CGRates Documentation

Release 0.9.1~rc8

Radu Ioan Fericean/Dan Christian Bogos

Sep 21, 2017

Contents

CHAPTER 1

Full contents:

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).

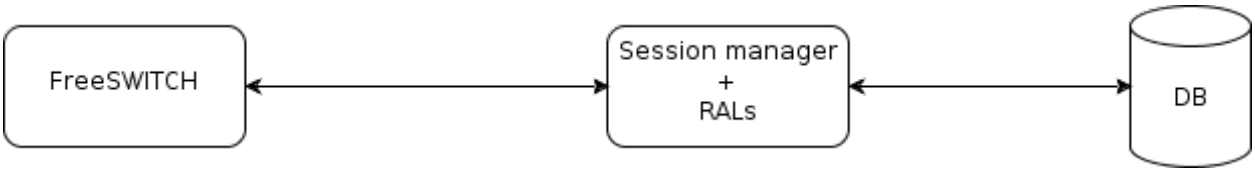
After testing various databases like Kyoto Cabinet, Apache Cassandra, Redis and MongoDB, the project focused on **Redis** as it delivers the best trade-off between speed, configuration and scalability.

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

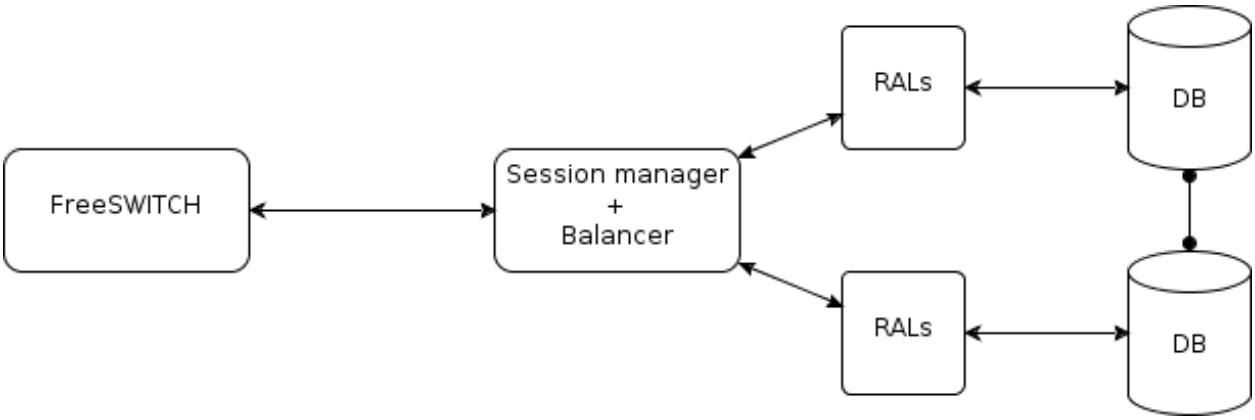
Thanks to CGRateS flexibility, connection to any database can be easily integrated by writing a simple adapter.

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

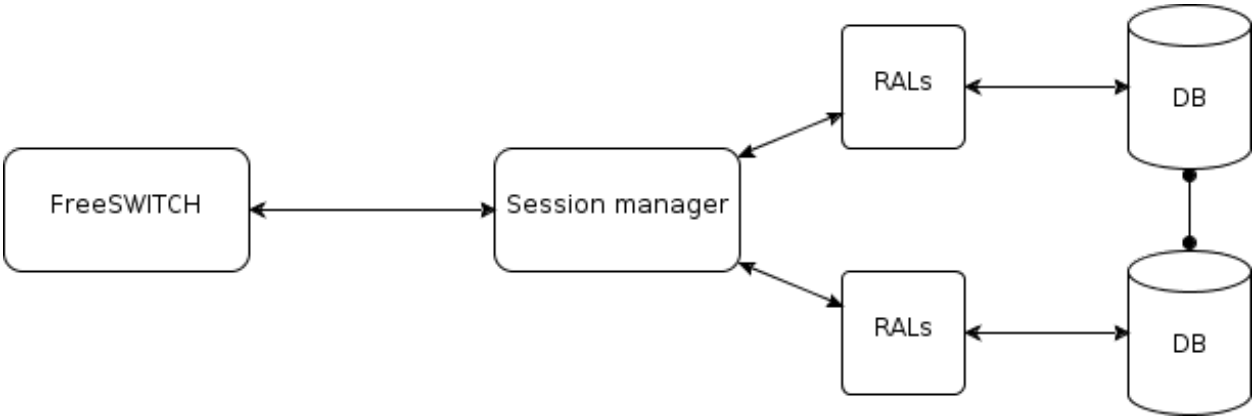
Note: **RALs** - is a CGRateS component and stands for RatingAccountingLCR service.



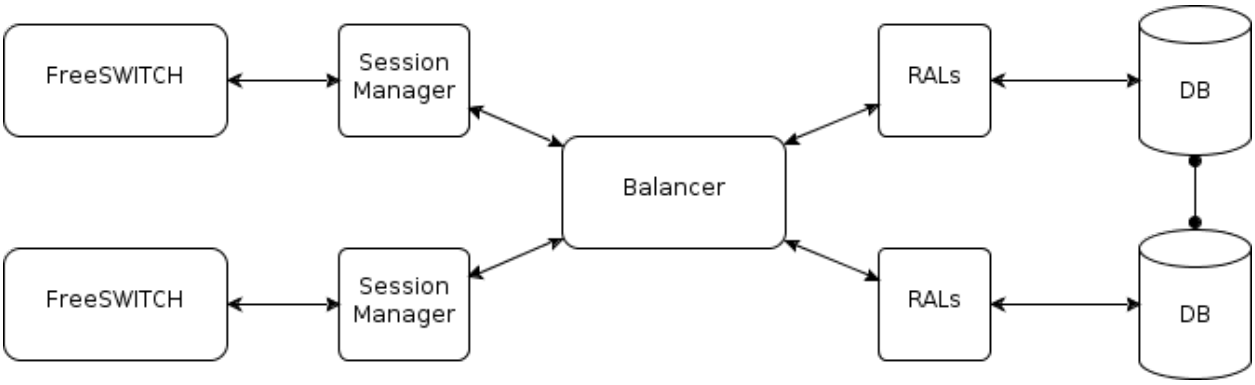
This scenario fits most of the simple installations.



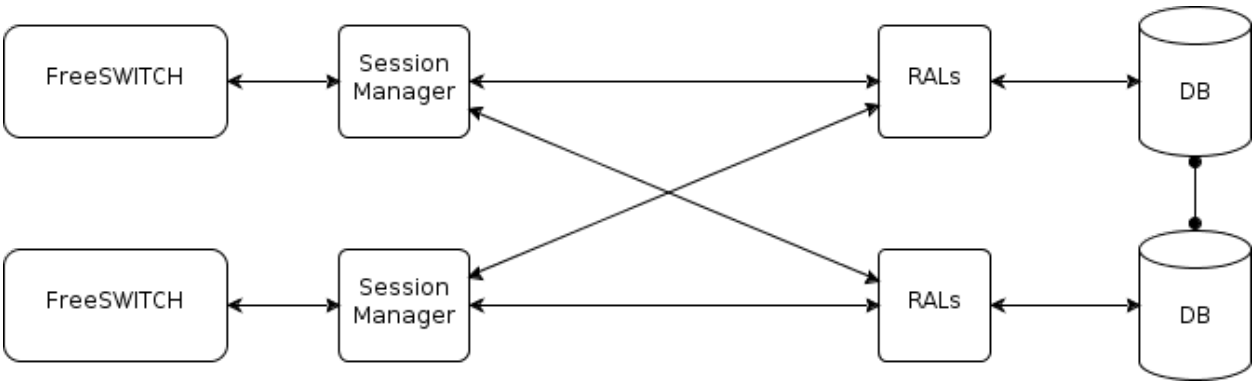
While the network grows more **RALs** can be thrown into the stack to offer more requests per seconds workload. This implies the usage of the **Balancer** to distribute the requests to the **RALs** running on the *different machines*.



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. 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, INTERNET_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.

1.2. 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

1.3. License

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

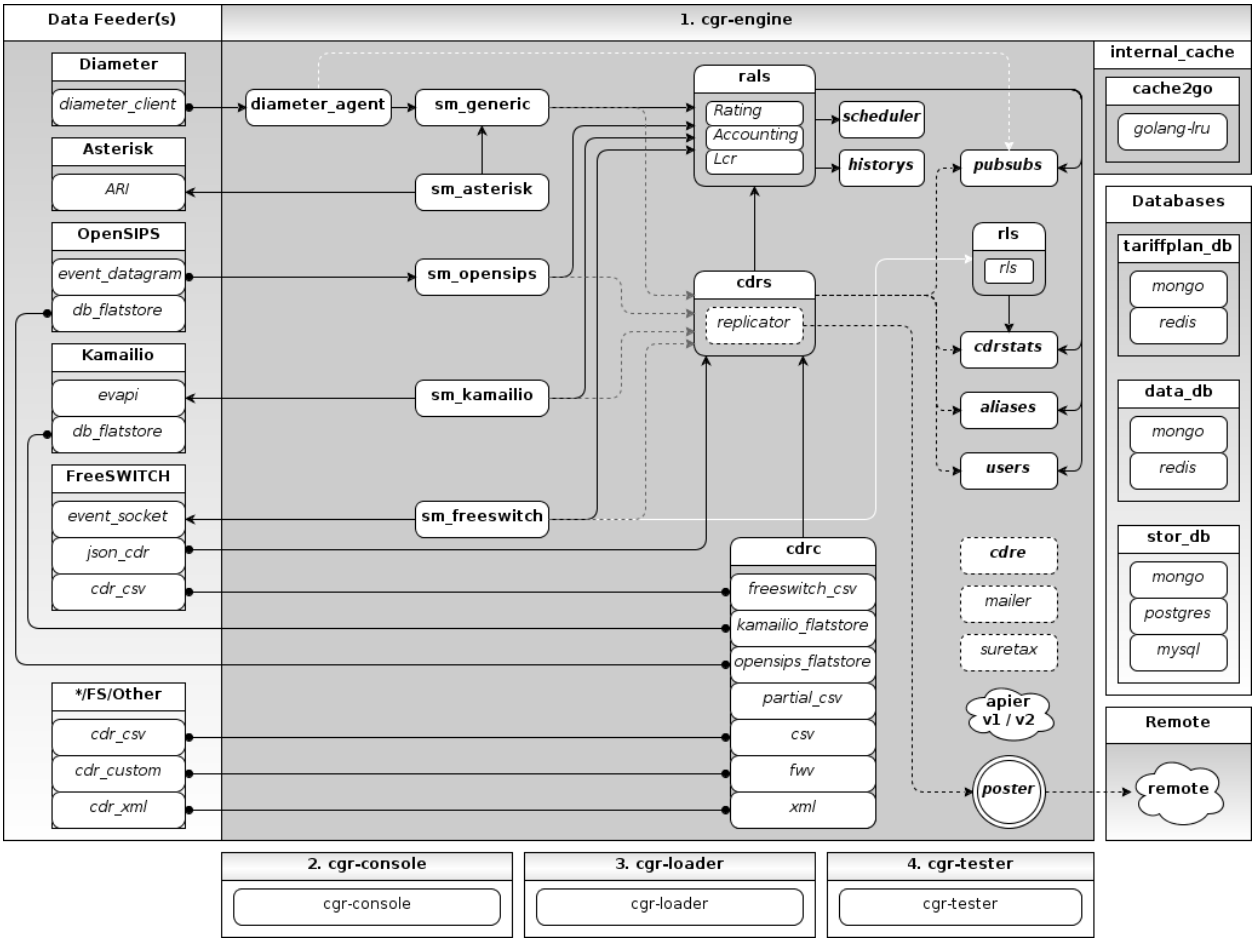


Fig. 1.1: CGRateS high level design

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
```

```
  -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.

Right now there are five session manager types.

- sm_freeswitch
- sm_kamailio
- sm_opensips
- sm_asterisk
- **sm_generic**

- **Communicates via:**
  - RPC
  - internal/in-process *within the same running* **cgr-engine** process.
- Operates with the following CGRateS database(s):

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

- **Config section in the CGRateS configuration file:**
  - `"sm_freeswitch": {...}`
  - `"sm_kamailio": {...}`
  - `"sm_opensips": {...}`
  - `"sm_asterisk": {...}`
  - `"sm_generic": {...}`

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. CDRS 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. History service

Archives rate changes in human readable JSON format using **GIT**.

- **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:**
  - `"historys": {...}`

2.1.9. 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": {...}`

2.1.10. 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": {...}`

2.1.11. 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": {...}`

2.1.12. 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": {...}`

2.1.13. 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

2.1.14. 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": {...}`

2.1.15. Mailer

TBD

- Communicates via:
- Operates with the following CGRateS database(s):
- **Config section in the CGRateS configuration file:**
  - `"mailer": {...}`

2.1.16. 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.

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
  -history_server string
        The history server address:port, empty to disable automatic history archiving␣
↪(default "127.0.0.1:2013")
  -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
        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”

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

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

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.
```

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.

3.1. Using packages

3.1.1. Debian Jessie/Wheezy

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:

```
cd /etc/apt/sources.list.d/
wget -O - http://apt.itsyscom.com/conf/cgrates.gpg.key|apt-key add -
wget http://apt.itsyscom.com/conf/cgrates.apt.list
apt-get update
apt-get install cgrates
```

Once the installation is completed, one should perform the *3.3. Post-install* 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**.

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.

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.

3.3. 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.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.

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.

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**.

```
{

// Real-time Online/Offline Charging System (OCS) for Telecom & ISP environments
// Copyright (C) ITsysCOM GmbH
//
// This file contains the default configuration hardcoded into CGRateS.
// This is what you get when you load CGRateS with an empty configuration file.

// "general": {
//         "instance_id": "",
↪                                                                                //␣
↪identifier of this instance in the cluster, if empty it will be autogenerated
//         "log_level": 6,
↪                                                                                /
↪/ control the level of messages logged (0-emerg to 7-debug)
//         "http_skip_tls_verify": false,
↪                                                    // if enabled Http Client␣
↪will accept any TLS certificate
//         "rounding_decimals": 5,
↪                                                                  // system␣
↪level precision for floats
//         "dbdata_encoding": "msgpack",
↪                                                    // encoding used to store␣
↪object data in strings: <msgpack|json>
//         "tpexport_dir": "/var/spool/cgrates/tpe",                             //
↪ path towards export folder for offline Tariff Plans
//         "poster_attempts": 3,
↪                                                                  // number␣
↪of attempts before considering post request failed (eg: *call_url, CDR replication)
//         "failed_posts_dir": "/var/spool/cgrates/failed_posts",        // directory␣
↪path where we store failed requests
//         "default_request_type": "*rated",
↪                                                // default request type to consider␣
↪when missing from requests: <""|*prepaid|*postpaid|*pseudoprepaid|*rated>
//         "default_category": "call",
↪                                                             // default category␣
↪to consider when missing from requests
//         "default_tenant": "cgrates.org",
↪                                                // default tenant to consider when␣
↪missing from requests
```

```
21 //         "default_timezone": "Local",
   ↪                                                          // default timezone for␣
   ↪timestamps where not specified <""|UTC|Local|$IANA_TZ_DB>
22 //         "connect_attempts": 3,
   ↪                                                                  // initial␣
   ↪server connect attempts
23 //         "reconnects": -1,
   ↪                                                                      //␣
   ↪number of retries in case of connection lost
24 //         "connect_timeout": "1s",
   ↪                                                        // consider␣
   ↪connection unsuccessful on timeout, 0 to disable the feature
25 //         "reply_timeout": "2s",
   ↪                                                                  // consider␣
   ↪connection down for replies taking longer than this value
26 //         "response_cache_ttl": "0s",
   ↪                                                        // the life span of␣
   ↪a cached response
27 //         "internal_ttl": "2m",
   ↪                                                                  // maximum␣
   ↪duration to wait for internal connections before giving up
28 //         "locking_timeout": "5s",
   ↪                                                        // timeout internal␣
   ↪locks to avoid deadlocks
29 // },
30
31
32 // "cache":{
33 //         "destinations": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control destination caching
34 //         "reverse_destinations": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪        // control reverse destinations index caching
35 //         "rating_plans": {"limit": 10000, "ttl":"0s","precache": true},
   ↪                                // control rating plans caching
36 //         "rating_profiles": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control rating profiles caching
37 //         "lcr": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                                                  // control lcr rules caching
38 //         "cdr_stats": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                                // control cdr stats queues caching
39 //         "actions": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                                        // control actions caching
40 //         "action_plans": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control action plans caching
41 //         "account_action_plans": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪        // control account action plans index caching
42 //         "action_triggers": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control action triggers caching
43 //         "shared_groups": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control shared groups caching
44 //         "aliases": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                                        // control aliases caching
45 //         "reverse_aliases": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control reverse aliases index caching
46 //         "derived_chargers": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪               // control derived charging rule caching
47 //         "resource_limits": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                        // control resource limits caching
```

```
48 //         "timings": {"limit": 10000, "ttl":"0s", "precache": false},
   ↪                                    // control timings caching
49 // },
50
51
52 // "listen": {
53 //         "rpc_json": "127.0.0.1:2012",                        // RPC JSON listening␣
   ↪address
54 //         "rpc_gob": "127.0.0.1:2013",                        // RPC GOB listening␣
   ↪address
55 //         "http": "127.0.0.1:2080",                             // HTTP listening␣
   ↪address
56 // },
57
58
59 // "http": {                                                                    //
   ↪ HTTP server configuration
60 //         "json_rpc_url": "/jsonrpc",                             // JSON RPC␣
   ↪relative URL ("" to disable)
61 //         "ws_url": "/ws",                                                 //␣
   ↪WebSockets relative URL ("" to disable)
62 //         "use_basic_auth": false,                              // use basic␣
   ↪authentication
63 //         "auth_users": {}                                                 // basic␣
   ↪authentication usernames and base64-encoded passwords (eg: { "username1":
   ↪"cGFzc3dvcmQ=", "username2": "cGFzc3dvcmQy "})
64 // },
65
66
67 // "data_db": {                                                                 //␣
   ↪database used to store runtime data (eg: accounts, cdr stats)
68 //         "db_type": "redis",                                               // data_
   ↪db type: <redis|mongo>
69 //         "db_host": "127.0.0.1",                                        // data_db␣
   ↪host address
70 //         "db_port": 6379,                                                  // data_
   ↪db port to reach the database
71 //         "db_name": "10",                                                  // data_
   ↪db database name to connect to
72 //         "db_user": "cgrates",                                          // username␣
   ↪to use when connecting to data_db
73 //         "db_password": "",                                                //␣
   ↪password to use when connecting to data_db
74 //         "load_history_size": 10,                              // Number of␣
   ↪records in the load history
75 // },
76
77
78 // "stor_db": {                                                                 //␣
   ↪database used to store offline tariff plans and CDRs
79 //         "db_type": "mysql",                                               // stor␣
   ↪database type to use: <mongo|mysql|postgres>
80 //         "db_host": "127.0.0.1",                                        // the host␣
   ↪to connect to
81 //         "db_port": 3306,                                                  // the␣
   ↪port to reach the stordb
82 //         "db_name": "cgrates",                                          // stor␣
   ↪database name
```

```
83  //         "db_user": "cgrates",                                        // username␣
    ↪to use when connecting to stordb
84  //         "db_password": "",                                             //␣
    ↪password to use when connecting to stordb
85  //         "max_open_conns": 100,                                        // maximum␣
    ↪database connections opened, not applying for mongo
86  //         "max_idle_conns": 10,                                        // maximum␣
    ↪database connections idle, not applying for mongo
87  //         "conn_max_lifetime": 0,                                  // maximum amount␣
    ↪of time in seconds a connection may be reused (0 for unlimited), not applying for␣
    ↪mongo
88  //         "cdrs_indexes": [],                                            //␣
    ↪indexes on cdrs table to speed up queries, used only in case of mongo
89  // },
90
91
92  // "rals": {
93  //         "enabled": false,                                          // enable␣
    ↪Rater service: <true|false>
94  //         "cdrstats_conns": [],                                      // address␣
    ↪where to reach the cdrstats service, empty to disable stats functionality: <"
    ↪"|*internal|x.y.z.y:1234>
95  //         "historys_conns": [],                                      // address␣
    ↪where to reach the history service, empty to disable history functionality: <"
    ↪"|*internal|x.y.z.y:1234>
96  //         "pubsubs_conns": [],                                     // address␣
    ↪where to reach the pubusb service, empty to disable pubsub functionality: <"
    ↪"|*internal|x.y.z.y:1234>
97  //         "users_conns": [],                                             //␣
    ↪address where to reach the user service, empty to disable user profile␣
    ↪functionality: <""|*internal|x.y.z.y:1234>
98  //         "aliases_conns": [],                                     // address␣
    ↪where to reach the aliases service, empty to disable aliases functionality: <"
    ↪"|*internal|x.y.z.y:1234>
99  //         "rp_subject_prefix_matching": false,        // enables prefix matching for␣
    ↪the rating profile subject
100 //         "lcr_subject_prefix_matching": false        // enables prefix matching for␣
    ↪the lcr subject
101 // },
102
103
104 // "scheduler": {
105 //         "enabled": false,                                          // start␣
    ↪Scheduler service: <true|false>
106 // },
107
108
109 // "cdrs": {
110 //         "enabled": false,                                          // start␣
    ↪the CDR Server service:  <true|false>
111 //         "extra_fields": [],                                            //␣
    ↪extra fields to store in CDRs for non-generic CDRs
112 //         "store_cdrs": true,                                            //␣
    ↪store cdrs in storDb
113 //         "cdr_account_summary": false,                          // add account␣
    ↪information from dataDB
114 //         "sm_cost_retries": 5,                                      // number of␣
    ↪queries to sm_costs before recalculating CDR
```

```
115 //         "rals_conns": [
116 //                 {"address": "*internal"}                                  // address where␣
    ↪to reach the Rater for cost calculation, empty to disable functionality: <"
    ↪"|*internal|x.y.z.y:1234>
117 //         ],
118 //         "pubsubs_conns": [],                                              // address␣
    ↪where to reach the pubusb service, empty to disable pubsub functionality: <"
    ↪"|*internal|x.y.z.y:1234>
119 //         "users_conns": [],                                                //␣
    ↪address where to reach the user service, empty to disable user profile␣
    ↪functionality: <""|*internal|x.y.z.y:1234>
120 //         "aliases_conns": [],                                              // address␣
    ↪where to reach the aliases service, empty to disable aliases functionality: <"
    ↪"|*internal|x.y.z.y:1234>
121 //         "cdrstats_conns": [],                                             // address␣
    ↪where to reach the cdrstats service, empty to disable stats functionality: <"
    ↪"|*internal|x.y.z.y:1234>
122 //         "online_cdr_exports":[],                                          // list of CDRE␣
    ↪profiles to use for real-time CDR exports
123 // },
124
125
126 // "cdre": {
127 //         "*default": {
128 //                 "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>
129 //                 "export_path": "/var/spool/cgrates/cdre",                 // path␣
    ↪where the exported CDRs will be placed
130 //                 "cdr_filter": "",
    ↪                                                                         // filter CDRs␣
    ↪exported by this template
131 //                 "synchronous": false,
    ↪                                                         // block processing until␣
    ↪export has a result
132 //                 "attempts": 1,
    ↪                                                                         // Number␣
    ↪of attempts if not success
133 //                 "field_separator": ",",
    ↪                                                         // used field separator in␣
    ↪some export formats, eg: *file_csv
134 //                 "usage_multiply_factor": {
135 //                         "*any":␣
    ↪1                                                                        //␣
    ↪multiply usage based on ToR field or *any for all
136 //                 },
137 //                 "cost_multiply_factor": 1,
    ↪                                                 // multiply cost before export, eg:␣
    ↪add VAT
138 //                 "header_fields": [],
    ↪                                                         // template of the exported␣
    ↪header fields
139 //                 "content_fields":␣
    ↪[                                                                        // template of the␣
    ↪exported content fields
140 //                         {"tag": "CGRID", "type": "*composed", "value": "CGRID"},
141 //                         {"tag":"RunID", "type": "*composed", "value": "RunID"},
142 //                         {"tag":"TOR", "type": "*composed", "value": "ToR"},
```

```
143 //                          {"tag":"OriginID", "type": "*composed", "value": "OriginID
    ↪"},
144 //                          {"tag":"RequestType", "type": "*composed", "value":
    ↪"RequestType"},
145 //                          {"tag":"Direction", "type": "*composed", "value":
    ↪"Direction"},
146 //                          {"tag":"Tenant", "type": "*composed", "value": "Tenant"},
147 //                          {"tag":"Category", "type": "*composed", "value": "Category
    ↪"},
148 //                          {"tag":"Account", "type": "*composed", "value": "Account"},
149 //                          {"tag":"Subject", "type": "*composed", "value": "Subject"},
150 //                          {"tag":"Destination", "type": "*composed", "value":
    ↪"Destination"},
151 //                          {"tag":"SetupTime", "type": "*composed", "value":
    ↪"SetupTime", "layout": "2006-01-02T15:04:05Z07:00"},
152 //                          {"tag":"AnswerTime", "type": "*composed", "value":
    ↪"AnswerTime", "layout": "2006-01-02T15:04:05Z07:00"},
153 //                          {"tag":"Usage", "type": "*composed", "value": "Usage"},
154 //                          {"tag":"Cost", "type": "*composed", "value": "Cost",
    ↪"rounding_decimals": 4},
155 //                  ],
156 //                  "trailer_fields": [],
    ↪                                                       // template of the exported␣
    ↪trailer fields
157 //          },
158 // },
159
160
161 // "cdrstats": {
162 //          "enabled": false,                                                  // starts␣
    ↪the cdrstats service: <true|false>
163 //          "save_interval": "1m",                                           // interval␣
    ↪to save changed stats into dataDb storage
164 // },
165
166
167 // "cdrc": [
168 //          {
169 //                  "id": "*default",
    ↪                                                               // identifier of␣
    ↪the CDRC runner
170 //                  "enabled": false,
    ↪                                                               // enable CDR␣
    ↪client functionality
171 //                  "dry_run": false,
    ↪                                                               // do not send the␣
    ↪CDRs to CDRS, just parse them
172 //                  "cdrs_conns": [
173 //                          {"address": "*internal"}
    ↪                                       // address where to reach CDR server.
    ↪<*internal|x.y.z.y:1234>
174 //                  ],
175 //                  "cdr_format": "csv",
    ↪                                                       // CDR file format
    ↪<csv|freeswitch_csv|fwv|opensips_flatstore|partial_csv>
176 //                  "field_separator": ",",
    ↪                                                       // separator used in case␣
    ↪of csv files
```

```
177 //                  "timezone": "",
    ↪                                                                    // timezone␣
    ↪for timestamps where not specified <""|UTC|Local|$IANA_TZ_DB>
178 //                  "run_delay": 0,
    ↪                                                                    // sleep␣
    ↪interval in seconds between consecutive runs, 0 to use automation via inotify
179 //                  "max_open_files": 1024,
    ↪                                                    // maximum simultaneous␣
    ↪files to process, 0 for unlimited
180 //                  "data_usage_multiply_factor": 1024,
    ↪                                 // conversion factor for data usage
181 //                  "cdr_in_dir": "/var/spool/cgrates/cdrc/in",                 //␣
    ↪absolute path towards the directory where the CDRs are stored
182 //                  "cdr_out_dir": "/var/spool/cgrates/cdrc/out",        // absolute␣
    ↪path towards the directory where processed CDRs will be moved
183 //                  "failed_calls_prefix": "missed_calls",                          //␣
    ↪used in case of flatstore CDRs to avoid searching for BYE records
184 //                  "cdr_path": "",
    ↪                                                                    // path␣
    ↪towards one CDR element in case of XML CDRs
185 //                  "cdr_source_id": "freeswitch_csv",                                  /
    ↪/ free form field, tag identifying the source of the CDRs within CDRS database
186 //                  "cdr_filter": "",
    ↪                                                          // filter CDR␣
    ↪records to import
187 //                  "continue_on_success": false,
    ↪                                     // continue to the next template if executed
188 //                  "partial_record_cache": "10s",
    ↪                                     // duration to cache partial records when␣
    ↪not pairing
189 //                  "partial_cache_expiry_action": "*dump_to_file",         // action␣
    ↪taken when cache when records in cache are timed-out <*dump_to_file|*post_cdr>
190 //                  "header_fields": [],
    ↪                                                    // template of the import␣
    ↪header fields
191 //                  "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
192 //                      {"tag": "TOR", "field_id": "ToR", "type": "*composed",
    ↪"value": "2", "mandatory": true},
193 //                      {"tag": "OriginID", "field_id": "OriginID", "type":
    ↪"*composed", "value": "3", "mandatory": true},
194 //                      {"tag": "RequestType", "field_id": "RequestType", "type":
    ↪"*composed", "value": "4", "mandatory": true},
195 //                      {"tag": "Direction", "field_id": "Direction", "type":
    ↪"*composed", "value": "5", "mandatory": true},
196 //                      {"tag": "Tenant", "field_id": "Tenant", "type": "*composed
    ↪", "value": "6", "mandatory": true},
197 //                      {"tag": "Category", "field_id": "Category", "type":
    ↪"*composed", "value": "7", "mandatory": true},
198 //                      {"tag": "Account", "field_id": "Account", "type":
    ↪"*composed", "value": "8", "mandatory": true},
199 //                      {"tag": "Subject", "field_id": "Subject", "type":
    ↪"*composed", "value": "9", "mandatory": true},
200 //                      {"tag": "Destination", "field_id": "Destination", "type":
    ↪"*composed", "value": "10", "mandatory": true},
201 //                      {"tag": "SetupTime", "field_id": "SetupTime", "type":
    ↪"*composed", "value": "11", "mandatory": true},
```

```
202 //                          {"tag": "AnswerTime", "field_id": "AnswerTime", "type":
    ↪"*composed", "value": "12", "mandatory": true},
203 //                          {"tag": "Usage", "field_id": "Usage", "type": "*composed",
    ↪"value": "13", "mandatory": true},
204 //                  ],
205 //                  "trailer_fields": [],
    ↪                                                   // template of the import␣
    ↪trailer fields
206 //                  "cache_dump_fields":␣
    ↪[                                                   // template used when␣
    ↪dumping cached CDR, eg: partial CDRs
207 //                          {"tag": "CGRID", "type": "*composed", "value": "CGRID"},
208 //                          {"tag": "RunID", "type": "*composed", "value": "RunID"},
209 //                          {"tag": "TOR", "type": "*composed", "value": "ToR"},
210 //                          {"tag": "OriginID", "type": "*composed", "value": "OriginID
    ↪"},
211 //                          {"tag": "RequestType", "type": "*composed", "value":
    ↪"RequestType"},
212 //                          {"tag": "Direction", "type": "*composed", "value":
    ↪"Direction"},
213 //                          {"tag": "Tenant", "type": "*composed", "value": "Tenant"},
214 //                          {"tag": "Category", "type": "*composed", "value": "Category
    ↪"},
215 //                          {"tag": "Account", "type": "*composed", "value": "Account"}
    ↪,
216 //                          {"tag": "Subject", "type": "*composed", "value": "Subject"}
    ↪,
217 //                          {"tag": "Destination", "type": "*composed", "value":
    ↪"Destination"},
218 //                          {"tag": "SetupTime", "type": "*composed", "value":
    ↪"SetupTime", "layout": "2006-01-02T15:04:05Z07:00"},
219 //                          {"tag": "AnswerTime", "type": "*composed", "value":
    ↪"AnswerTime", "layout": "2006-01-02T15:04:05Z07:00"},
220 //                          {"tag": "Usage", "type": "*composed", "value": "Usage"},
221 //                          {"tag": "Cost", "type": "*composed", "value": "Cost"},
222 //                  ],
223 //          },
224 // ],
225
226
227 // "sm_generic": {
228 //          "enabled": false,                                                  // starts␣
    ↪SessionManager service: <true|false>
229 //          "listen_bijson": "127.0.0.1:2014",                // address where to␣
    ↪listen for bidirectional JSON-RPC requests
230 //          "rals_conns": [
231 //                  {"address": "*internal"}                          // address where␣
    ↪to reach the Rater <""|*internal|127.0.0.1:2013>
232 //          ],
233 //          "cdrs_conns": [
234 //                  {"address": "*internal"}                          // address where␣
    ↪to reach CDR Server, empty to disable CDR capturing <*internal|x.y.z.y:1234>
235 //          ],
236 //          "smg_replication_conns": [],                          // replicate sessions␣
    ↪towards these SMGs
237 //          "debit_interval": "0s",                                       // interval␣
    ↪to perform debits on.
238 //          "min_call_duration": "0s",                                  // only␣
    ↪authorize calls with allowed duration higher than this
```

```
239 //         "max_call_duration": "3h",                                    // maximum call␣
    ↪duration a prepaid call can last
240 //         "session_ttl": "0s",                                           // time after␣
    ↪a session with no updates is terminated, not defined by default
241 //         //"session_ttl_max_delay": "",                             // activates session_
    ↪ttl randomization and limits the maximum possible delay
242 //         //"session_ttl_last_used": "",                             // tweak LastUsed␣
    ↪for sessions timing-out, not defined by default
243 //         //"session_ttl_usage": "",                                   // tweak Usage␣
    ↪for sessions timing-out, not defined by default
244 //         "session_indexes": [],                                        // index␣
    ↪sessions based on these fields for GetActiveSessions API
245 // },
246
247
248 // "sm_asterisk": {
249 //         "enabled": false,                                               // starts␣
    ↪Asterisk SessionManager service: <true|false>
250 //         "create_cdr": false,                                          // create CDR␣
    ↪out of events and sends it to CDRS component
251 //         "asterisk_conns":[                                             //␣
    ↪instantiate connections to multiple Asterisk servers
252 //                 {"address": "127.0.0.1:8088", "user": "cgrates", "password":
    ↪"CGRateS.org", "connect_attempts": 3,"reconnects": 5}
253 //         ],
254 // },
255
256
257 // "sm_freeswitch": {
258 //         "enabled": false,                                               // starts␣
    ↪SessionManager service: <true|false>
259 //         "rals_conns": [
260 //                 {"address": "*internal"}                          // address where␣
    ↪to reach the Rater <""|*internal|127.0.0.1:2013>
261 //         ],
262 //         "cdrs_conns": [
263 //                 {"address": "*internal"}                          // address where␣
    ↪to reach CDR Server, empty to disable CDR capturing <*internal|x.y.z.y:1234>
264 //         ],
265 //         "rls_conns": [],                                                // address␣
    ↪where to reach the ResourceLimiter service, empty to disable functionality: <"
    ↪"|*internal|x.y.z.y:1234>
266 //         "create_cdr": false,                                          // create CDR␣
    ↪out of events and sends them to CDRS component
267 //         "extra_fields": [],                                             //␣
    ↪extra fields to store in auth/CDRs when creating them
268 //         "debit_interval": "10s",                                    // interval to␣
    ↪perform debits on.
269 //         "min_call_duration": "0s",                                    // only␣
    ↪authorize calls with allowed duration higher than this
270 //         "max_call_duration": "3h",                                    // maximum call␣
    ↪duration a prepaid call can last
271 //         "min_dur_low_balance": "5s",                              // threshold which␣
    ↪will trigger low balance warnings for prepaid calls (needs to be lower than debit_
    ↪interval)
272 //         "low_balance_ann_file": "",                                    // file to be␣
    ↪played when low balance is reached for prepaid calls
273 //         "empty_balance_context": "",                              // if defined, prepaid␣
    ↪calls will be transfered to this context on empty balance
```

```
274 //         "empty_balance_ann_file": "",                           // file to be played␣
    ↪before disconnecting prepaid calls on empty balance (applies only if no context␣
    ↪defined)
275 //         "subscribe_park": true,                                         //␣
    ↪subscribe via fsock to receive park events
276 //         "channel_sync_interval": "5m",                          // sync channels␣
    ↪with freeswitch regularly
277 //         "max_wait_connection": "2s",                           // maximum duration to␣
    ↪wait for a connection to be retrieved from the pool
278 //         "event_socket_conns":[                                          //␣
    ↪instantiate connections to multiple FreeSWITCH servers
279 //                 {"address": "127.0.0.1:8021", "password": "ClueCon", "reconnects":␣
    ↪5}
280 //         ],
281 // },
282
283
284 // "sm_kamailio": {
285 //         "enabled": false,                                               // starts␣
    ↪SessionManager service: <true|false>
286 //         "rals_conns": [
287 //                 {"address": "*internal"}                        // address where␣
    ↪to reach the Rater <""|*internal|127.0.0.1:2013>
288 //         ],
289 //         "cdrs_conns": [
290 //                 {"address": "*internal"}                        // address where␣
    ↪to reach CDR Server, empty to disable CDR capturing <*internal|x.y.z.y:1234>
291 //         ],
292 //         "rls_conns": [],                                                // address␣
    ↪where to reach the ResourceLimiter service, empty to disable functionality: <"
    ↪"|*internal|x.y.z.y:1234>
293 //         "create_cdr": false,                                    // create CDR␣
    ↪out of events and sends them to CDRS component
294 //         "debit_interval": "10s",                                // interval to␣
    ↪perform debits on.
295 //         "min_call_duration": "0s",                                // only␣
    ↪authorize calls with allowed duration higher than this
296 //         "max_call_duration": "3h",                                // maximum call␣
    ↪duration a prepaid call can last
297 //         "evapi_conns":[                                                 //␣
    ↪instantiate connections to multiple Kamailio servers
298 //                 {"address": "127.0.0.1:8448", "reconnects": 5}
299 //         ],
300 // },
301
302
303 // "sm_opensips": {
304 //         "enabled": false,                                               // starts␣
    ↪SessionManager service: <true|false>
305 //         "listen_udp": "127.0.0.1:2020",                         // address where to␣
    ↪listen for datagram events coming from OpenSIPS
306 //         "rals_conns": [
307 //                 {"address": "*internal"}                        // address where␣
    ↪to reach the Rater <""|*internal|127.0.0.1:2013>
308 //         ],
309 //         "cdrs_conns": [
310 //                 {"address": "*internal"}                        // address where␣
    ↪to reach CDR Server, empty to disable CDR capturing <*internal|x.y.z.y:1234>
```

```
311 //         ],
312 //         "create_cdr": false,                                    // create CDR␣
    ↪out of events and sends it to CDRS component
313 //         "debit_interval": "10s",                               // interval to␣
    ↪perform debits on.
314 //         "min_call_duration": "0s",                              // only␣
    ↪authorize calls with allowed duration higher than this
315 //         "max_call_duration": "3h",                              // maximum call␣
    ↪duration a prepaid call can last
316 //         "events_subscribe_interval": "60s",                 // automatic events␣
    ↪subscription to OpenSIPS, 0 to disable it
317 //         "mi_addr": "127.0.0.1:8020",                         // address where to␣
    ↪reach OpenSIPS MI to send session disconnects
318 // },
319
320
321 // "diameter_agent": {
322 //         "enabled": false,
    ↪                                                                              /
    ↪/ enables the diameter agent: <true|false>
323 //         "listen": "127.0.0.1:3868",
    ↪                                                                  // address␣
    ↪where to listen for diameter requests <x.y.z.y:1234>
324 //         "dictionaries_dir": "/usr/share/cgrates/diameter/dict/",        // path␣
    ↪towards directory holding additional dictionaries to load
325 //         "sm_generic_conns": [
326 //                 {"address": "*internal"}
    ↪                                                              // connection␣
    ↪towards SMG component for session management
327 //         ],
328 //         "pubsubs_conns": [],
    ↪                                                                          //␣
    ↪address where to reach the pubusb service, empty to disable pubsub functionality: <"
    ↪"|*internal|x.y.z.y:1234>
329 //         "create_cdr": true,
    ↪                                                                              /
    ↪/ create CDR out of CCR terminate and send it to SMG component
330 //         "cdr_requires_session": true,
    ↪                                                              // only create CDR␣
    ↪if there is an active session at terminate
331 //         "debit_interval": "5m",
    ↪                                                                          //␣
    ↪interval for CCR updates
332 //         "timezone": "",
    ↪                                                                                /
    ↪/ timezone for timestamps where not specified, empty for general defaults <"
    ↪"|UTC|Local|$IANA_TZ_DB>
333 //         "origin_host": "CGR-DA",
    ↪                                                                  // diameter␣
    ↪Origin-Host AVP used in replies
334 //         "origin_realm": "cgrates.org",
    ↪                                                              // diameter Origin-
    ↪Realm AVP used in replies
335 //         "vendor_id": 0,
    ↪                                                                               /
    ↪/ diameter Vendor-Id AVP used in replies
336 //         "product_name": "CGRateS",
    ↪                                                                  // diameter␣
    ↪Product-Name AVP used in replies
```

```
337 //         "request_processors": [],
338 // },
339
340
341 // "radius_agent": {
342 //         "enabled": false,
    ↪                                                                                                    /
    ↪/ enables the radius agent: <true|false>
343 //         "listen_net": "udp",
    ↪                                                                                                   //␣
    ↪network to listen on <udp|tcp>
344 //         "listen_auth": "127.0.0.1:1812",
    ↪                                                          // address where to listen␣
    ↪for radius authentication requests <x.y.z.y:1234>
345 //         "listen_acct": "127.0.0.1:1813",
    ↪                                                          // address where to listen␣
    ↪for radius accounting requests <x.y.z.y:1234>
346 //         "client_secrets":
    ↪{                                                                                                   /
    ↪/ hash containing secrets for clients connecting here <*default|$client_ip>
347 //                 "*default": "CGRateS.org"
348 //         },
349 //         "client_dictionaries":
    ↪{                                                                                 // per␣
    ↪client path towards directory holding additional dictionaries to load (extra to RFC)
350 //                 "*default": "/usr/share/cgrates/radius/dict/",
    ↪                        // key represents the client IP or catch-all <*default|
    ↪$client_ip>
351 //         },
352 //         "sm_generic_conns": [
353 //                 {"address": "*internal"}
    ↪                                                        // connection␣
    ↪towards SMG component for session management
354 //         ],
355 //         "create_cdr": true,
    ↪                                                                                                    /
    ↪/ create CDR out of Accounting-Stop and send it to SMG component
356 //         "cdr_requires_session": false,
    ↪                                                        // only create CDR␣
    ↪if there is an active session at terminate
357 //         "timezone": "",
    ↪                                                                                                      /
    ↪/ timezone for timestamps where not specified, empty for general defaults <"
    ↪"|UTC|Local|$IANA_TZ_DB>
358 //         "request_processors": [],
359 // },
360
361
362 // "historys": {
363 //         "enabled": false,                                                                  //
    ↪ starts History service: <true|false>.
364 //         "history_dir": "/var/lib/cgrates/history",        // location on disk␣
    ↪where to store history files.
365 //         "save_interval": "1s",                                                            //␣
    ↪interval to save changed cache into .git archive
366 // },
367
368
```

```
// "pubsubs": {
//         "enabled": false,                                                    // starts PubSub service: <true|false>.
// },


// "aliases": {
//         "enabled": false,                                                    // starts Aliases service: <true|false>.
// },


// "users": {
//         "enabled": false,                                                    // starts User service: <true|false>.
//         "indexes": [],                                                       // user profile field indexes
// },


// "rls": {
//         "enabled": false,                                                    // starts ResourceLimiter service: <true|false>.
//         "cdrstats_conns": [],                                                // address where to reach the cdrstats service, empty to disable stats functionality: <""|*internal|x.y.z.y:1234>
//         "cache_dump_interval": "0s",                                         // dump cache regularly to dataDB, 0 - dump at start/shutdown: <""|*never|$dur>
// },


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


// "suretax": {
//         "url": "",                                                           // API url
//         "client_number": "",                                                 // client number, provided by SureTax
//         "validation_key": "",                                                // validation key provided by SureTax
//         "business_unit": "",                                                 // client's Business Unit
//         "timezone": "Local",                                                 // convert the time of the events to this timezone before sending request out <UTC|Local|$IANA_TZ_DB>
```

```
406 //         "include_local_cost": false,                          // sum local␣
    ↪calculated cost with tax one in final cost
407 //         "return_file_code": "0",                                 // default or␣
    ↪Quote purposes <0|Q>
408 //         "response_group": "03",                                      //␣
    ↪determines how taxes are grouped for the response <03|13>
409 //         "response_type": "D4",                                     //␣
    ↪determines the granularity of taxes and (optionally) the decimal precision for the␣
    ↪tax calculations and amounts in the response
410 //         "regulatory_code": "03",                                 // provider type
411 //         "client_tracking": "CGRID",                                 // template␣
    ↪extracting client information out of StoredCdr; <$RSRFields>
412 //         "customer_number": "Subject",                          // template␣
    ↪extracting customer number out of StoredCdr; <$RSRFields>
413 //         "orig_number":  "Subject",                                  // template␣
    ↪extracting origination number out of StoredCdr; <$RSRFields>
414 //         "term_number": "Destination",                          // template␣
    ↪extracting termination number out of StoredCdr; <$RSRFields>
415 //         "bill_to_number": "",                                        // template␣
    ↪extracting billed to number out of StoredCdr; <$RSRFields>
416 //         "zipcode": "",                                                //␣
    ↪template extracting billing zip code out of StoredCdr; <$RSRFields>
417 //         "plus4": "",                                                 //␣
    ↪template extracting billing zip code extension out of StoredCdr; <$RSRFields>
418 //         "p2pzipcode": "",                                             //␣
    ↪template extracting secondary zip code out of StoredCdr; <$RSRFields>
419 //         "p2pplus4": "",                                               //␣
    ↪template extracting secondary zip code extension out of StoredCdr; <$RSRFields>
420 //         "units": "^1",                                                //␣
    ↪template extracting number of "lines" or unique charges contained within the␣
    ↪revenue out of StoredCdr; <$RSRFields>
421 //         "unit_type": "^00",                                          //␣
    ↪template extracting number of unique access lines out of StoredCdr; <$RSRFields>
422 //         "tax_included": "^0",                                        // template␣
    ↪extracting tax included in revenue out of StoredCdr; <$RSRFields>
423 //         "tax_situs_rule": "^04",                                 // template␣
    ↪extracting tax situs rule out of StoredCdr; <$RSRFields>
424 //         "trans_type_code": "^010101",                          // template␣
    ↪extracting transaction type indicator out of StoredCdr; <$RSRFields>
425 //         "sales_type_code": "^R",                                 // template␣
    ↪extracting sales type code out of StoredCdr; <$RSRFields>
426 //         "tax_exemption_code_list": "",                          // template␣
    ↪extracting tax exemption code list out of StoredCdr; <$RSRFields>
427 // },
428
429 }
```

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_1003	1003
DST_1007	1007
DST_FS	10
DST_DE_MOBILE	+49151
DST_DE_MOBILE	+49161
DST_DE_MOBILE	+49171

[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
```

#Tag	Years	Months	MonthDays	WeekDays	Time
PEAK	*any	*any	*any	1;2;3;4;5	08:00:00
OFFPEAK_MORNING	*any	*any	*any	1;2;3;4;5	00:00:00
OFFPEAK_EVENING	*any	*any	*any	1;2;3;4;5	19:00:00
OFFPEAK_WEEKEND	*any	*any	*any	6;7	00:00:00

[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_GENERIC_1	0	1	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_1002_20CNT	DST_1002	RT_20CNT	*up	4	0	
DR_1002_10CNT	DST_1002	RT_10CNT	*up	4	0	
DR_1003_20CNT	DST_1003	RT_40CNT	*up	4	0	
DR_1003_10CNT	DST_1003	RT_10CNT	*up	4	0	
DR_FS_40CNT	DST_FS	RT_40CNT	*up	4	0	
DR_FS_10CNT	DST_FS	RT_10CNT	*up	4	0	
DR_SPECIAL_1002	DST_1002	RT_1CNT	*up	4	0	
DR_1007_MAXCOST_DISC	DST_1007	RT_1CNT_PER_SEC	*up	4	0.62	*disconnect
DR_1007_MAXCOST_FREE	DST_1007	RT_1CNT_PER_SEC	*up	4	0.62	*free
DR_GENERIC	*any	RT_GENERIC_1	*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_RETAIL1	DR_FS_40CNT	PEAK	10
RP_RETAIL1	DR_FS_10CNT	OFFPEAK_MORNING	10
RP_RETAIL1	DR_FS_10CNT	OFFPEAK_EVENING	10
RP_RETAIL1	DR_FS_10CNT	OFFPEAK_WEEKEND	10
RP_RETAIL1	DR_1007_MAXCOST_DISC	*any	10
RP_RETAIL2	DR_1002_20CNT	PEAK	10
RP_RETAIL2	DR_1003_20CNT	PEAK	10
RP_RETAIL2	DR_FS_40CNT	PEAK	10
RP_RETAIL2	DR_1002_10CNT	OFFPEAK_MORNING	10
RP_RETAIL2	DR_1002_10CNT	OFFPEAK_EVENING	10
RP_RETAIL2	DR_1002_10CNT	OFFPEAK_WEEKEND	10
RP_RETAIL2	DR_1003_10CNT	OFFPEAK_MORNING	10
RP_RETAIL2	DR_1003_10CNT	OFFPEAK_EVENING	10
RP_RETAIL2	DR_1003_10CNT	OFFPEAK_WEEKEND	10
RP_RETAIL2	DR_FS_10CNT	OFFPEAK_MORNING	10
RP_RETAIL2	DR_FS_10CNT	OFFPEAK_EVENING	10
RP_RETAIL2	DR_FS_10CNT	OFFPEAK_WEEKEND	10
RP_RETAIL2	DR_1007_MAXCOST_FREE	*any	10
RP_SPECIAL_1002	DR_SPECIAL_1002	*any	10
RP_GENERIC	DR_GENERIC	*any	10

[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	*any	2014-01-14T00:00:00Z	RP_RETAIL1		
*out	cgrates.org	call	1001	2014-01-14T00:00:00Z	RP_RETAIL2		
*out	cgrates.org	call	SPECIAL_1002	2014-01-14T00:00:00Z	RP_SPECIAL_1002		
*out	cgrates.org	cr_profile1	suppl1	2014-01-14T00:00:00Z	RP_RETAIL1		STATS_SUPPL1
*out	cgrates.org	cr_profile1	suppl2	2014-01-14T00:00:00Z	RP_RETAIL2		STATS_SUPPL2
*out	cgrates.org	cr_profile2	suppl1	2014-01-14T00:00:00Z	RP_RETAIL2		STATS_SUPPL1
*out	cgrates.org	cr_profile2	suppl2	2014-01-14T00:00:00Z	RP_RETAIL1		STATS_SUPPL2
*out	cgrates.org	cr_profile2	suppl3	2014-01-14T00:00:00Z	RP_SPECIAL_1002		
*out	cgrates.org	generic	*any	2014-01-14T00:00:00Z	RP_GENERIC		

[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	PACKAGE_1001	STANDARD_TRIGGERS		
cgrates.org	1002	PACKAGE_10	STANDARD_TRIGGERS		
cgrates.org	1003	PACKAGE_10	STANDARD_TRIGGERS		
cgrates.org	1004	PACKAGE_10	STANDARD_TRIGGERS		
cgrates.org	1007	USE_SHARED_A	STANDARD_TRIGGERS		

[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
```

#Tag[0]	UniqueID[1]	ThresholdType[2]	ThresholdValue[3]	Recurrent[4]	MinSleep[5]	ExpiryTime[6]	ActivationTime[7]	BalanceTag[8]	BalanceType[9]	BalanceDestinationIds[10]	BalanceWeight[11]	BalanceExpirationDate[12]	BalanceTimingTags[13]	BalanceRatingSubject[14]	BalanceCategories[15]	BalanceSharedGroups[16]	BalanceBlocker[17]	BalanceDisabled[18]	StatsOneTimeAction[19]	ActionsId[20]	MinQueuedItems[21]	Weight[22]
STANDARD_TRIGGERS		*min_balance	2	false	0				*monetary	*out										LOG_WARNING	10	
STANDARD_TRIGGERS		*max_event_counter	5	false	0				*monetary	*out			FS_USERS									
STANDARD_TRIGGERS		*max_balance	20	false	0				*monetary	*out										LOG_WARNING	10	
STANDARD_TRIGGERS		*max_balance	100	false	0				*monetary	*out										DISABLE_AND_LOG	10	
CDRST1_WARN		*min_asr	45	true	1m														3	LOG_WARNING	10	
CDRST1_WARN		*min_acd	120	true	1m														5	LOG_WARNING	10	
CDRST1_WARN		*max_acc	10	true	1m														5	LOG_WARNING	10	
CDRST1001_WARN		*min_asr	65	true	1m														3	LOG_WARNING	10	
CDRST1001_WARN		*min_acd	90	true	1m														5	LOG_WARNING	10	
CDRST1001_WARN		*max_acc	5	true	1m														5	LOG_WARNING	10	
CDRST3_WARN		*min_acd	90	false	1m														5	LOG_WARNING	10	

[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_account_plans" - stor_db
```

#Id	ActionsId	TimingId	Weight
PACKAGE_10	TOPUP_RST_10	*asap	10
PACKAGE_10_SHARED_A_5	TOPUP_RST_5	*asap	10
PACKAGE_10_SHARED_A_5	TOPUP_RST_SHARED_5	*asap	10
USE_SHARED_A	SHARED_A_0	*asap	10
PACKAGE_1001	TOPUP_RST_5	*asap	10
PACKAGE_1001	TOPUP_RST_SHARED_5	*asap	10
PACKAGE_1001	TOPUP_120_DST1003	*asap	10
PACKAGE_1001	TOPUP_RST_DATA_100	*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
```

#Action-sId[0]	Action[1]	Extra-Parameters[2]	Filter[3]	Balance-Id[4]	Balance-Type[5]	Direction[6]	Category[7]	Destination-Ids[8]	Rating-Subject[9]	SharedGroup[10]	ExpiryTime[11]	TimingIds[12]	Units[13]	BalanceWeight[14]	BalanceBlocker[15]	BalanceDisabled[16]	Weight[17]
TOPUP_RST_10	*topup_reset				*monetary	*out		*any			*unlimited		10	10	false	false	10
TOPUP_RST_5	*topup_reset				*monetary	*out		*any			*unlimited		5	20	false	false	10
TOPUP_RST_5	*topup_reset				*voice	*out		DST_1002	SPECIAL_1002		*unlimited		90	20	false	false	10
TOPUP_120_DST1003	*topup_reset				*voice	*out		DST_1003			*unlimited		120	20	false	false	10
TOPUP_RST_SHARED_5	*topup				*monetary	*out		*any		SHARED_A	*unlimited		5	10	false	false	10
SHARED_A_0	*topup_reset				*monetary	*out		*any		SHARED_A	*unlimited		0	10	false	false	10
TOPUP_RST_DATA_100	*topup_reset				*data	*out		*any			*unlimited		1024000	10	false	false	10
LOG_WARNING	*log														false	false	10
DISABLE_AND_LOG	*log														false	false	10
DISABLE_AND_LOG	*disable_account														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
```

#Direction[0]	Tenant[1]	Category[2]	Account[3]	Subject[4]	DestinationIds[5]	RunId[6]	RunFilter[7]	ReqTypeField[8]	DirectionField[9]	TenantField[10]	CategoryField[11]	AccountField[12]	SubjectField[13]	DestinationField[14]	SetupTimeField[15]	PddField[16]	AnswerTimeField[17]	UsageField[18]	SupplierField[19]	DisconnectCause[20]	RatedField[21]	CostField[22]
*out	cgrates.org	call	1001	1001		derived_run1		^*rated	*default	*default	*default	*default	^1002	*default	*default	*default	*default	*default	*default	*default	*default	*default

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
```

#Id[0]	QueueLength[1]	TimeWindow[2]	SaveInterval[3]	Metric[4]	SetupInterval[5]	TOR[6]	CdrHost[7]	CdrSource[8]	ReqType[9]	Direction[10]	Tenant[11]	Category[12]	Account[13]	Subject[14]	DestinationIds[15]	Pdd[16]	UsageInterval[17]	Supplier[18]	DisconnectCause[19]	RunFilters[20]	Raccount[21]	Rsubject[22]	ActivationInterval[23]	Triggers[24]
CDRST1	10	10s	ASR								cgrates.org								*default					CDRST1_WARN
CDRST1			ACD																					
CDRST1			ACC																					
CDRST1			TCD																					
CDRST1			TCC																					
CDRST_1001	10	10s	ASR								cgrates.org			1001					*default					CDRST1001_WARN
CDRST_1001			ACD																					
CDRST_1001			ACC																					
CDRST_1002	10	10s	ASR								cgrates.org			1002					*default					CDRST1001_WARN
CDRST_1002			ACD																					
CDRST_1002			ACC																					
CDRST_1003			ASR								cgrates.org				1003				*default					CDRST3_WARN
CDRST_1003			ACD																					
STATS_SUPPL1			ACD															suppl1						
STATS_SUPPL1			ASR															suppl1						
STATS_SUPPL1			ACC															suppl1						
STATS_SUPPL1			TCD															suppl1						
STATS_SUPPL1			TCC															suppl1						
STATS_SUPPL2			ACD															suppl2						
STATS_SUPPL2			ASR															suppl2						
STATS_SUPPL2			ACC															suppl2						
STATS_SUPPL2			TCD															suppl2						
STATS_SUPPL2			TCC															suppl2						

[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
```

#Id	Account	Strategy	RatingSubject
SHARED_A	*any	*highest	

[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
```

#Direction	Tenant	Category	Account	Subject	DestinationId	RPCategory	Strategy	StrategyParams	ActivationTime	Weight
*out	cgrates.org	call	1001	*any	DST_1002	lcr_profile1	*static	suppl2;suppl1	2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1001	*any	*any	lcr_profile1	*static	suppl1;suppl2	2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1002	*any	DST_1002	lcr_profile1	*highest_cost		2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1002	*any	*any	lcr_profile1	*qos		2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1003	*any	DST_1002	lcr_profile1	*qos_threshold	20;;;;2m;;;;;;;	2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1003	*any	*any	lcr_profile1	*qos_threshold	40;;;;90s;;;;;;;	2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1004	*any	DST_1002	lcr_profile1	*load_distribution	supplier1:5;supplier2:3;default:1	2014-01-14T00:00:00Z	10
*out	cgrates.org	call	1004	*any	*any	lcr_profile1	*load_distribution		2014-01-14T00:00:00Z	10
*out	cgrates.org	call	*any	*any	DST_1002	lcr_profile2	*lowest_cost		2014-01-14T00:00:00Z	10
*out	cgrates.org	call	*any	*any	*any	lcr_profile1	*lowest_cost		2014-01-14T00:00:00Z	10

[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
```

#Tenant[0]	UserName[1]	Masked[2]	AttributeName[3]	AttributeValue[4]	Weight[5]
cgrates.org	1001		SysUserName	danb	10
cgrates.org	1001		SysPassword	hisPass321	10
cgrates.org	1001		Cli	+4986517174963	10
cgrates.org	1001		Account	1001	10
cgrates.org	1001		Subject	1001	10
cgrates.org	1001		Uuid	388539dfd4f5cefee8f488b78c6c244b9e19138e	10
cgrates.org	1001		SubscriberId	1001	10
cgrates.org	1001		RequestType	*prepaid	10
cgrates.org	1002		SysUserName	rif	10
cgrates.org	1002		RifAttr	RifVal	10
cgrates.org	1002		Account	1002	10
cgrates.org	1002		Subject	1002	10
cgrates.org	1002		Uuid	27f37edec0670fa34cf79076b80ef5021e39c5bb	10
cgrates.org	1002		SubscriberId	1002	10
cgrates.org	1004		SysUserName	danb4	10
cgrates.org	1004		SysPassword	hisPass321	10
cgrates.org	1004		Cli	+4986517174964	10
cgrates.org	1004		Account	1004	10
cgrates.org	1004		Subject	1004	10
cgrates.org	1004		RequestType	*rated	10
cgrates.org	1004		SubscriberId	1004	10

[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
```

#Direction	Tenant	Category	Account	Subject	DestinationId	Context	Target	Original	Alias	Weight
*out	cgrates.org	call	1006	1006	*any	*rating	Subject	1006	1001	10
*out	cgrates.org	call	1006	1006	*any	*rating	Account	1006	1002	10

[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

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.

6. Advanced Topics

API Calls

API calls are documented in the following GoDoc

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>
- 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 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 "curl --data "tor=*voice&accid=iiaasbfdsaf&cdrhost=192.168.1.1&
↪cdrsource=curl_cdr&reqtype=rated&direction=*out&tenant=192.168.56.66&category=call&
↪account=dan&subject=dan&destination=%2B4986517174963&answer_time=1383813746&usage=1&
↪sip_user=Jitsi&subject2=1003" 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>
  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
```

}

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>]...*.

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.

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"'
```

LCR System

In voice telecommunications, least-cost routing (LCR) is the process of selecting the path of outbound communications traffic based on cost. Within a telecoms carrier, an LCR team might periodically (monthly, weekly or even daily) choose between routes from several or even hundreds of carriers for destinations across the world. This function might also be automated by a device or software program known as a "Least Cost Router." *[WIKI2015]*

Data structures

The LCR rule parameters are: Direction, Tenant, Category, Account, Subject, DestinationId, RPCategory, Strategy, StrategyParameters, ActivationTime, Weight.

The first five are used to match the rule for a specific call descriptor. They can have a value or marked as *any.

The DestinationId can be used to filter the LCR rules entries or to make the rule more specific.

RPCategory is used to indicate the rating profile category.

Strategy indicates supplier selection algorithm and StrategyParams will be specific to each strategy. Strategy can be one of the following:

***static (filter)** Will use the suppliers provided as params. StrategyParams: suppier1;supplier2;etc

***lowest_cost (sorting)** Matching suppliers will be sorted by ascending cost. StrategyParams: None

***highest_cost (sorting)** Matching suppliers will be sorted by descending cost. StrategyParams: None

***qos_with_threshold (filter)** The system will reject the suppliers that have out of bounds average success ratio or average call duration. StrategyParams: min_asr;max_asr;min_acd;max_acd;min_tcd;max_tcd;min_acc;max_acc;min_tcc;max_tcc

***qos (sorting)** The system will sort by metrics in the order of appearance. StrategyParams: metric1;metric2;etc

***load_distribution (sorting/filter)** The system will sort the suppliers in order to achieve the specified load distribution. - if all have less than ratio return random order - if some have a cdr count not divisible by ratio return them first and all ordered by cdr times, oldest first - if all have a multiple of ratio return in the order of cdr times, oldest first StrategyParams: supplier1:ratio;supplier2:ratio;*default:ratio

ActivationTime is the date/time when the LCR entry starts to be active.

Weight is used to sort the rules with the same activation time.

Example

```
*in, cgrates.org,call,*any,*any,EU_LANDLINE,LCR_STANDARD,*static,ivo;dan;rif,2012-01-
↪01T00:00:00Z,10
```

Code implementation

The general process of getting LCRs is this.

The LCR rules for a specific call descriptor are searched using direction, tenant, category, account and subject of the call descriptor matched as strictly as possible with LCR rules.

Because a rule can have several entries they will be sorted by activation time.

Next the system will find out the most recent LCR entry that applies to this call considering entries activation times.

The LCR entry is processed according to it's strategy. For static strategy the cost is calculated for each supplier found in the parameters and the suppliers are listed as they are found.

For the QOS strategies the suppliers are searched using call descriptor parameters (direction, tenant, category, account, subject), than the cdrstats module is queried for the QOS values and the suppliers are filtered or sorted according to the StrategyParameters field. The suppliers that have the QOS parameters in the stats queues but did not get the chance to process any calls are favored in the QOS sorting algorithm. If a certain QOS metric is missing from the supplier queues than the metric is ignored and the sorting or filtering is done using the next metrics that are considered.

For the lowest/highest cost strategies the matched suppliers are sorted ascending/descending on cost.

```
{
 "Entry": {
      "DestinationId": "*any",
      "RPCategory": "LCR_STANDARD",
      "Strategy": "*lowest_cost",
      "StrategyParams": "",
      "Weight": 20
  },
  "SupplierCosts": [{"Supplier":"rif", Cost:"2.0"},{"Supplier":"dan", Cost:"1.0"}]
 }
```

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.

Rating history

Enhances CGRateS with ability to archive rates modifications.

Large scaling posibility using server-agents approach. In a distributed environment, there will be a single server (which can be backed up using technologies such as Linux-HA) and more agents sending the modifications to be archived.

History-Server

Part of the *cgr-engine*.

Controlled within *history_server* section of the configuration file.

Stores rating archive in a .git folder, hence making the changes available for analysis via any git browser tool (eg: gitg in linux).

Functionality:

- On startup reads the rating archive out of .git folder and caches the data.
- When receiving rating information from the agents it will recompile the cache.
- Based on configured save interval it will dump the rating cache (if changed) into the .git archive.
- Archives the following rating data:
- Destinations inside *destinations.json* file.
- Rating plans inside *rating_plans.json* file.
- Rating profiles inside *rating_profiles.json* file.

History-Agent

Integrated in the rating loader components.

Part of *cgr-engine* and *cgr-loader*.

Enabled via *history_agent* configuration section within *cgr-engine* and *history_server* command line parameter in case of *cgr-loader*.

Sends the complete rating data loaded into dataDb to *history_server* for archiving.

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.

7. Tutorials

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 libjansson-dev
cd /tmp/
wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-14-current.tar.gz
tar xzvf asterisk-14-current.tar.gz
cd asterisk-14.0.2/
./configure --with-pjproject-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
```

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: 1001/1234, 1002/1234, 1003/1234 and 1004/1234.

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.
- Will charge by default *RT_40CNT* for all 10xx destinations during peak times (Monday-Friday 08:00-19:00) and *RT_10CNT* during offpeatimes (rest).
- Account 1001 will receive a special *deal* for 1002 and 1003 destinations during peak times with *RT_20CNT*, otherwise having default rating.
- Accounting part will have following configured:
- Create 5 accounts: 1001, 1002, 1003, 1004, 1007.
- Create 1 account alias (1006 - alias of account 1002).

- Create 1 rating profile alias (1006 - alias of rating profile 1001).
- 1002, 1003, 1004 will receive 10units of **monetary* balance.
- 1001 will receive 5 units of general **monetary*, 5 units of shared balance in the shared group "SHARED_A" and 90 seconds of calling destination 1002 with special rates *RT_1CNT*.
- 1007 will receive 0 units of shared balance in the shared group "SHARED_A".
- Define the shared balance "SHARED_A" with debit policy **highest*.
- For each balance created, attach 4 triggers to control the balance: log on balance<2, log on balance>20, log on 5 mins talked towards 10xx destination, disable the account and log if a balance is higher than 100 units.
- *DerivedCharging* will execute one extra mediation run when the sessions will have as account and rating subject 1001 resulting in a cloned session with most of parameters identical to original except RequestType which will be set on *rated* instead of original *prepaid* one. The extra run will be identified by *derived_run1* in CDRs.
- Will configure 4 extra CdrStatQueues:
  - *CDRST1* with 10 CDRs in the Queue and unlimited time window, calculating *ASR*, *ACD* and *ACC* for CDRs with Tenant matching *cgrates.org* and MediationRunId matching *default*. On this StatsQueue we will attach an ActionTrigger profile identified by *CDRST1_WARN*
  - *CDRST_1001* with 10 CDRs in the Queue and 10 minutes time window calculating *ASR*, *ACD* and *ACC* for CDRs with Tenant matching *cgrates.org*, RatingSubject matching *1001* and MediationRunId matching *default*. On this StatsQueue we will attach an ActionTrigger profile identified by *CDRST1001_WARN*
  - *CDRST_1002* with 10 CDRs in the Queue and 10 minutes time window calculating *ASR*, *ACD* and *ACC* for CDRs with Tenant matching *cgrates.org*, RatingSubject matching *1002* and MediationRunId matching *default*. On this StatsQueue we will attach an ActionTrigger profile identified by *CDRST1001_WARN*
  - *CDRST_1003* with 10 CDRs in the Queue and 10 minutes time window calculating *ASR*, and *ACD* for CDRs with Tenant matching *cgrates.org*, Destination matching *1003* and MediationRunId matching *default*. On this StatsQueue we will attach an ActionTrigger profile identified by *CDRST3_WARN*
  - The ActionTrigger *CDRST1_WARN* will monitor following StatsQueue Metric values:
  - ASR drop under 45 and a minimum of 3 CDRs in the StatsQueue will call Action profile *LOG_WARNING* which will log the StatsQueue to syslog. The Action will be recurrent with a sleep time of 1 minute.
  - ACD drop under 10 and a minimum of 5 CDRs in the StatsQueue will cause the same log to syslog. The Action will be recurrent with a sleep time of 1 minute.
  - ACC increase over 10 and a minimum of 5 CDRs in the StatsQueue will cause the StatsQueue to be again logged to syslog. The Action will be recurrent with a sleep time of 1 minute.
  - The ActionTrigger *CDRST1001_WARN* will monitor following StatsQueue Metric values:
  - ASR drop under 65 and a minimum of 3 CDRs in the StatsQueue will call Action profile *LOG_WARNING* which will log the StatsQueue to syslog. The Action will be recurrent with a sleep time of 1 minute.
  - ACD drop under 10 and a minimum of 5 CDRs in the StatsQueue will cause the same log to syslog. The Action will be recurrent with a sleep time of 1 minute.
  - ACC increase over 5 and a minimum of 5 CDRs in the StatsQueue will cause the StatsQueue to be again logged to syslog. The Action will be recurrent with a sleep time of 1 minute.

- The ActionTrigger *CDRST3_WARN* will monitor ACD Metric and react at a minimum ACD of 60 with 5 CDRs in the StatsQueue by writing again to syslog. This ActionTrigger will be fired one time then cleared by the scheduler.

```
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"]'
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1003"]'
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1004"]'
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1007"]'
```

- 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" TimeStart="2014-08-04T13:00:00Z" TimeEnd="2014-08-
↪04T13:00:20Z"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1002" TimeStart="2014-08-04T13:00:00Z" TimeEnd="2014-08-
↪04T13:01:25Z"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1003" TimeStart="2014-08-04T13:00:00Z" TimeEnd="2014-08-
↪04T13:00:20Z"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1003" TimeStart="2014-08-04T13:00:00Z" TimeEnd="2014-08-
↪04T13:01:25Z"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1004" TimeStart="2014-08-04T13:00:00Z" TimeEnd="2014-08-
↪04T13:00:20Z"'
cgr-console 'cost Category="call" Tenant="cgrates.org" Subject="1001"␣
↪Destination="1004" TimeStart="2014-08-04T13:00:00Z" TimeEnd="2014-08-
↪04T13:01:25Z"'
```

- Make sure *CDRStats Queues* were created:

```
cgr-console cdrstats_queueids
cgr-console 'cdrstats_metrics StatsQueueId="*default"'
```

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"]'
```

1003 -> 1001

The user 1003 is marked as *pseudoprepaid* inside the telecom switch hence his calls will be considered same as prepaid (no call setups possible on negative balance due to pre-auth mechanism) but not handled automatically by session manager. His call costs will be calculated directly out of CDRs and balance updated by the time when mediation process occurs. This is sometimes a good compromise of prepaid running without influencing performance (there are no recurrent call debits during a call).

To check that there are no debits during or by the end of the call, but when the CDR reaches the CDRS component(which is close to real-time in case of *http-json* CDRs):

```
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1003"]'
```

1004 -> 1001

The user 1004 is marked as *rated* inside the telecom switch hence his calls not interact in any way with accounting subsystem. The only action perfomed by **CGRateS** related to his calls wil be rating/mediation of his CDRs.

1006 -> 1002

Since the user 1006 is marked as *prepaid* inside the telecom switch, calling between 1006 and 1002 should generate pre-auth and prepaid debits which can be checked with *get_account* command integrated within *cgr-console* tool. One thing to note here is that 1006 is not defined as an account inside CGR Accounting Subsystem but as an alias of another account, hence *get_account* ran on 1006 will return "not found" and the debits can be monitored on the real account which is 1001.

Check that 1001 balance is properly debitted, 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=["1006"]'
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1001"]'
```

1007 -> 1002

Since the user 1007 is marked as *prepaid* inside the telecom switch, calling between 1007 and 1002 should generate pre-auth and prepaid debits which can be checked with *get_account* command integrated within *cgr-console* tool. Since 1007 has no units left into his accounts but he has one balance marked as shared, debits for this call should take

place in accounts which are a part of the same shared balance as the one of *1007/SHARED_A*, which in our scenario corresponds to the one of the account 1001.

Check that call can proceed even if 1007 has no units left into his own balances, and that the costs attached to the call towards 1002 are debited from the balance marked as shared within account 1001.

```
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1007"]'
cgr-console 'accounts Tenant="cgrates.org" AccountIds=["1001"]'
```

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" ExportDir="/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).

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" ExportDir="/tmp"'
```

CGRateS Usage

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

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:

```
apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xfb40d3e6508ea4c8
echo "deb http://deb.kamailio.org/kamailio44 jessie 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

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 2.2-releases" >>/etc/apt/sources.list
apt-get update
apt-get install opensips opensips-json-module opensips-restclient-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_async/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_async/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.

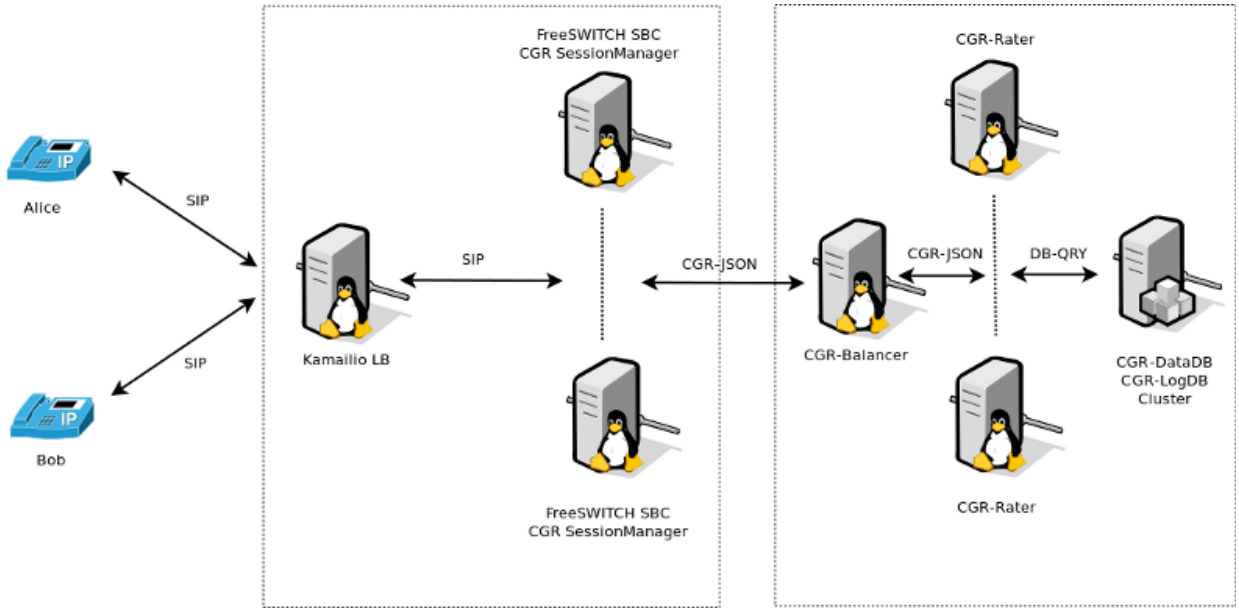
CGRateS Usage

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

8. Miscellaneous

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 INSUFFICIENT_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.

Bibliography

[WIKI2015] http://en.wikipedia.org/wiki/Least-cost_routing