *uuid_setvar* to one of the following values:**

            · MISSING_PARAMETER: if one of the required channel variables is missing and CGRateS cannot make rating.

            · SYSTEM_ERROR: if rating could not be performed due to a system error.

            · INSUFFICIENT_FUNDS: if MaximSessionTime is 0.

· AUTH_OK: Call is authorized to proceed.

* Un-Park the call via *uuid_transfer* to original dialed number. The FreeSWITCH administrator is expected to make use of *cgr_notify* variable value to either allow the call going further or reject it (eg: towards an IVR or returning authorization fail message to call originator).

– **On *CHANNEL_ANSWER* event received:**

* Index the call into CGRateS's cache.

* Starts debit loop by calling at configured interval *MaxDebit* on the Rater.

* **If any of the debits fail:**

· Set *cgr_notify* channel variable to either SYSTEM_ERROR in case of errors or INSUF-FICIENT_FUNDS of there would be not enough balance for the next debit to proceed.

· Send *hangup* command with cause *MANAGER_REQUEST*.

– **On *CHANNEL_HANGUP_COMPLETE* event received:**

* Refund the reserved balance back to the user's account (works for both monetary and minutes debited).

* Save call costs into CGRateS LogDB.

• In Postpaid mode:

– **On *CHANNEL_ANSWER* event received:**

* Index the call into CGRateS's cache.

– **On *CHANNEL_HANGUP_COMPLETE* event received:**

* Call *Debit* RPC method on the Rater.

* Save call costs into CGRateS LogDB.

• **On CGRateS Shutdown execute, for security reasons, hangup commands on calls which can be CGR related:**

– *hupall MANAGER_REQUEST cgr_reqtype prepaid*

– *hupall MANAGER_REQUEST cgr_reqtype postpaid*

8.1.2. Mediator

TODO - remove this section. Mediator functionality is handled by CDRs and CDRc.

Attaches costs to FreeSWITCH native written .csv files. Since writing channel variables during hangup is asynchronous and can be missed by the CDR recorder mechanism of FreeSWITCH, we decided to keep this as separate process after the call is completed and do not write the costs via channel variables.

8.1.2.1. Modes of operation

The Mediator process for FreeSWITCH works in two different modes:

• **Costs from LogDB (activated by setting -1 as *subject_idx* in the *cgrates.cfg*:**

– Queries LogDB for a previous saved price by SessionManager.

– This behavior is typical for prepaid/postpaid calls which were previously processed by SessionManager and important in the sense that we write in CDRs exactly what was billed real-time from user's account.

- **Costs queried from Rater:**

    - This mode is specific for multiple process mediation and does not necessary reflect the price which was deducted from the user's account during real-time rating.

    - Another application for this mode is pseudoprepaid when there is no SessionManager monitoring and charging calls in real-time (debit done directly from CDRs).

    - This mode is triggered from configuration file by setting proper indexes (or leave them defaults if cgrates rating template is using whitin FreeSWITCH cdr_csv configuration file.

A typical usage into our implementations is a combination between the two modes of operation (by setting at a minimum -1 as subject_idx to run from LogDB and succesive mediation processes with different indexes).

8.1.2.2. Implementation logic

- The Mediator process is configured and started in the *cgrates.cfg* file and is alive as long as the *cgr-engine* application is on.

- To avoid concurrency issues, the Mediator does not process active maintained CDR csv files by FreeSWITCH but picks them up as soon as FreeSWITCH has done with them by rotating. The information about rotation comes in real-time on the Linux OS through the use of inotify.

- Based on configured indexes in the configuration file, the Mediator will start multiple processes for the same CDR.

- For each mediation process configured the Mediator will apped the original CDR with costs calculated. In case of errors of some kind, the value *-1* will be prepended.

- When mediation is completed on a file, the file will be moved to configured *cdr_out_dir* path.