
CGRateS Documentation

Release 0.11.0~dev

Dan Christian Bogos

Feb 21, 2020

| | | |
|----------|------------------------------|----------|
| 1 | Overview | 3 |
| 1.1 | Features | 4 |
| 1.2 | Links | 7 |
| 1.3 | License | 7 |
| 2 | Architecture | 9 |
| 2.1 | cgr-engine | 9 |
| 2.1.1 | Agents | 10 |
| 2.1.1.1 | DiameterAgent | 11 |
| 2.1.1.2 | RadiusAgent | 19 |
| 2.1.1.3 | HTTPAgent | 19 |
| 2.1.1.4 | DNSAgent | 19 |
| 2.1.1.5 | AsteriskAgent | 19 |
| 2.1.1.6 | FreeSWITCHAgent | 19 |
| 2.1.1.7 | KamailioAgent | 19 |
| 2.1.1.8 | EventReaderService | 20 |
| 2.1.2 | SessionS | 24 |
| 2.1.3 | RALs | 24 |
| 2.1.4 | CDRs | 24 |
| 2.1.5 | AttributeS | 24 |
| 2.1.5.1 | Selection | 25 |
| 2.1.5.2 | Parameters | 25 |
| 2.1.5.3 | Use cases | 26 |
| 2.1.6 | ChargerS | 26 |
| 2.1.6.1 | DerivedCharging | 27 |
| 2.1.6.2 | Processing logic | 27 |
| 2.1.6.3 | Parameters | 27 |
| 2.1.6.4 | Use cases | 27 |
| 2.1.7 | ResourceS | 28 |
| 2.1.7.1 | Processing logic | 28 |
| 2.1.7.2 | Parameters | 28 |
| 2.1.7.3 | Use cases | 29 |
| 2.1.8 | SupplierS | 30 |
| 2.1.8.1 | Processing logic | 30 |
| 2.1.8.2 | APIs logic | 30 |
| 2.1.8.3 | Parameters | 30 |

| | | |
|----------|--|-----------|
| 2.1.8.4 | Use cases | 32 |
| 2.1.9 | StatS | 32 |
| 2.1.9.1 | Processing logic | 33 |
| 2.1.9.2 | Parameters | 33 |
| 2.1.9.3 | Use cases | 34 |
| 2.1.10 | ThresholdS | 34 |
| 2.1.10.1 | Processing logic | 35 |
| 2.1.10.2 | APIs logic | 35 |
| 2.1.10.3 | Parameters | 35 |
| 2.1.10.4 | Use cases | 36 |
| 2.1.11 | FilterS | 36 |
| 2.1.11.1 | Filter profile | 37 |
| 2.1.11.2 | Filter rule | 37 |
| 2.1.11.3 | Inline Filter | 38 |
| 2.1.11.4 | Subsystem profiles selection based on Filters | 38 |
| 2.1.12 | DispatcherS | 38 |
| 2.1.13 | SchedulerS | 38 |
| 2.1.14 | CDRe | 39 |
| 2.1.15 | APIerS | 39 |
| 2.1.16 | LoaderS | 39 |
| 2.1.17 | DataDB | 39 |
| 2.1.18 | StorDB | 39 |
| 2.1.19 | CacheS | 39 |
| 2.2 | cgr-console | 39 |
| 2.3 | cgr-loader | 40 |
| 2.4 | cgr-migrator | 41 |
| 2.5 | cgr-tester | 43 |
| 3 | 3. Installation | 45 |
| 3.1 | 3.1. Using packages | 45 |
| 3.1.1 | 3.1.1. Debian | 45 |
| 3.1.1.1 | 3.1.1.1. Aptitude repository | 45 |
| 3.1.1.2 | 3.1.1.2. Manual installation of .deb package out of archive server | 46 |
| 3.1.2 | 3.1.2. Redhat/Fedora/CentOS | 46 |
| 3.1.2.1 | 3.1.2.1. YUM repository | 46 |
| 3.1.2.2 | 3.1.2.2. Manual installation of .rpm package out of archive server | 46 |
| 3.2 | 3.2. Using source | 46 |
| 3.2.1 | 3.2.1 Install GO Lang | 47 |
| 3.2.2 | 3.2.2 Build CGRateS from Source | 47 |
| 3.2.3 | 3.2.3 Create Debian / Ubuntu Packages from Source | 47 |
| 3.2.4 | 3.2.4 Install Custom Debian / Ubuntu Package | 47 |
| 3.3 | 3.3. Post-install | 47 |
| 3.3.1 | 3.3.1. Database setup | 47 |
| 3.3.2 | 3.3.2 Set versions data | 48 |
| 4 | 4. Configuration | 49 |
| 5 | 5. Administration | 75 |
| 6 | 6. Advanced Topics | 77 |
| 6.1 | API Calls | 77 |
| 6.2 | CDR Server | 77 |
| 6.2.1 | CDR-CGR | 77 |
| 6.2.2 | CDR-FS_JSON | 78 |
| 6.2.3 | CDR-RPC | 79 |

| | | |
|----------|---|-----------|
| 6.3 | CDR Client (cdrc) | 80 |
| 6.3.1 | Import Templates | 81 |
| 6.3.1.1 | CGR-RSR Regexp Rule | 81 |
| 6.3.1.2 | CGR-RSR Static Rule | 81 |
| 6.3.2 | CDR .CSV | 81 |
| 6.4 | CDR Exporter | 82 |
| 6.4.1 | Export Templates | 82 |
| 6.4.1.1 | CGR-RSR Regexp Rule | 82 |
| 6.4.1.2 | CGR-RSR Static Rule | 82 |
| 6.4.2 | CGR-CSV | 83 |
| 6.4.3 | CGR-FWV | 84 |
| 6.4.4 | Hybrid CSV-FWV | 84 |
| 6.5 | CDR Stats Server | 84 |
| 6.5.1 | Configuration | 85 |
| 6.5.2 | Metrics Types | 85 |
| 6.5.3 | ExternalQueries | 85 |
| 6.5.4 | Example use | 86 |
| 6.6 | DerivedCharging | 87 |
| 6.6.1 | Configuration | 87 |
| 6.7 | Rating logic | 88 |
| 6.7.1 | User balances | 90 |
| 7 | 7. Tutorials | 91 |
| 7.1 | Asterisk Integration Tutorials | 91 |
| 7.1.1 | Software installation | 91 |
| 7.1.1.1 | CGRateS | 91 |
| 7.1.1.2 | Asterisk | 91 |
| 7.1.2 | SIP UA - Jitsi | 92 |
| 7.1.3 | Asterisk interaction via <i>ARI</i> | 92 |
| 7.1.3.1 | Scenario | 92 |
| 7.1.3.2 | Starting Asterisk with custom configuration | 92 |
| 7.1.3.3 | Starting CGRateS with custom configuration | 92 |
| 7.1.3.4 | CDR processing | 93 |
| 7.1.3.5 | CGRateS Usage | 93 |
| 7.1.4 | CGRateS Usage | 93 |
| 7.1.4.1 | Loading CGRateS Tariff Plans | 93 |
| 7.1.4.2 | Test calls | 94 |
| 7.1.4.3 | CDR Exporting | 94 |
| 7.1.4.4 | Fraud detection | 94 |
| 7.2 | FreeSWITCH Integration Tutorials | 95 |
| 7.2.1 | Software installation | 95 |
| 7.2.1.1 | CGRateS | 95 |
| 7.2.1.2 | FreeSWITCH | 95 |
| 7.2.2 | FreeSWITCH generating <i>http-json</i> CDRs | 95 |
| 7.2.2.1 | Scenario | 95 |
| 7.2.2.2 | Starting FreeSWITCH with custom configuration | 96 |
| 7.2.2.3 | Starting CGRateS with custom configuration | 96 |
| 7.2.2.4 | CDR processing | 96 |
| 7.2.2.5 | CGRateS Usage | 96 |
| 7.3 | Kamailio Integration Tutorials | 96 |
| 7.3.1 | Software installation | 97 |
| 7.3.1.1 | CGRateS | 97 |
| 7.3.1.2 | Kamailio | 97 |
| 7.3.2 | Kamailio interaction via <i>evapi</i> module | 97 |

| | | |
|----------|---|------------|
| 7.3.2.1 | Scenario | 97 |
| 7.3.2.2 | Starting Kamailio with custom configuration | 97 |
| 7.3.2.3 | Starting CGRateS with custom configuration | 98 |
| 7.3.2.4 | CDR processing | 98 |
| 7.3.2.5 | CGRateS Usage | 98 |
| 7.4 | OpenSIPS Integration Tutorials | 98 |
| 7.4.1 | Software installation | 98 |
| 7.4.1.1 | CGRateS | 98 |
| 7.4.1.2 | OpenSIPS | 98 |
| 7.4.2 | OpenSIPS interaction via <i>event_datagram</i> | 99 |
| 7.4.2.1 | Scenario | 99 |
| 7.4.2.2 | Starting OpenSIPS with custom configuration | 99 |
| 7.4.2.3 | Starting CGRateS with custom configuration | 99 |
| 7.4.2.4 | CDR processing | 99 |
| 7.4.2.5 | CGRateS Usage | 99 |
| 8 | 8. Miscellaneous | 101 |
| 8.1 | 8.1. FreeSWITCH integration | 101 |
| 8.1.1 | 8.1.1. SessionManager | 101 |
| 8.1.2 | 8.1.2. Mediator | 103 |
| 8.1.2.1 | 8.1.2.1. Modes of operation | 103 |
| 8.1.2.2 | 8.1.2.2. Implementation logic | 103 |

Welcome to CGRateS's documentation!

CGRateS is a *very fast* (**50k+ CPS**) and *easily scalable* (**load-balancer + replication** included) **Real-time Enterprise Billing Suite** targeted especially for ISPs and Telecom Operators (but not only).

Starting as a pure **billing engine**, CGRateS has evolved over the years into a reliable **real-time charging framework**, able to accommodate various business cases in a *generic way*.

Being an “*engine style*” the project focuses on providing best ratio between **functionality** (over 15 daemons/services implemented with a rich number of *features* and a development team agile in implementing new ones) and **performance** (dedicated benchmark tool, asynchronous request processing, own transactional cache component), however not losing focus of **quality** (test driven development policy).

It is written in **Go** programming language and accessible from any programming language via **JSON RPC**. The code is well documented (**go doc** compliant **API docs**) and heavily tested (**5k+** tests are part of the unit test suite).

Meant to be pluggable into existing billing infrastructure and as non-intrusive as possible, CGRateS passes the decisions about logic flow to system administrators and incorporates as less as possible business logic.

Modular and flexible, CGRateS provides APIs over a variety of simultaneously accessible communication interfaces:

- **In-process** : optimal when there is no need to split services over different processes
- **JSON over TCP** : most preferred due to its simplicity and readability
- **JSON over HTTP** : popular due to fast interoperability development
- **JSON over Websockets** : useful where 2 ways interaction over same TCP socket is required
- **GOB over TCP** : slightly faster than JSON one but only accessible for the moment out of Go (<https://golang.org/>).

CGRateS is capable of four charging modes:

- ***prepaid**
 - Session events monitored in real-time
 - Session authorization via events with security call timer
 - Real-time balance updates with configurable debit interval
 - Support for simultaneous sessions out of the same account

- Real-time fraud detection with automatic mitigation
- *Advantage*: real-time overview of the costs and fast detection in case of fraud, concurrent account sessions supported
- *Disadvantage*: more CPU intensive.
- ***pseudoprepaid**
 - Session authorization via events
 - Charging done at the end of the session out of CDR received
 - *Advantage*: less CPU intensive due to less events processed
 - *Disadvantage*: as balance updates happen only at the end of the session there can be costs discrepancy in case of multiple sessions out of same account (including going on negative balance).
- ***postpaid**
 - Charging done at the end of the session out of CDR received without session authorization
 - Useful when no authorization is necessary (trusted accounts) and no real-time event interaction is present (balance is updated only when CDR is present).
- ***rated**
 - Special charging mode where there is no accounting interaction (no balances are used) but the primary interest is attaching costs to CDRs.
 - Specific mode for Wholesale business processing high-throughput CDRs
 - Least CPU usage out of the four modes (fastest charging).

1.1 Features

- **Performance oriented.** To get an idea about speed, we have benchmarked 50000+ req/sec on commodity hardware without a
 - Using most modern programming concepts like multiprocessor support, asynchronous code execution within microthreads, channel based locking
 - Built-in data caching system with LRU and TTL support
 - Linear performance increase via simple hardware addition
 - On demand performance increase via in-process / over network communication between engine services.
- **Modular architecture**
 - Pluggable into existing infrastructure
 - Non-intrusive into existing setups
 - Easy to enhance functionality by writing custom components
 - Flexible API accessible via both **GOB** (Go specific, increased performance) or **JSON** (platform independent, universally accessible)
 - Easy distribution (one binary concept, can run via NFS on all Linux servers without install).
- **Easy administration**
 - One binary can run on all Linux servers without additional installation (simple copy)

- Can run diskless via NFS
- Virtualization/containerization friendly(runs on [Docker](#)).
- **GOCS (Global Online Charging System)**
 - Support for global networks with one master + multi-cache nodes around the globe for low query latency
 - Mutiple Balance types per Account (*monetary, *voice, *sms, *data, *generic)
 - Unlimited number of Account Balances with weight based prioritization
 - Various Balance filters (ie: per-destination, roaming-only, weekend-only)
 - Support for Volume based discounts and automatic bonuses (ie: 5 SMS free for every 10 minutes in one hour to specific destination)
 - Session based charging with support for concurrent sessions per account and per session dynamic debit interval
 - Session emulation combined with Derived Charging (separate charging for distributors chaining, customer/supplier parallel calculations)
 - Balance reservation and refunds
 - Event based charging (ie: SMS, MESSAGE)
 - Built-in Task-Scheduler supporting both one-time as well as recurrent actions (automatic subscriptions management, recurrent *debit/*topup, DID charging)
 - Real-time balance monitors with automatic actions triggered (bonuses or fraud detection).
- **Highly configurable Rating**
 - Connect Fees
 - Priced Units definition
 - Rate increments
 - Rate groups (ie. charge first minute in a call as a whole and next ones per second)
 - Verbose durations(up to nanoseconds billing)
 - Configurable decimals per destination
 - Rating subject categorization (ie. premium/local charges, roaming)
 - Recurrent rates definition (per year, month, day, dayOfWeek, time)
 - Rating Profiles activation times (eg: rates becoming active at specific time in the future)
 - Rating Profiles fallback (per subject destinations with fallback to server wide pricing)
 - Verbose charging logs to comply strict rules imposed by some country laws.
- **Multi-Tenant from day one**
 - Default Tenant configurable for one-tenant systems
 - Security enforced for RPC-API on Tenant level.
- **Online configuration reloads without restart**
 - Engine configuration from .json folder or remote http server
 - Tariff Plans from .csv folder or database storage.
- **CDR server**

- Optional offline database storage
- Online (rating queues) or offline (via RPC-API) exports with customizable content via .json templates
- Multiple export interfaces: files, HTTP, AMQP, SQS, Kafka.
- **Generic Event Reader**
 - Process various sources of events and convert them into internal ones which are sent to CDR server for rating
 - Conversion rules defined in .json templates
 - Supported interfaces: .csv, .xml, fixed width files, Kafka.
- **Events mediation**
 - Ability to add/change/remove information within *Events* to achieve additional services or correction
 - Performance oriented.
- **Routing server for VoIP**
 - Implements strategies like *Least Cost Routing*, *Load Balacer*, *High Availability*
 - Implements *Number Portability* service.
- **Resource allocation controller**
 - Generic filters for advanced logic
 - In-memory operations for increased performance
 - Backup in offline storage.
- **Stats service**
 - Generic stats (*sum, *difference, *multiply, *divide)
 - In-memory operations for increased performance
 - Backup in offline storage.
- **Thresholds monitor**
 - Particular implementation of *Fraud Detection with automatic mitigation*
 - Execute independent actions which can serve various purposes (notifications, accounts disables, bonuses to accounts).
- **Multiple RPC interfaces**
 - Support for *JSON-RPC*, *GOB-PC* over TCP, HTTP, websockets
 - Support for HTTP-REST interface.
- **Various agents to outside world:**
 - Asterisk
 - FreeSWITCH
 - Kamailio
 - OpenSIPS
 - Diameter
 - Radius
 - Generic HTTP

- DNS/ENUM.
- **Built in High-Availability mechanisms:**
 - Dispatcher with static or dynamic routing
 - Server data replication
 - Client remote data querying.
- Good documentation (that’s me :).
- **“Free as in Beer”** with commercial support available on-demand.

1.2 Links

- 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 [#cgrates](irc.freenode.net)

1.3 License

CGRateS is released under the terms of the [GNU GENERAL PUBLIC LICENSE Version 3].

The CGRateS framework consists of functionality packed within **five** software applications, described below.

2.1 cgr-engine

Groups most of functionality from services and components.

Customisable through the use of *json JSON configuration* or command line arguments (higher prio).

Able to read the configuration from either a local directory of *.json* files with an unlimited number of subfolders (ordered alphabetically) or a list of http paths (separated by “;”).

```
$ cgr-engine -help
Usage of cgr-engine:
  -config_path string
    Configuration directory path. (default "/etc/cgrates/")
  -cpuprof_dir string
    write cpu profile to files
  -httprof_path string
    http address used for program profiling
  -log_level int
    Log level (0-emergency to 7-debug) (default -1)
  -logger string
    logger <*syslog|*stdout>
  -memprof_dir string
    write memory profile to file
  -memprof_interval duration
    Time between memory profile saves (default 5s)
  -memprof_nrfiles int
    Number of memory profile to write (default 1)
  -node_id string
    The node ID of the engine
  -pid string
    Write pid file
```

(continues on next page)

(continued from previous page)

```
-scheduled_shutdown string
    shutdown the engine after this duration
-singlecpu
    Run on single CPU core
-version
    Prints the application version.
```

Hint: \$ cgr-engine -config_path=/etc/cgrates

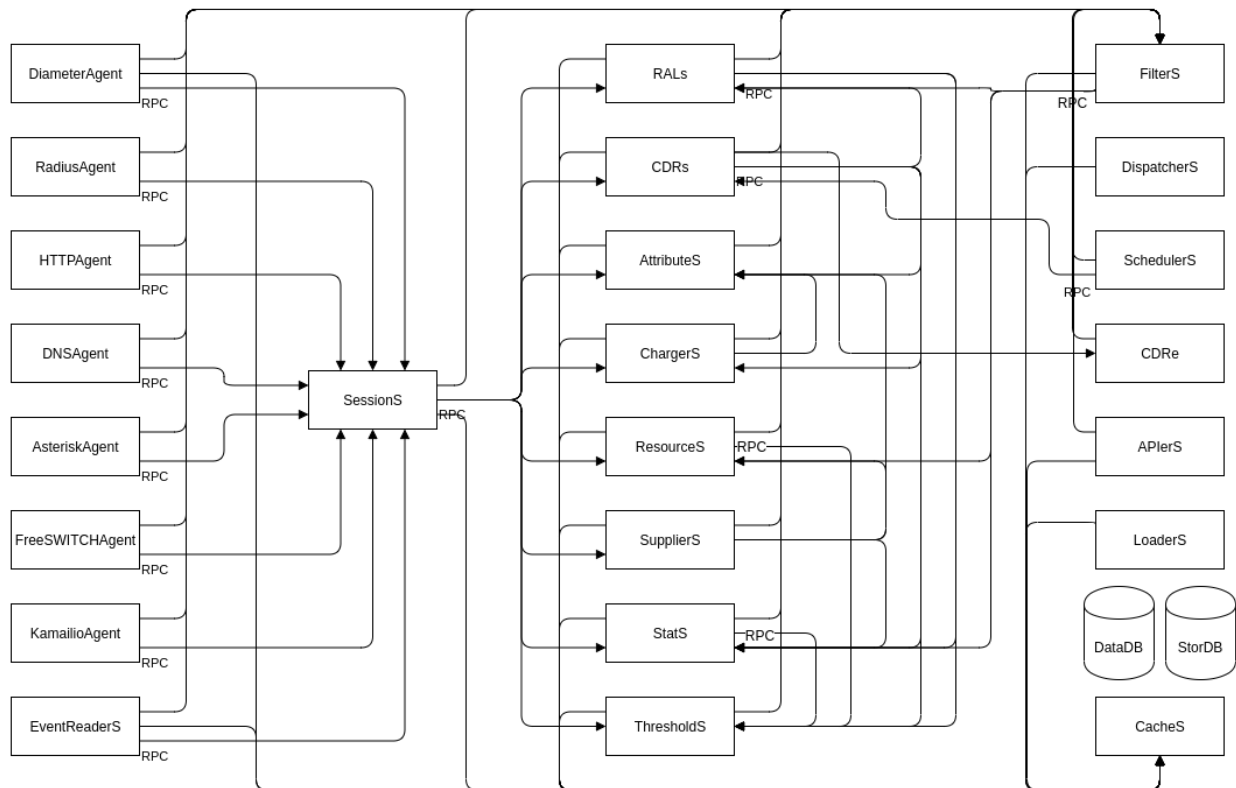


Fig. 1: Internal Architecture of **cgr-engine**

The components from the diagram can be found documented in the links below:

2.1.1 Agents

Agents are interfaces towards external systems, implementing protocols enforced by the communication channels opened. These can be standard or privately defined.

All of the **Agents** implemented within CGRateS are flexible to be configured with generic parameters configurable for both *request* and *replies*.

Following *Agents* are implemented within CGRateS:

2.1.1.1 DiameterAgent

DiameterAgent translates between **Diameter** and **CGRateS**, sending *RPC* requests towards **CGRateS/SessionS** component and returning replies from it to the *DiameterClient*.

Implements **Diameter** protocol in a standard agnostic manner, giving users the ability to implement own interfaces by defining simple *processor templates* within the *JSON configuration* files.

Used mostly in modern mobile networks (LTE/xG).

Configuration

The **DiameterAgent** is configured within *diameter_agent* section from *JSON configuration*.

Sample config

With explanations in the comments:

```
"diameter_agent": {
    "enabled": false,                                // enables the
↪diameter agent: <true|false>
    "listen": "127.0.0.1:3868",                      // address where to listen for
↪diameter requests <x.y.z.y/x1.y1.z1.y1:1234>
    "listen_net": "tcp",                             // transport type for diameter
↪<tcp|sctp>
    "dictionaries_path": "/usr/share/cgrates/diameter/dict/", // path
↪towards directory
                                                                //
↪holding additional dictionaries to load
    "sessions_conns": ["*internal"],                 // connection towards SessionS
    "origin_host": "CGR-DA",                         // diameter Origin-Host AVP
↪used in replies
    "origin_realm": "cgrates.org",                  // diameter Origin-Realm AVP used in
↪replies
    "vendor_id": 0,                                 // diameter Vendor-Id
↪AVP used in replies
    "product_name": "CGRateS",                      // diameter Product-Name AVP
↪used in replies
    "concurrent_requests": -1,                       // limit the number of active
↪requests processed by the server <-1|0-n>
    "synced_conn_requests": false,                  // process one request at the time per
↪connection
    "asr_template": "*asr",                          // enable AbortSession message
↪being sent to client
                                                                //
↪forcing session disconnection from CGRateS side
    "templates":{                                   // message templates
↪which can be injected within request/replies
        "*err": [                                    // *err is
↪used mostly in automatic diameter replies with errors
            {
                "tag": "SessionId", "path": "*rep.Session-Id",
                "type": "*variable", "mandatory": true,
                "value": "~*req.Session-Id"
            },
        ],
    },
}
```

(continues on next page)

(continued from previous page)

```

        {
            "tag": "OriginHost", "path": "*rep.Origin-Host
↔",
            "type": "*variable", "mandatory": true,
            "value": "~*vars.OriginHost"
        },
        {
            "tag": "OriginRealm", "path": "*rep.Origin-
↔Realm",
            "type": "*variable", "mandatory": true,
            "value": "~*vars.OriginRealm"
        },
    ],
    "*cca": [ // *cca is used into CallControlAnswer messages
        {
            "tag": "SessionId", "path": "*rep.Session-Id",
            "type": "*composed", "mandatory": true,
            "value": "~*req.Session-Id"
        },
        {
            "tag": "ResultCode", "path": "*rep.Result-Code
↔",
            "type": "*constant", "value": "2001"},
        {
            "tag": "OriginHost", "path": "*rep.Origin-Host
↔",
            "type": "*variable", "mandatory": true,
            "value": "~*vars.OriginHost"
        },
        {
            "tag": "OriginRealm", "path": "*rep.Origin-
↔Realm",
            "type": "*variable", "mandatory": true,
            "value": "~*vars.OriginRealm"
        },
        {
            "tag": "AuthApplicationId",
            "path": "rep.Auth-Application-Id",
            "type": "*variable", "mandatory": true,
            "value": "~*vars.*appid"
        },
        {
            "tag": "CCRequestType",
            "path": "*rep.CC-Request-Type",
            "type": "*variable", "mandatory": true,
            "value": "~*req.CC-Request-Type"
        },
        {
            "tag": "CCRequestNumber",
            "path": "*rep.CC-Request-Number",
            "type": "*variable", "mandatory": true,
            "value": "~*req.CC-Request-Number"
        },
    ],
    "*asr": [ // *asr is used to build AbortSessionRequest
        {
            "tag": "SessionId", "path": "*diamreq.Session-
↔Id",

```

(continues on next page)

(continued from previous page)

```

        "type": "*variable", "mandatory": true,
        "value": "~*req.Session-Id"
    },
    {
        "tag": "OriginHost", "path": "diamreq.Origin-
↪Host",
        "type": "*variable", "mandatory": true,
        "value": "~*req.Destination-Host"
    },
    {
        "tag": "OriginRealm", "path": "diamreq.Origin-
↪Realm",
        "type": "*variable", "mandatory": true,
        "value": "~*req.Destination-Realm"
    },
    {
        "tag": "DestinationRealm",
        "path": "*diamreq.Destination-Realm",
        "type": "*variable", "mandatory": true,
        "value": "~*req.Origin-Realm"
    },
    {
        "tag": "DestinationHost",
        "path": "*diamreq.Destination-Host",
        "type": "*variable", "mandatory": true,
        "value": "~*req.Origin-Host"
    },
    {
        "tag": "AuthApplicationId",
        "path": "*diamreq.Auth-Application-Id",
        "type": "*variable", "mandatory": true,
        "value": "~*vars.*appid"
    },
    {
        "tag": "UserName", "path": "*diamreq.User-Name
↪",
        "type": "*variable", "mandatory": true,
        "value": "~*req.User-Name"
    },
    {
        "tag": "OriginStateID",
        "path": "*diamreq.Origin-State-Id",
        "type": "*constant", "value": "1"
    }
    ]
},
"request_processors": [ // decision logic for message processing
    {
        "id": "SMSes", // id is used for debug in logs (ie: ↪
↪using *log flag)
        "filters": [ // list of filters to be applied on ↪
↪message for this processor to run
            "*string::~*cmd:CCR",
            "*string::~*req.CC-Request-Type:4",
            "*string::~*req.Service-Context-Id:LPP"
        ],
        "flags": ["*event", "*accounts", "*cdrs"], // influence ↪
↪processing logic within CGRateS workflow (continues on next page)

```

(continued from previous page)

```

        "request_fields":[
↪      // data exchanged between Diameter and CGRateS
        {
            "tag": "ToR",                // tag is used
↪in debug,
            "path": "*cgreg.ToR",       // path is the field
↪on CGRateS side
            "type": "*constant",       // type defines the
↪method to provide the value
            "value": "*sms"}
        {
            "tag": "OriginID",          //
↪OriginID will identify uniquely
            "path": "*cgreg.OriginID",  // the session
↪on CGRateS side
            "type": "*variable",       // it's value
↪will be taken from Diameter AVP:
            "mandatory": true,         //
↪Multiple-Services-Credit-Control.Service-Identifier
            "value": "~*req.Multiple-Services-Credit-
↪Control.Service-Identifier"
        },
        {
            "tag": "OriginHost",        // OriginHost
↪combined with OriginID
            "path": "*cgreg.OriginHost", // is used by
↪CGRateS to build the CGRID
            "mandatory": true,
            "type": "*variable",       // have the
↪value out of special variable: *vars
            "value": "*vars.OriginHost"
        },
        {
            "tag": "RequestType",       //
↪RequestType instructs SessionS
            "path": "*cgreg.RequestType", // about
↪charging type to apply for the event
            "type": "*constant",
            "value": "*prepaid"
        },
        {
            "tag": "Category",          //
↪Category serves for attaching Account
            "path": "*cgreg.Category",  // and
↪RatingProfile to the request
            "type": "*constant",
            "value": "sms"
        },
        {
            "tag": "Account",           //
↪Account is required by charging
            "path": "*cgreg.Account",
            "type": "*variable",       // value is
↪taken dynamically from a group AVP
            "mandatory": true,         //
↪where Subscription-Id-Type is 0
            "value": "~*req.Subscription-Id.Subscription-
↪Id-Data[~Subscription-Id-Type(0)]"

```

(continues on next page)

(continued from previous page)

```

    },
    {
        "tag": "Destination", //
↪ Destination is used for charging
        "path": "*cgregq.Destination", // value from
↪ Diameter will be mediated before sent to CGRateS
        "type": "*variable",
        "mandatory": true,
        "value": "~*req.Service-Information.SMS-
↪ Information.Recipient-Info.Recipient-Address.Address-Data:s/^\+49(\d+)/int${1}:/s/
↪ ^0049(\d+)/int${1}:/s/^49(\d+)/int${1}:/s/^00(\d+)/+${1}:/s/^[\\+]?(\d+)/int${1}
↪ /:s/int(\d+)/+49${1}/"
    },
    {
        "tag": "Destination", // Second
↪ Destination will overwrite the first if filter matches
        "path": "*cgregq.Destination",
        "filters": [
↪ // Only overwrite when filters are matching
            "*notprefix::~*req.Service-Information.
↪ SMS-Information.Recipient-Info.Recipient-Address.Address-Data:49",
            "*notprefix::~*req.Service-Information.
↪ SMS-Information.Recipient-Info.Recipient-Address.Address-Data:3312"
        ],
        "type": "*variable",
        "mandatory": true,
        "value": "~*req.Service-Information.SMS-
↪ Information.Recipient-Info.Recipient-Address.Address-Data:s/^[\\+]?(\d+)/int${1}
↪ :s/int(\d+)/+00${1}/"
    },
    {
        "tag": "SetupTime", //
↪ SetupTime is used by charging
        "path": "*cgregq.SetupTime",
        "type": "*variable",
        "value": "~*req.Event-Timestamp",
        "mandatory": true
    },
    {
        "tag": "AnswerTime", // AnswerTime
↪ is used by charging
        "path": "*cgregq.AnswerTime",
        "type": "*variable",
        "mandatory": true,
        "value": "~*req.Event-Timestamp"
    },
    {
        "tag": "Usage", // Usage is
↪ used by charging
        "path": "*cgregq.Usage",
        "type": "*variable",
        "mandatory": true,
        "value": "~*req.Multiple-Services-Credit-
↪ Control.Requested-Service-Unit.CC-Service-Specific-Units"
    },
    {
        "tag": "Originator-SCCP-Address",
↪ // Originator-SCCP-Address is an extra field which we want in (continues on next page)

```

(continued from previous page)

```

    "path": "*cgreg.Originator-SCCP-Address",
    "type": "*variable", "mandatory": true,
    "value": "~*req.Service-Information.SMS-
    Information.Originator-SCCP-Address"
  },
  "reply_fields": [ // fields which are
    sent back to DiameterClient
    {
      "tag": "CCATemplate", // inject complete
      Template defined as *cca above
      "type": "*template",
      "value": "*cca"
    },
    {
      "tag": "ResultCode", // Change the
      ResultCode if the reply received from CGRateS contains a 0 MaxUsage
      "filters": ["*eq::~*cgreg.MaxUsage:0"],
      "path": "*rep.Result-Code",
      "blocker": true, // do not
      consider further fields if this one is processed
      "type": "*constant",
      "value": "4012"},
      {"tag": "ResultCode", // Change the
      ResultCode AVP if there was an error received from CGRateS
      "filters": ["*notempty::~*cgreg.Error:"],
      "path": "*rep.Result-Code",
      "blocker": true,
      "type": "*constant",
      "value": "5030"}
    ]
  },
],
},
},

```

Config params

Most of the parameters are explained in *JSON configuration*, hence we mention here only the ones where additional info is necessary or there will be particular implementation for *DiameterAgent*.

listen_net The network the *DiameterAgent* will bind to. CGRateS supports both **tcp** and **sctp** specified in *Diameter* standard.

concurrent_requests The maximum number of active requests processed at one time by the *DiameterAgent*. When this number is reached, new inbound requests will be rejected with *DiameterError* code until the concurrent number drops below again. The default value of *-1* imposes no limits.

asr_template The template (out of templates config section) used to build the AbortSession message. If not specified the ASR message is never sent out.

templates Group fields based on their usability. Can be used in both processor templates as well as hardcoded within

CGRateS functionality (ie **err* or **asr*). The IDs are unique, defining the same id in multiple configuration places/files will result into overwrite.

err Is a hardcoded template used when *DiameterAgent* cannot parse the incoming message. Aside from logging the error via internal logger the message defined via **err* template will be sent out.

***asr** Can be activated via *asr_template* config key to enable sending of *Diameter ASR* message to *Diameter-Client*.

***cca** Defined for convenience to follow the standard for the fields used in *Diameter CCA* messages.

request_processors List of processor profiles applied on request/replies.

Once a request processor will be matched (it's *filters* should match), the *request_fields* will be used to craft a request object and the flags will decide what sort of processing logic will be applied to the crafted request.

After request processing, there will be a second part executed: reply. The reply object will be built based on the *reply_fields* section in the request processor.

Once the *reply_fields* are finished, the object converted and returned to the *DiameterClient*, unless *continue* flag is enabled in the processor, which makes the next request processor to be considered.

filters Will specify a list of filter rules which need to match in order for the processor to run (or field to be applied).

For the dynamic content (prefixed with ~) following special variables are available:

***vars** Request related shared variables between processors, populated especially by core functions. The data put in there is not automatically transferred into requests sent to CGRateS, unless instructed inside templates.

Following vars are automatically set by core:

- **OriginHost:** agent configured *origin_host*
- **OriginRealm:** agent configured *origin_realm*
- **ProductName:** agent configured *product_name*
- **RemoteHost:** the Address of the remote client
- ***app:** current request application name (out of diameter dictionary)
- ***appid:** current request application id (out of diameter dictionary)
- ***cmd:** current command short naming (out of diameter dictionary) plus *R* as suffix - ie: **CCR*

***req** Diameter request as it comes from the *DiameterClient*.

Special selector format defined in case of groups **req.Path.To.Attribute[\$groupIndex]* or **req.Absolute.Path.To.Attribute[~AnotherAttributeRelativePath(\$valueAnotherAttribute)]*.

Example 1: *~*req.Multiple-Services-Credit-Control.Rating-Group[1]* translates to: value of the group attribute at path *Multiple-Services-Credit-Control.Rating-Group* which is located in the second group (groups start at index 0). Example 2: *~*req.Multiple-Services-Credit-Control.Used-Service-Unit.CC-Input-Octets[~Rating-Group(1)]* which translates to: value of the group attribute at path: *Multiple-Services-Credit-Control.Used-Service-Unit.CC-Input-Octets* where *Multiple-Services-Credit-Control.Used-Service-Unit.Rating-Group* has value of "1".

***rep** Diameter reply going to *DiameterClient*.

***cgreg** Request sent to CGRateS.

***cgrep** Reply coming from CGRateS.

***diamreq** Diameter request generated by CGRateS (ie: *ASR*).

flags Found within processors, special tags enforcing the actions/verbs done on a request. There are two types of flags: **main** and **auxiliary**.

There can be any number of flags or combination of those specified in the list however the flags have priority one against another and only some simultaneous combinations of *main* flags are possible.

The **main** flags will select mostly the action taken on a request.

The **auxiliary** flags only make sense in combination with **main** ones.

Implemented **main** flags are (in order of priority, and not working simultaneously unless specified):

***log** Logs the Diameter request/reply. Can be used together with other *main* actions.

***none** Disable transferring the request from *Diameter* to *CGRateS* side. Used mostly to passively answer *Diameter* requests or troubleshoot (mostly in combination with **log* flag).

***dryrun** Together with not transferring the request on *CGRateS* side will also log the *Diameter* request/reply, useful for troubleshooting.

***auth** Sends the request for authorization on *CGRateS*.

Auxiliary flags available: ***attributes**, ***thresholds**, ***stats**, ***resources**, ***accounts**, ***suppliers**, ***suppliers_ignore_errors**, ***suppliers_event_cost** which are used to influence the auth behavior on *CGRateS* side. More info on that can be found on the **SessionS** component's API behavior.

***initiate** Initiates a session out of request on *CGRateS* side.

Auxiliary flags available: ***attributes**, ***thresholds**, ***stats**, ***resources**, ***accounts** which are used to influence the auth behavior on *CGRateS* side.

***update** Updates a session with the request on *CGRateS* side.

Auxiliary flags available: ***attributes**, ***accounts** which are used to influence the behavior on *CGRateS* side.

***terminate** Terminates a session using the request on *CGRateS* side.

Auxiliary flags available: ***thresholds**, ***stats**, ***resources**, ***accounts** which are used to influence the behavior on *CGRateS* side.

***message** Process the request as individual message charging on *CGRateS* side.

Auxiliary flags available: ***attributes**, ***thresholds**, ***stats**, ***resources**, ***accounts**, ***suppliers**, ***suppliers_ignore_errors**, ***suppliers_event_cost** which are used to influence the behavior on *CGRateS* side.

***event** Process the request as generic event on *CGRateS* side.

Auxiliary flags available: all flags supported by the "SessionSv1.ProcessEvent" generic API

cdrs** Build a CDR out of the request on *CGRateS* side. Can be used simultaneously with other flags (except *dry_run**)

path Defined within field, specifies the path where the value will be written. Possible values:

***vars** Write the value in the special container, **vars*, available for the duration of the request.

***cgreq** Write the value in the request object which will be sent to *CGRateS* side.

***cgrep** Write the value in the reply returned by *CGRateS*.

***rep** Write the value to reply going out on *Diameter* side.

***diamreq** Write the value to request built by *DiameterAgent* to be sent out on *Diameter* side.

type Defined within field, specifies the logic type to be used when writing the value of the field. Possible values:

***none** Pass

- *filler** Fills the values with an empty string
- *constant** Writes out a constant
- *remote_host** Writes out the Address of the remote *DiameterClient* sending us the request
- *variable** Writes out the variable value, overwriting previous one set
- *composed** Writes out the variable value, postpending to previous value set
- *group** Writes out the variable value, postpending to the list of variables with the same path
- *usage_difference** Calculates the usage difference between two arguments passed in the *value*. Requires 2 arguments: *\$stopTime*; *\$startTime*
- *cc_usage** Calculates the usage out of *CallControl* message. Requires 3 arguments: *\$reqNumber*; *\$usedCCTime*; *\$debitInterval*
- *sum** Calculates the sum of all arguments passed within *value*. It supports summing up duration, time, float, int autodetecting them in this order.
- *difference** Calculates the difference between all arguments passed within *value*. Possible value types are (in this order): duration, time, float, int.
- *value_exponent** Calculates the exponent of a value. It requires two values: *\$val*; *\$exp*
- *template** Specifies a template of fields to be injected here. Value should be one of the template ids defined.

2.1.1.2 RadiusAgent

TBD

2.1.1.3 HTTPAgent

TBD

2.1.1.4 DNSAgent

TBD

2.1.1.5 AsteriskAgent

TBD

2.1.1.6 FreeSWITCHAgent

TBD

2.1.1.7 KamailioAgent

TBD

2.1.1.8 EventReaderService

EventReaderService/ERs is a subsystem designed to read events coming from external sources and convert them into internal ones. The converted events are then sent to other CGRateS subsystems, like *SessionS* where further processing logic is applied to them.

The translation between external and internal events is done based on field mapping, defined in *JSON configuration*.

Configuration

The **EventReaderService** is configured within *ers* section from *JSON configuration*.

Sample config

With explanations in the comments:

```
"ers": {
  "enabled": true,                                // enable the service
  "sessions_conns": ["*internal"],                // connection towards SessionS
  "readers": [                                    // list of active_
↳ readers
    {
      "id": "file_reader2",                        // file_reader2 reader
      "run_delay": "-1",                           // reading of events_
↳ it is triggered outside of ERs
      "field_separator": ";",                      // field separator definition
      "type": "*file_csv",                          // type of reader, *file_csv_
↳ can read .csv files
      "flags": [                                    // influence_
↳ processing logic within CGRateS workflow
        "*cdrs",                                    // *cdrs_
↳ will create CDRs
        "*log"                                       // *log will_
↳ log the events to syslog
      ],
      "source_path": "/tmp/ers2/in",                // location of the_
↳ files
      "processed_path": "/tmp/ers2/out",            // move the files here_
↳ once processed
      "content_fields":[                            //_
↳ mapping definition between line index in the file and CGRateS field
        {
          "tag": "OriginID",                          //_
↳ OriginID together with OriginHost will
          "path": "*cgregq.OriginID",                 // uniquely_
↳ identify the session on CGRateS side
          "type": "*variable",
          "value": "~*req.0",q                         // take the_
↳ content from line index 0
          "mandatory": true                            // _
↳ in the request file
        },
        {
          "tag": "RequestType",                       // RequestType_
↳ instructs SessionS

```

(continues on next page)

(continued from previous page)

```

    "path": "*cgreg.RequestType", // about_
    "type": "*variable",
    "value": "~*req.1",
    "mandatory": true
  },
  {
    "tag": "Category", //
    "path": "*cgreg.Category", // and_
    "type": "*constant",
    "value": "call",
    "mandatory": true
  },
  {
    "tag": "Account", //
    "path": "*cgreg.Account",
    "type": "*variable",
    "value": "~*req.3",
    "mandatory": true
  },
  {
    "tag": "Subject", //
    "path": "*cgreg.Subject",
    "type": "*variable",
    "value": "~*req.3",
    "mandatory": true
  },
  {
    "tag": "Destination", // Destination_
    "path": "*cgreg.Destination",
    "type": "*variable",
    "value": "~*req.4:s/0([1-9]\\d+)/+49${1}/",
    "mandatory": true //
  },
  {
    "tag": "AnswerTime", // AnswerTime_
    "path": "*cgreg.AnswerTime",
    "type": "*variable",
    "value": "~*req.5",
    "mandatory": true
  },
  {
    "tag": "Usage", //
    "path": "*cgreg.Usage",
    "type": "*variable",
    "value": "~*req.6",
    "mandatory": true
  }
}

```

↔charging type to apply for the event

↔Category serves for attaching Account

↔RatingProfile to the request

↔Account is required by charging

↔Subject is required by charging

↔is required by charging

↔Additional mediation is performed on number format

↔is required by charging

↔Usage is required by charging

(continues on next page)

(continued from previous page)

```

                                "tag": "HDRExtral",                //  _
↪HDRExtral is transparently stored into CDR
                                "path": "*cgregq.HDRExtral",      //  as extra_
↪field not used by CGRateS
                                "type": "*composed",
                                "value": "~*req.6",
                                "mandatory": true
                                },
                                ],
                                },
                                ]
                                }

```

Config params

Most of the parameters are explained in *JSON configuration*, hence we mention here only the ones where additional info is necessary or there will be particular implementation for *EventReaderService*.

readers List of reader profiles which ERs manages. Simultaneous readers of the same type are possible.

id Reader identifier, used mostly for debug. The id should be unique per each reader since it can influence updating configuration from different *.json* configuration.

type Reader type. Following types are implemented:

***file_csv** Reader for *comma separated* files.

***partial_csv** Reader for *comma separated* where content spans over multiple files.

***flatstore** Reader for *Kamailio/OpenSIPS db_flatstore* files.

***file_xml** Reader for *.xml* formatted files.

***file_fwv** Reader for *fixed width value* formatted files.

***kafka_json_map** Reader for hashmaps within *Kafka_* database.

***sql** Reader for generic content out of *SQL* databases. Supported databases are: *MySQL*, *PostgreSQL* and *MSSQL*.

run_delay Duration interval between consecutive reads from source. If 0 or less, *ERs* relies on external source (ie. Linux inotify for files) for starting the reading process.

concurrent_requests Limits the number of concurrent reads from source (ie: the number of simultaneously opened files).

source_path Path towards the events source

processed_path Optional path for moving the events source to after processing.

xml_root_path Used in case of XML content and will specify the prefix path applied to each xml element read.

tenant Will auto-populate the Tenant within the API calls sent to CGRateS. It has the form of a RSRField. If undefined, default one from *general* section will be used.

timezone Defines the timezone for source content which does not carry that information. If undefined, default one from *general* section will be used.

filters List of filters to pass for the reader to process the event. For the dynamic content (prefixed with ~) following special variables are available:

- *vars** Request related shared variables between processors, populated especially by core functions. The data put in there is not automatically transferred into requests sent to CGRateS, unless instructed inside templates.
- *req** Request read from the source. In case of file content without field name, the index will be passed instead of field source path.
- *hdr** Header values (available only in case of **file_fwv*). In case of file content without field name, the index will be passed instead of field source path.
- *trl** Trailer values (available only in case of **file_fwv*). In case of file content without field name, the index will be passed instead of field source path.

flags Special tags enforcing the actions/verbs done on an event. There are two types of flags: **main** and **auxiliary**.

There can be any number of flags or combination of those specified in the list however the flags have priority one against another and only some simultaneous combinations of *main* flags are possible.

The **main** flags will select mostly the action taken on a request.

The **auxiliary** flags only make sense in combination with **main** ones.

Implemented **main** flags are (in order of priority, and not working simultaneously unless specified):

- *log** Logs the Event read. Can be used together with other *main* flags.
- *none** Disable transferring the Event from *Reader* to *CGRateS* side.
- *dryrun** Together with not transferring the Event on *CGRateS* side will also log it, useful for troubleshooting.
- *auth** Sends the Event for authorization on *CGRateS*.

Auxiliary flags available: ***attributes**, ***thresholds**, ***stats**, ***resources**, ***accounts**, ***suppliers**, ***suppliers_ignore_errors**, ***suppliers_event_cost** which are used to influence the auth behavior on *CGRateS* side. More info on that can be found on the **SessionS** component's API behavior.

***initiate** Initiates a session out of Event on *CGRateS* side.

Auxiliary flags available: ***attributes**, ***thresholds**, ***stats**, ***resources**, ***accounts** which are used to influence the behavior on *CGRateS* side.

***update** Updates a session with the Event on *CGRateS* side.

Auxiliary flags available: ***attributes**, ***accounts** which are used to influence the behavior on *CGRateS* side.

***terminate** Terminates a session using the Event on *CGRateS* side.

Auxiliary flags available: ***thresholds**, ***stats**, ***resources**, ***accounts** which are used to influence the behavior on *CGRateS* side.

***message** Process the Event as individual message charging on *CGRateS* side.

Auxiliary flags available: ***attributes**, ***thresholds**, ***stats**, ***resources**, ***accounts**, ***suppliers**, ***suppliers_ignore_errors**, ***suppliers_event_cost** which are used to influence the behavior on *CGRateS* side.

***event** Process the Event as generic event on *CGRateS* side.

Auxiliary flags available: all flags supported by the "SessionSv1.ProcessEvent" generic API

cdrs** Build a CDR out of the Event on *CGRateS* side. Can be used simultaneously with other flags (except *dry_run**)

path Defined within field, specifies the path where the value will be written. Possible values:

- *vars** Write the value in the special container, **vars*, available for the duration of the request.
- *cgreg** Write the value in the request object which will be sent to *CGRateS* side.

***hdr** Header values (available only in case of **file_fwv*). In case of file content without field name, the index will be passed instead of field source path.

***trl** Trailer values (available only in case of **file_fwv*). In case of file content without field name, the index will be passed instead of field source path.

type Defined within field, specifies the logic type to be used when writing the value of the field. Possible values:

***none** Pass

***filler** Fills the values with an empty string

***constant** Writes out a constant

***remote_host** Writes out the Address of the remote host sending us the Event

***variable** Writes out the variable value, overwriting previous one set

***composed** Writes out the variable value, postpending to previous value set

***usage_difference** Calculates the usage difference between two arguments passed in the *value*. Requires 2 arguments: *\$stopTime;\$startTime*

***sum** Calculates the sum of all arguments passed within *value*. It supports summing up duration, time, float, int autodetecting them in this order.

***difference** Calculates the difference between all arguments passed within *value*. Possible value types are (in this order): duration, time, float, int.

***value_exponent** Calculates the exponent of a value. It requires two values: *\$val;\$exp*

***template** Specifies a template of fields to be injected here. Value should be one of the template ids defined.

2.1.2 SessionS

TBD

2.1.3 RALs

TBD

2.1.4 CDRs

TBD

2.1.5 AttributeS

AttributeS is a standalone subsystem within **CGRateS** and it is the equivalent of a key-value store. It is accessed via CGRateS RPC APIs.

As most of the other subsystems, it is performance oriented, stored inside *DataDB* but cached inside the *cgr-engine* process. Caching can be done dynamically/on-demand or at start-time/precached and it is configurable within *cache* section in the *JSON configuration*.

2.1.5.1 Selection

It is able to process generic events (hashmaps) and decision for matching it is outsourced to *FilterS*.

If there are multiple profiles (configurations) matching, the one with highest *Weight* will be the winner. There can be only one *AttributeProfile* processing the event per *process run*. If one configures multiple *process runs* either in *JSON configuration* or as parameter to the *.ProcessEvent* API call, the output event from one *process run* will be forwarded as input to the next selected profile. There will be independent *AttributeProfile* selection performed for each run, hence the event fields modified in one run can be applied as filters to the next *process run*, giving out the possibility to chain *AttributeProfiles* and have multiple replacements with a minimum of performance penalty (in-memory matching).

2.1.5.2 Parameters

AttributesS

AttributesS is the **CGRateS** component responsible of handling the *AttributeProfiles*.

It is configured within **attributes** section from *JSON configuration* via the following parameters:

enabled Will enable starting of the service. Possible values: <true|false>.

indexed_selects Enable profile matching exclusively on indexes. If not enabled, the *ResourceProfiles* are checked one by one which for a larger number can slow down the processing time. Possible values: <true|false>.

string_indexed_fields Query string indexes based only on these fields for faster processing. If commented out, each field from the event will be checked against indexes. If uncommented and defined as empty list, no fields will be checked.

prefix_indexed_fields Query prefix indexes based only on these fields for faster processing. If defined as empty list, no fields will be checked.

nested_fields Applied when all event fields are checked against indexes, and decides whether subfields are also checked.

process_runs Limit the number of loops when processing an Event. The event loop is however clever enough to stop when the same processing occurs or no more additional profiles are matching, so higher numbers are ignored if not needed.

AttributeProfile

Represents the configuration for a group of attributes applied.

Tenant The tenant on the platform (one can see the tenant as partition ID)

ID Identifier for the *AttributeProfile*, unique within a *Tenant*

Context A list of *contexts* applying to this profile. A *context* is usually associated with a logical phase during event processing (ie: **sessions* or **cdrs* for events parsed by *SessionS* or *CDRs*)

FilterIDs List of *FilterProfiles* which should match in order to consider the *AttributeProfile* matching the event.

ActivationInterval The time interval when this profile becomes active. If undefined, the profile is always active. Other options are start time, end time or both.

Blocker In case of multiple *process runs* are allowed, this flag will break further processing.

Weight Used in case of multiple profiles matching an event. The higher, the better (0 has lowest possible priority).

Attributes List of *Attribute* objects part of this profile.

Attribute

FilterIDs List of *FilterProfiles* which should match in order to consider the *Attribute* matching the event.

Path Identifying the targeted absolute path within the processed event.

Type Represents the type of substitution which will be performed on the Event. The following *Types* are available:

***constant** The *Value* is a constant value, it will just set the *FieldName* to this value as it is.

***variable** The *Value* is a *RSRParser* which will be able to capture the value out of one or more fields in the event (also combined with other constants) and write it to *Path*.

***composed** Same as **variable* but instead of overwriting *Path*, it will append to it.

***usage_difference** Will calculate the duration difference between two field names defined in the *Value*. If the number of fields in the *Value* are different than 2, it will error.

***sum** Will sum up the values in the *Value*.

***value_exponent** Will compute the exponent of the first field in the *Value*.

Value The value which will be set for *Path*. It can be a list of *RSRParser*s capturing even from multiple sources in the same event. If the *Value* is **remove* the field with *Path* will be removed from *Event*

Inline Attribute

In order to facilitate quick attribute definition (without the need of separate *AttributeProfile*), one can define attributes directly as *AttributeIDs* following the special format.

Inline filter format:

```
attributeType:attributePath:attributeValue
```

Example:

```
*constant:*req.RequestType:*prepaid
```

2.1.5.3 Use cases

- Fields aliasing * Number portability (replacing a dialed number with it's translation) * Roaming (using *Category* to point out the zone where the user is roaming in so we can apply different rating or consume out of restricted account bundles).
- Appending new fields * Adding separate header with location information * Adding additional rating information (ie: SMS only contains origin+destination, add *Tenant*, *Account*, *Subject*, *RequestType*) * Using as query language (ie: append user password for a given user so we can perform authorization on SIP Proxy side).

2.1.6 ChargersS

ChargerS is a **CGRateS** subsystem designed to produce billing runs via *DerivedCharging* mechanism.

It works as standalone component of **CGRateS**, accessible via **CGRateS** RPC via a rich set of *APIs*. As input **ChargerS** is capable of receiving generic events (hashmaps) with dynamic types for fields.

ChargerS is an **important** part of the charging process within **CGRateS** since with no *ChargingProfile* matching, there will be no billing run performed.

2.1.6.1 DerivedCharging

Is a process of receiving an event as input and *deriving* that into multiples (unlimited) out. The *derived* event will be a standalone clone of original with possible modifications of individual event fields. In case of billing, this will translate into multiple Events or CDRs being billed simultaneously for the same input.

2.1.6.2 Processing logic

For the received *Event* we will retrieve the list of matching *ChargingProfiles* via *:ref: 'FilterS'*. *These profiles will be then ordered based on their *Weight* - higher *Weight* will have more priority. If no profile will match due to *Filter* or *ActivationInterval*, *NOT_FOUND* will be returned back to the RPC client.

Each *ChargingProfile* matching the *Event* will produce a standalone event based on configured *RunID*. These events will each have a special field added (or overwritten), the *RunID*, which is taken from the applied *ChargingProfile*.

If *AttributeIDs* are different than **none*, the newly created *Event* will be sent to AttributeS and fields replacement will be performed based on the logic there. If the *AttributeIDs* is populated, these profile IDs will be selected directly for faster processing, otherwise (if empty) the *AttributeProfiles* will be selected using *FilterS*.

2.1.6.3 Parameters

ChargerProfile

A *ChargerProfile* is the configuration producing the *DerivedCharging* for the Event received. It's made of the following fields:

Tenant Is the tenant on the platform (one can see the tenant as partition ID)

ID Identifier for the ChargerProfile, unique within a *Tenant*.

FilterIDs List of *FilterProfiles* which should match in order to consider the ChargerProfile matching the event.

ActivationInterval Is the time interval when this profile becomes active. If undefined, the profile is always active. Other options are start time, end time or both.

RunID The identifier for a single bill run / charged output *Event*.

AttributeIDs List of *AttributeProfileIDs* which will be applied for the output *Event* in order to change some of it's fields. If empty, the list is discovered via FilterS (*AttributeProfiles* matching the event). If **none*, no *AttributeProfile* will be applied, event will be a simple clone of the one at input with just **RunID* being different.

Weight Used in case of multiple profiles matching an event. The higher, the better (0 has lowest possible priority).

2.1.6.4 Use cases

- Calculating standard charges for the *Customer* calling as well as for the *Reseller/Distributor*. One can build chains of charging rules if multiple *Resellers* are involved.
- Calculating revenue based on *Customer vs Supplier* pricing.
- Calculating pricing for multiple *Suppliers* for revenue protection.
- Adding *local vs mobile* charges for *premium numbers* when accessed from mobile headsets.
- etc.

2.1.7 ResourceS

ResourceS is a standalone subsystem part of the **CGRateS** infrastructure, designed to allocate virtual resources for the generic *Events* (hashmaps) it receives.

Both receiving of *Events* as well as operational commands on the virtual resources is performed via a complete set of CGRateS RPC APIs.

Due it's real-time nature, **ResourceS** are designed towards high throughput being able to process thousands of *Events* per second. This is doable since each *Resource* is a very light object, held in memory and eventually backed up in *DataDB*.

2.1.7.1 Processing logic

When a new *Event* is received, **ResourceS** will pass it to *FilterS* in order to find all *Resource* objects matching the *Event*.

As a result of the selection process we will further get an ordered list of *Resource* which are matching the *Event* and are active at the request time.

Depending of the *RPC API* used, we will have the following behavior further:

ResourcesForEvent Will simply return the list of *Resources* matching so far.

AuthorizeResources Out of *Resources* matching, ordered based on *Weight*, it will use the first one with available units to authorize the request. Returns *RESOURCE_UNAVAILABLE* error back in case of no available units found. No actual allocation is performed.

AllocateResource All of the *Resources* matching the event will be operated and requested units will be deducted, independent of being available or going on negative. The first one with value higher or equal to zero will be responsible of allocation and it's message will be returned as allocation message. If no allocation message is defined for the allocated resource, it's ID will be returned instead.

If no resources are allocated *RESOURCE_UNAVAILABLE* will be returned as error.

ReleaseResource Will release all the previously allocated resources for an *UsageID*. If *UsageID* is not found (which can be the case of restart), will perform a standard search via *FilterS* and try to dealocate the resources matching there.

Depending on configuration each *Resource* can be backed up regularly and asynchronously to *DataDB* so it can survive process restarts.

After each resource modification (allocation or release) the *ThresholdS* will be notified with the *Resource* itself where mechanisms like notifications or fraud-detection can be triggered.

2.1.7.2 Parameters

ResourceS

ResourceS is the **CGRateS** component responsible of handling the *Resources*.

It is configured within **resources** section from *JSON configuration* via the following parameters:

enabled Will enable starting of the service. Possible values: <true|false>.

store_interval Time interval for backing up the stats into *DataDB*.

thresholds_conns Connections IDs towards *ThresholdS* component. If not defined, there will be no notifications sent to *ThresholdS* on *Resource* changes.

indexed_selects Enable profile matching exclusively on indexes. If not enabled, the *ResourceProfiles* are checked one by one which for a larger number can slow down the processing time. Possible values: <true/false>.

string_indexed_fields Query string indexes based only on these fields for faster processing. If commented out, each field from the event will be checked against indexes. If uncommented and defined as empty list, no fields will be checked.

prefix_indexed_fields Query prefix indexes based only on these fields for faster processing. If defined as empty list, no fields will be checked.

nested_fields Applied when all event fields are checked against indexes, and decides whether subfields are also checked.

ResourceProfile

The **ResourceProfile** is the configuration of a *Resource*. This will be performed over CGRateS RPC APIs or *.csv* files. A profile is comprised out of the following parameters:

Tenant The tenant on the platform (one can see the tenant as partition ID).

ID Identifier for the *ResourceProfile*, unique within a *Tenant*.

FilterIDs List of *FilterProfiles* which should match in order to consider the *ResourceProfile* matching the event.

ActivationInterval The time interval when this profile becomes active. If undefined, the profile is always active. Other options are start time, end time or both.

UsageTTL Autoexpire resource allocation after this time duration.

Limit The number of allocations this resource is entitled to.

AllocationMessage The message returned when this resource is responsible for allocation.

Blocker When specified, no further resources are processed after this one.

Stored Enable offline backups for this resource

Weight Order the *Resources* matching the event. Higher value - higher priority.

ThresholdIDs List of *ThresholdProfiles* targetted by the *Resource*. If empty, the match will be done in *ThresholdS* component.

ResourceUsage

A **ResourceUsage** represents a counted allocation within a *Resource*. The following parameters are present within:

Tenant The tenant on the platform (one can see the tenant as partition ID).

ID Identifier for the *ResourceUsage*.

ExpiryTime Exact time when this allocation expires.

Units Number of units allocated by this *ResourceUsage*.

2.1.7.3 Use cases

- Monitor resources for a group of accounts(ie. based on a special field in the events).
- Limit the number of CPS for a destination/supplier/account (done via UsageTTL of 1s).
- Limit resources for a destination/supplier/account/time of day/etc.

2.1.8 SupplierS

SupplierS is a standalone subsystem within **CGRateS** responsible to compute a list of suppliers which can be used for a specific event received to process. It is accessed via CGRateS RPC APIs.

As most of the other subsystems, it is performance oriented, stored inside *DataDB* but cached inside the *cgr-engine* process. Caching can be done dynamically/on-demand or at start-time/precached and it is configurable within *cache* section in the *JSON configuration*.

2.1.8.1 Processing logic

When a new *Event* is received, **SupplierS** will pass it to *FilterS* in order to find all *SupplierProfiles* matching the *Event*.

As a result of the selection process we will get a single *SupplierProfile* matching the *Event*, is active at the *EventTime* and has a higher priority than the other matching *SupplierProfiles*.

Depending on the *Strategy* defined in the *SupplierProfile*, further steps will be taken (ie: query cost, stats, ordering) for each of the individual *SupplierIDs* defined within the *SupplierProfile*.

2.1.8.2 APIs logic

GetSupplierProfilesForEvent

Given the *Event* it will return a list of ordered *SupplierProfiles* matching at the *EventTime*.

This API is useful to test configurations.

GetSuppliers

Will return a list of *Suppliers* from within a *SupplierProfile* ordered based on *Strategy*.

2.1.8.3 Parameters

SupplierS

SupplierS is the **CGRateS** component responsible for handling the *SupplierProfiles*.

It is configured within **suppliers** section from *JSON configuration* via the following parameters:

enabled Will enable starting of the service. Possible values: <true|false>.

indexed_selects Enable profile matching exclusively on indexes. If not enabled, the *SupplierProfiles* are checked one by one which for a larger number can slow down the processing time. Possible values: <true|false>.

string_indexed_fields Query string indexes based only on these fields for faster processing. If commented out, each field from the event will be checked against indexes. If defined as empty list, no fields will be checked.

prefix_indexed_fields Query prefix indexes based only on these fields for faster processing. If defined as empty list, no fields will be checked.

nested_fields Applied when all event fields are checked against indexes, and decides whether subfields are also checked.

attributes_conns Connections to AttributeS for altering events before supplier queries. If undefined, fields modifications are disabled.

resources_conns Connections to ResourceS for **res* sorting, empty to disable functionality.

stats_conns Connections to StatS for **stats* sorting, empty to disable stats functionality.

default_ratio Default ratio used in case of **load* strategy

SupplierProfile

Contains the configuration for a set of suppliers which will be returned in case of match. Following fields can be defined:

Tenant The tenant on the platform (one can see the tenant as partition ID).

ID The profile identifier.

FilterIDs List of *FilterProfileIDs* which should match in order to consider the profile matching the event.

ActivationInterval The time interval when this profile becomes active. If undefined, the profile is always active. Other options are start time, end time or both.

Sorting Sorting strategy applied when ordering the individual *Suppliers* defined below. Possible values are:

***weight** Classic method of statically sorting the suppliers based on their priority.

***lc** LeastCost will sort the suppliers based on their cost (lowest cost will have higher priority). If two suppliers will be identical as cost, their *Weight* will influence the sorting further. If *AccountIDs* will be specified, bundles can be also used during cost calculation, the only condition is that the bundles should cover complete usage.

The following fields are mandatory for cost calculation: *Account/Subject, Destination, SetupTime*. *Usage* is optional and if present in event, it will be used for the cost calculation.

***hc** HighestCost will sort the suppliers based on their cost (higher cost will have higher priority). If two suppliers will be identical as cost, their *Weight* will influence the sorting further.

The following fields are mandatory for cost calculation: *Account/Subject, Destination, SetupTime*. *Usage* is optional and if present in event, it will be used for the cost calculation.

***qos** QualityOfService strategy will sort the suppliers based on their stats. It takes the *StatIDs* to check from the supplier *StatIDs* definition. The metrics used as part of sorting are to be defined in *SortingParameters* field below. If Stats are missing the metrics defined in *SortingParameters* defaults for those will be populated for order (10000000 as PDD and -1 for the rest).

***reas** ResourceAscendentSorter will sort the suppliers based on their resource usage, lowest usage giving higher priority. The resources will be queried for each supplier based on its *ResourceIDs* field and the final usage for each supplier will be given by the sum of all the resource usages queried.

***reds** ResourceDescendentSorter will sort the suppliers based on their resource usage, highest usage giving higher priority. The resources will be queried for each supplier based on its *ResourceIDs* field and the final usage for each supplier will be given by the sum of all the resource usages queried.

***load** LoadDistribution will sort the suppliers based on their load. An important parameter is the **ratio* which is defined as *supplierID:Ratio* within the *SortingParameters*. If no *supplierID* is present within *SortingParameters*, the system will look for **default* or *fallback* in the configuration to *default_ratio* within *JSON configuration*. The **ratio* will specify the probability to get traffic on a *Supplier*, the higher the **ratio* more chances will a *Supplier* get for traffic.

The load will be calculated out of the *StatIDs* parameter of each *Supplier*. It is possible to also specify there directly the metric being used in the format *StatID:MetricID*. If only *StatID* is instead specified, all metrics will be summed to get the final value.

SortingParameters Will define additional parameters for each strategy. Following extra parameters are available (based on strategy):

***qos** List of metrics to be used for sorting in order of importance.

Weight Priority in case of multiple *SupplierProfiles* matching an *Event*. Higher *Weight* will have more priority.

Suppliers List of *Supplier* objects which are part of this *SupplierProfile*

Supplier

The *Supplier* represents one supplier within the *SupplierProfile*. Following parameters are defined for it:

ID Supplier ID, will be returned via APIs. Should be known on the remote side and match some business logic (ie: gateway id or directly an IP address).

FilterIDs List of *FilterProfileIDs* which should match in order to consider the *Supplier* in use/active.

AccountIDs List of account IDs which should be checked in case of some strategies (ie: **lc*, **hc*).

RatingPlanIDs List of RatingPlanIDs which should be checked in case of some strategies (ie: **lc*, **hc*).

ResourceIDs List of ResourceIDs which should be checked in case of some strategies (ie: **reas* or **reds*).

StatIDs List of StatIDs which should be checked in case of some strategies (ie: *qos* or **load*). *Can also be defined as *StatID:MetricID.*

Weight Used for sorting in some strategies (ie: **weight*, **lc* or **hc*).

Blocker No more suppliers are provided after this one.

SupplierParameters Container which is transparently passed to the remote client to be used in it's own logic (ie: gateway prefix stripping or other gateway parameters).

2.1.8.4 Use cases

- Calculate LCR directly by querying APIs (GetSuppliers).
- LCR system together with *Kamailio dispatcher* module where the *SupplierID* within *CGRateS* will be used as dispatcher set within *Kamailio*.
- LCR system together with *OpenSIPS* drouting module where the *SupplierID* within *CGRateS* will be used as drouting carrier id.
- LCR system together with *FreeSWITCH* or *Asterisk* where the *SupplierID* within *CGRateS* will be used as gateway ID within the dialplan of *FreesWITCH* or *Asterisk*.

2.1.9 StatS

StatS is a standalone subsystem part of the **CGRateS** infrastructure, designed to aggregate and calculate statistical metrics for the generic *Events* (hashmaps) it receives.

Both receiving of *Events* as well as *Metrics* displaying is performed via a complete set of *CGRateS* RPC APIs.

Due it's real-time nature, **StatS** are designed towards high throughput being able to process thousands of *Events* per second. This is doable since each *StatQueue* is a very light object, held in memory and eventually backed up in *DataDB*.

2.1.9.1 Processing logic

When a new *Event* is received, **StatS** will pass it to *FilterS* in order to find all *StatProfiles* matching the *Event*.

As a result of the selection process we will further get an ordered list of *StatProfiles* which are matching the *Event* and are active at the request time.

For each of these profiles we will further calculate the metrics it has configured for the *Event* received. If *ThresholdIDs* are not **none*, we will include the *Metrics* into special *StatUpdate* events, defined internally, and pass them further to the ThresholdS for processing.

Depending on configuration each *StatQueue* can be backed up regularly and asynchronously to DataDB so it can survive process restarts.

2.1.9.2 Parameters

StatS

StatS is the **CGRateS** component responsible of handling the *StatQueues*.

It is configured within **stats** section from *JSON configuration* via the following parameters:

enabled Will enable starting of the service. Possible values: <true|false>.

store_interval Time interval for backing up the stats into *DataDB*.

store_uncompressed_limit After this limit is hit the events within *StatQueue* will be stored aggregated.

thresholds_conns Connections IDs towards *ThresholdS* component. If not defined, there will be no notifications sent to *ThresholdS* on *StatQueue* changes.

indexed_selects Enable profile matching exclusively on indexes. If not enabled, the *StatQueues* are checked one by one which for a larger number can slow down the processing time. Possible values: <true|false>.

string_indexed_fields Query string indexes based only on these fields for faster processing. If commented out, each field from the event will be checked against indexes. If uncommented and defined as empty list, no fields will be checked.

prefix_indexed_fields Query prefix indexes based only on these fields for faster processing. If defined as empty list, no fields will be checked.

nested_fields Applied when all event fields are checked against indexes, and decides whether subfields are also checked.

StatQueueProfile

Is made of the following fields:

Tenant The tenant on the platform (one can see the tenant as partition ID).

ID Identifier for the *StatQueueProfile*, unique within a *Tenant*.

FilterIDs List of *FilterProfileIDs* which should match in order to consider the profile matching the event.

ActivationInterval The time interval when this profile becomes active. If undefined, the profile is always active. Other options are start time, end time or both.

QueueLength Maximum number of items stored in the queue. Once the *QueueLength* is reached, new items entering will cause oldest one to be dropped (FIFO mode).

TTL Time duration causing items in the queue to expire and be removed automatically from the queue.

Metrics List of statistical metrics to build for items within this *StatQueue*. See [below](#statqueue-metrics) for possible values here.

ThresholdIDs List of threshold IDs to check on when new items are updating the queue metrics.

Blocker Do not process further *StatQueues*.

Stored Enable offline backups for this *StatQueue*

Weight Order the *StatQueues* matching the event. Higher value - higher priority.

MinItems Display metrics only if the number of items in the queue is higher than this.

StatQueue Metrics

Following metrics are implemented:

***asr** Answer-seizure ratio. Relies on *AnswerTime* field in the *Event*.

***acd** Average call duration. Uses *AnswerTime* and *Usage* fields in the *Event*.

***tcd** Total call duration. Uses *Usage* out of *Event*.

***acc** Average call cost. Uses *Cost* field out of *Event*.

***tcc** Total call cost. Uses *Cost* field out of *Event*.

***pdd** Post dial delay <<https://www.voip-info.org/pdd/>>. Uses *PDD* field in the event.

***ddc** Distinct destination count will keep the number of unique destinations found in *Events*. Relies on *Destination* field in the *Event*.

***sum** Generic metric to calculate mathematical sum for a specific field in the *Events*. Format: < *sum#FieldName >.

***average** Generic metric to calculate the mathematical average of a specific field in the *Events*. Format: < *average#FieldName >.

***distinct** Generic metric to return the distinct number of appearance of a field name within *Events*. Format: < *distinct#FieldName >.

2.1.9.3 Use cases

- Aggregate various traffic metrics for traffic transparency.
- Revenue assurance applications.
- Fraud detection by aggregating specific billing metrics during sensitive time intervals (**acc*, **tcc*, **tcd*).
- Building call patterns.
- Building statistical information to train systems capable of artificial intelligence.
- Building quality metrics used in traffic routing.

2.1.10 ThresholdS

ThresholdS is a standalone subsystem within **CGRateS** responsible to execute a list of *Actions* for a specific event received to process. It is accessed via CGRateS RPC APIs.

As most of the other subsystems, it is performance oriented, stored inside *DataDB* but cached inside the *cgr-engine* process. Caching can be done dynamically/on-demand or at start-time/precached and it is configurable within *cache* section in the *JSON configuration*.

2.1.10.1 Processing logic

When a new *Event* is received, **ThresholdS** will pass it to *FilterS* in order to find all *SupplierProfiles* matching the *Event*.

As a result of the selection process we will get a list of *Thresholds* matching the *Event* and are active at the *EventTime*.

2.1.10.2 APIs logic

GetThresholdIDs

Returns a list of *ThresholdIDs* configured on a *Tenant*.

GetThresholdsForEvent

Returns a list of *Thresholds* matching the event.

GetThreshold

Returns a specific *Threshold* based on it's *Tenant* and *ID*.

ProcessEvent

Technically processes the *Event*, executing all the *Actions* configured within all the matching *Thresholds*.

2.1.10.3 Parameters

ThresholdS

ThresholdS is the **CGRateS** component responsible for handling the *Thresholds*.

It is configured within **thresholds** section from *JSON configuration* via the following parameters:

enabled Will enable starting of the service. Possible values: <true|false>.

store_interval Time interval for backing up the thresholds into *DataDB*.

indexed_selects Enable profile matching exclusively on indexes. If not enabled, the *Thresholds* are checked one by one which for a larger number can slow down the processing time. Possible values: <true|false>.

string_indexed_fields Query string indexes based only on these fields for faster processing. If commented out, each field from the event will be checked against indexes. If defined as empty list, no fields will be checked.

prefix_indexed_fields Query prefix indexes based only on these fields for faster processing. If defined as empty list, no fields will be checked.

nested_fields Applied when all event fields are checked against indexes, and decides whether subfields are also checked.

ThresholdProfile

Contains the configuration to create a *Threshold*. Following fields can be defined:

Tenant The tenant on the platform (one can see the tenant as partition ID).

ID The profile identifier.

FilterIDs List of *FilterProfileIDs* which should match in order to consider the profile matching the event.

ActivationInterval The time interval when this profile becomes active. If undefined, the profile is always active. Other options are start time, end time or both.

MaxHits Limit number of hits for this threshold. Once this is reached, the threshold is considered disabled.

MinHits Only execute actions after this number is reached.

MinSleep Disable the threshold for consecutive hits for the duration of *MinSleep*.

Blocker Do not process thresholds who's *Weight* is lower.

Weight Sorts the execution of multiple thresholds matching the event. The higher the *Weight* is, the higher the priority to be executed.

ActionIDs List of *Actions* to execute for this threshold.

Async If true, do not wait for actions to complete.

Threshold

Represents one threshold, instantiated from a *ThresholdProfile*. It contains the following fields:

Tenant The tenant on the platform (one can see the tenant as partition ID).

ID The threshold identifier.

Hits Number of hits so far.

Snooze If initialized, it will contain the time when this threshold will become active again.

2.1.10.4 Use cases

- Improve network transparency and automatic reaction to outages monitoring stats produced by StatS.
- Monitor active channels used by a supplier/customer/reseller/destination/weekends/etc out of *ResourceS* events.
- Monitor balance consumption out of *Account* events.
- Monitor calls out of *CDRs* events or *SessionS*.
- Fraud detection with automatic mitigation based of all events mentioned above.

2.1.11 FilterS

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

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

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

A subsystem can use a *Filter* via *FilterProfile* or in-line (ad-hock in the same place where subsystem profile is defined).

2.1.11.1 Filter profile

Definition:

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

A Filter profile can be shared between multiple subsystem profile definitions.

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

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

2.1.11.2 Filter rule

Definition:

```
type FilterRule struct {
    Type          string          // Filter type
    Element       string          // Name of the field providing us the
    ↪Values to check (used in case of some )
    Values        []string       // Filter definition
}
```

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

The following types are implemented:

string Will match in full the *Element* with at least one value defined inside *Values*. Any of the values matching will have the FilterRule as *matched*.

***notstring** Is the negation of **string*.

***prefix** Will match at beginning of *Element* one of the values defined inside *Values*.

***notprefix** Is the negation of **prefix*.

***suffix** Will match at end of *Element* one of the values defined inside *Values*.

notsuffix Is the negation of **suffix*.

***empty** Will make sure that *Element* is empty or it does not exist in the event.

***notempty** Is the negation of **empty*.

***exists** Will make sure that *Element* exists in the event.

***notexists** Is the negation of **exists*.

***timings** Will compare the time contained in *Element* with one of the TimingIDs defined in *Values*.

***nottimings** Is the negation of **timings*.

***destinations** Will make sure that the *Element* is a prefix contained inside one of the destination IDs as *Values*.

***notdestinations** Is the negation of **destinations*.

***rsr** Will match the *RSRRules* defined in *Values*.

notrsr Is the negation of **rsr*.

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

2.1.11.3 Inline Filter

In order to facilitate quick filter definition (without the need of separate FilterProfile), one can define filters directly as FilterIDs following the special format.

Inline filter format:

```
filterType:fieldName:fieldValue
```

Example:

```
*string:WebsiteName:CGRateS.org
```

2.1.11.4 Subsystem profiles selection based on Filters

When a subsystem will process an event it will need to find fast enough (close to real-time and most preferably with constant speed) all the profiles having filters matching the event. For low number of profiles (tens of) we can go through all available profiles and check their filters but as soon as the number of profiles is growing, processing time will exponentially grow also. As an example, the *AttributeS* need to deal with 20 mil+ profiles in case of number portability implementation.

In order to guarantee constant processing time - **O(1)** - *CGRateS* will use internally a profile selection mechanism based on indexed filters which can be enabled within *.json* configuration file via *indexed_selects*. When *indexed_selects* is disabled, the indexes will not be used at all and profiles will be checked one by one. On the other hand, if *indexed_selects* is enabled, each FilterProfile needs to have at least one **string* or **prefix* type in order to be visible to the indexes (otherwise being completely ignored).

The following settings are further applied once *indexed_selects* is enabled:

string_indexed_fields list of field names in the event which will be checked against string indexes (defaults to nil which means check all fields)

prefix_indexed_fields list of field names in the event which will be checked against prefix indexes (default is empty, hence prefix matching is disabled inside indexes - small optimization since for prefixes there are multiple queries done for one field)

2.1.12 DispatcherS

TBD

2.1.13 SchedulerS

TBD

2.1.14 CDRe

TBD

2.1.15 APlerS

TBD

2.1.16 LoaderS

TBD

2.1.17 DataDB

TBD

2.1.18 StorDB

TBD

2.1.19 CacheS

TBD

2.2 cgr-console

Command line tool used to interface via the APIs implemented within :ref:cgr-engine.

Configurable via command line arguments.

```
$ cgr-console -help
Usage of cgr-console:
  -ca_path string
        path to CA for tls connection(only for self sign certificate)
  -crt_path string
        path to certificate for tls connection
  -key_path string
        path to key for tls connection
  -reply_timeout int
        Reply timeout in seconds (default 300)
  -rpc_encoding string
        RPC encoding used <*gob|*json> (default "*json")
  -server string
        server address host:port (default "127.0.0.1:2012")
  -tls
        TLS connection
  -verbose
        Show extra info about command execution.
  -version
        Prints the application version.
```

Hint: # cgr-console status

2.3 cgr-loader

Tool used to load/import TariffPlan data into CGRateS databases.

Can be used to:

- load TariffPlan data from **csv files** to **DataDB**.
- import TariffPlan data from **csv files** to **StorDB** as offline data. `-to_stordb -tpid`
- import TariffPlan data from **StorDB** to **DataDB**. `-from_stordb -tpid`

Customisable through the use of *JSON configuration* or command line arguments (higher prio).

```
$ cgr-loader -h
Usage of cgr-loader:
  -api_key string
        Api Key used to comosed ArgDispatcher
  -caches_address string
        Caches component to contact for cache reloads, empty to disable automatic_
↪cache reloads (default "*localhost")
  -caching string
        Caching strategy used when loading TP
  -config_path string
        Configuration directory path.
  -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 "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|*mongo> (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")
  -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.
  -field_sep string
        Separator for csv file (by default "," is used) (default ",")
  -flush_stordb
        Remove tariff plan data for id from the database
  -from_stordb
        Load the tariff plan from storDb to dataDb
  -import_id string
        Uniquely identify an import/load, postpended to some automatic fields
  -path string
```

(continues on next page)

(continued from previous page)

```

    The path to folder containing the data files (default "./")
-recursive
    Loads data from folder recursive.
-redis_sentinel string
    The name of redis sentinel
-remove
    Will remove instead of adding data from DB
-route_id string
    RouteID used to comosed ArgDispatcher
-rpc_encoding string
    RPC encoding used <*gob|*json> (default "*json")
-scheduler_address string
    (default "*localhost")
-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.
-storDb_port string
    The storDb port to bind to. (default "3306")
-storDb_type string
    The type of the storDb database <*mysql|*postgres|*mongo> (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>
-to_storDb
    Import the tariff plan from files to storDb
-tpid string
    The tariff plan ID from the database
-verbose
    Enable detailed verbose logging output
-version
    Prints the application version.

```

2.4 cgr-migrator

Command line migration tool.

Customisable through the use of *JSON configuration* or command line arguments (higher prio).

```

$ cgr-migrator -h
Usage of cgr-migrator:
-config_path string
    Configuration directory path.
-datadb_host string
    the DataDB host (default "127.0.0.1")
-datadb_name string
    the name/number of the DataDB (default "10")
-datadb_passwd string
    the DataDB password
-datadb_port string
    the DataDB port (default "6379")

```

(continues on next page)

(continued from previous page)

```

-databdb_type string
    the type of the DataDB Database <*redis|*mongo> (default "redis")
-databdb_user string
    the DataDB user (default "cgrates")
-dbdatab_encoding string
    the encoding used to store object Data in strings (default "msgpack")
-dry_run
    parse loaded data for consistency and errors, without storing it
-exec string
    fire up automatic migration <*set_versions|*cost_
↔details|*accounts|*actions|*action_triggers|*action_plans|*shared_
↔groups|*filters|*stordb|*databdb>
-out_databdb_host string
    output DataDB host to connect to (default "*databdb")
-out_databdb_name string
    output DataDB name/number (default "*databdb")
-out_databdb_passwd string
    output DataDB password (default "*databdb")
-out_databdb_port string
    output DataDB port (default "*databdb")
-out_databdb_type string
    output DataDB type <*redis|*mongo> (default "*databdb")
-out_databdb_user string
    output DataDB user (default "*databdb")
-out_dbdatab_encoding string
    the encoding used to store object Data in strings in move mode (default
↔"*databdb")
-out_redis_sentinel string
    the name of redis sentinel (default "*databdb")
-out_stordb_host string
    output StorDB host (default "*stordb")
-out_stordb_name string
    output StorDB name/number (default "*stordb")
-out_stordb_passwd string
    output StorDB password (default "*stordb")
-out_stordb_port string
    output StorDB port (default "*stordb")
-out_stordb_type string
    output StorDB type for move mode <*mysql|*postgres|*mongo> (default "*stordb")
-out_stordb_user string
    output StorDB user (default "*stordb")
-redis_sentinel string
    the name of redis sentinel
-stordb_host string
    the StorDB host (default "127.0.0.1")
-stordb_name string
    the name/number of the StorDB (default "cgrates")
-stordb_passwd string
    the StorDB password
-stordb_port string
    the StorDB port (default "3306")
-stordb_type string
    the type of the StorDB Database <*mysql|*postgres|*mongo> (default "mysql")
-stordb_user string
    the StorDB user (default "cgrates")
-verbose
    enable detailed verbose logging output

```

(continues on next page)

(continued from previous page)

```
-version
    prints the application version
```

2.5 cgr-tester

Command line stress testing tool configurable via command line arguments.

```
$ cgr-tester -h
Usage of cgr-tester:
  -category string
        The Record category to test. (default "call")
  -config_path string
        Configuration directory path.
  -cpuprofile string
        write cpu profile to file
  -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 "10")
  -datadb_pass 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")
  -destination string
        The destination to use in queries. (default "1002")
  -file_path string
        read requests from file with path
  -json
        Use JSON RPC
  -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.
  -redis_sentinel string
        The name of redis sentinel
  -req_separator string
        separator for requests in file (default "\n\n")
  -runs int
        stress cycle number (default 100000)
  -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")
  -usage string
```

(continues on next page)

(continued from previous page)

```
    The duration to use in call simulation. (default "1m")  
-version  
    Prints the application version.
```

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 3.1. Using packages

Depending on the packaged distribution, following methods are available:

3.1.1 3.1.1. Debian

This is for the moment the only packaged and the most recommended to use method to install CGRateS. CGRateS development team maintains official debian packages out of master branch, released under nightly tag in aptitude.

There are two main ways of installing the maintained packages:

3.1.1.1 3.1.1.1. Aptitude repository

Add the gpg key:

```
sudo wget -O - http://apt.cgrates.org/apt.cgrates.org.gpg.key | sudo apt-key add -
```

Add the repository in apt sources list:

```
echo "deb http://apt.cgrates.org/debian/ nightly main" | sudo tee /etc/apt/sources.  
↪list.d/cgrates.list
```

Update & install:

```
sudo apt-get update  
sudo apt-get install cgrates
```

Once the installation is completed, one should perform the 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**.

3.1.1.2 3.1.1.2. Manual installation of .deb package out of archive server

Run the following commands:

```
wget http://pkg.cgrates.org/deb/nightly/cgrates_current_amd64.deb
dpkg -i cgrates_current_amd64.deb
```

As a side note on <http://pkg.cgrates.org> one can find an entire archive of CGRateS packages.

3.1.2 3.1.2. Redhat/Fedora/CentOS

There are two main ways of installing the maintained packages:

3.1.2.1 3.1.2.1. YUM repository

To install CGRateS out of YUM execute the following commands

```
sudo tee -a /etc/yum.repos.d/cgrates.repo > /dev/null <<EOT
[cgrates]
name=CGRateS
baseurl=http://yum.cgrates.org/yum/nightly/
enabled=1
gpgcheck=1
gpgkey=http://yum.cgrates.org/yum.cgrates.org.gpg.key
EOT
sudo yum update
sudo yum install cgrates
```

Once the installation is completed, one should perform the 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**.

3.1.2.2 3.1.2.2. Manual installation of .rpm package out of archive server

Run the following commands:

```
sudo rpm -i http://pkg.cgrates.org/rpm/nightly/cgrates_current.rpm
```

As a side note on <http://pkg.cgrates.org> one can find an entire archive of CGRateS packages.

3.2 3.2. Using source

For developing CGRateS and switching between its versions, we are using the **new go mods feature** introduced in go 1.13.

3.2.1 3.2.1 Install GO Lang

First we have to setup the GO Lang to our OS. Feel free to download the latest GO binary release from <https://golang.org/dl/> In this Tutorial we are going to install Go 1.13

```
rm -rf /usr/local/go
cd /tmp
wget https://dl.google.com/go/go1.13.1.linux-amd64.tar.gz
sudo tar -xvf go1.13.1.linux-amd64.tar.gz -C /usr/local/
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
```

3.2.2 3.2.2 Build CGRateS from Source

Configure the project with the following commands:

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

3.2.3 3.2.3 Create Debian / Ubuntu Packages from Source

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

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

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

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

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

3.2.4 3.2.4 Install Custom Debian / Ubuntu Package

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

3.3 3.3. Post-install

3.3.1 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 3.3.2 Set versions data

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

Sample usage for MySQL:

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

CHAPTER 4

Configuration

Has a *JSON* format with commented lines starting with *//*.

Organized into configuration sections which offers the advantage of being easily splittable.

Hint: You can split the configuration into any number of *.json* files/directories since the *:ref:cgr-engine* loads recursively the complete configuration folder, alphabetically ordered

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.

Can be loaded from local folders or remotely using HTTP transport.

The configuration can be loaded at start and reloaded at run time using APIs designed for that. This can be done either as *config pull* (reload from path) or as *config push* (the *JSON BLOB* is sent via API to the engine).

Hint: You can reload from remote HTTP server as well.

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

```
1 {
2
3 // Real-time Online/Offline Charging System (OCS) for Telecom & ISP environments
4 // Copyright (C) ITsysCOM GmbH
5 //
6 // This file contains the default configuration hardcoded into CGRateS.
7 // This is what you get when you load CGRateS with an empty configuration file.
8
9 // "general": {
10 //     "node_id": "",
11 //     ↪
12 //     ↪ identifier of this instance in the cluster, if empty it will be autogenerated
13 //     "logger": "*syslog",
14 //     ↪
15 //     ↪ controls the destination of logs < *syslog | *stdout >
```

(continues on next page)

(continued from previous page)

```

12 //      "log_level": 6,
    ↪
    ↪/ control the level of messages logged (0-emerg to 7-debug)
13 //      "http_skip_tls_verify": false,
    ↪
    ↪                                // if enabled HttpClient_
    ↪will accept any TLS certificate
14 //      "rounding_decimals": 5,
    ↪
    ↪                                // system_
    ↪level precision for floats
15 //      "dbdata_encoding": "*msgpack",
    ↪
    ↪                                // encoding used to store_
    ↪object data in strings: < *msgpack|*json>
16 //      "tpexport_dir": "/var/spool/cgrates/tpexport",
    ↪                                //
    ↪ path towards export folder for offline TariffPlans
17 //      "poster_attempts": 3,
    ↪
    ↪                                // number_
    ↪of attempts before considering post request failed (eg: *http_post, CDR exports)
18 //      "failed_posts_dir": "/var/spool/cgrates/failed_posts",
    ↪                                // directory_
    ↪path where we store failed requests
19 //      "failed_posts_ttl": "5s",
    ↪
    ↪                                // time to wait_
    ↪before writing the failed posts in a single file
20 //      "default_request_type": "*rated",
    ↪
    ↪                                // default request type to consider_
    ↪when missing from requests: <"|*prepaid|*postpaid|*pseudoprepaid|*rated>
21 //      "default_category": "call",
    ↪
    ↪                                // default category_
    ↪to consider when missing from requests
22 //      "default_tenant": "cgrates.org",
    ↪
    ↪                                // default tenant to consider when_
    ↪missing from requests
23 //      "default_timezone": "Local",
    ↪
    ↪                                // default timezone for_
    ↪timestamps where not specified <"|UTC|Local|$IANA_TZ_DB>
24 //      "default_caching": "*reload",
    ↪
    ↪                                // default actions to do_
    ↪when caching items
25 //      "connect_attempts": 5,
    ↪
    ↪                                // initial_
    ↪server connect attempts
26 //      "reconnects": -1,
    ↪
    ↪                                //
    ↪number of retries in case of connection lost
27 //      "connect_timeout": "1s",
    ↪
    ↪                                // consider_
    ↪connection unsuccessful on timeout, 0 to disable the feature
28 //      "reply_timeout": "2s",
    ↪
    ↪                                // consider_
    ↪connection down for replies taking longer than this value
29 //      "locking_timeout": "0",
    ↪
    ↪                                // timeout_
    ↪internal locks to avoid deadlocks
30 //      "digest_separator": ",",
    ↪
    ↪                                // separator to use_
    ↪in replies containing data digests
31 //      "digest_equal": ":",
    ↪
    ↪                                // equal_
    ↪symbol used in case of digests

```

(continues on next page)

(continued from previous page)

```

72 //      "*thresholds": {"remote":false,"replicate":false,"ttl": ""},
73 //      "*filters": {"remote":false,"replicate":false,"ttl": ""},
74 //      "*supplier_profiles":{"remote":false,"replicate":false,"ttl": ""},
75 //      "*attribute_profiles":{"remote":false,"replicate":false,"ttl": ""},
76 //      "*charger_profiles": {"remote":false,"replicate":false,"ttl": ""},
77 //      "*dispatcher_profiles":{"remote":false,"replicate":false,"ttl": ""}
78 //      "*dispatcher_hosts":{"remote":false,"replicate":false,"ttl": ""},
79 //      "*filter_indexes" : {"remote":false,"replicate":false,"ttl": ""},
80 //      "*load_ids":{"remote":false,"replicate":false,"ttl": ""},
81 //    },
82 //  },
83
84
85 // "stor_db": { //
86 //   database used to store offline tariff plans and CDRs
87 //   "db_type": "mysql", // stor_
88 //   database type to use: <*mongo|*mysql|*postgres|*internal>
89 //   "db_host": "127.0.0.1", // the host_
90 //   to connect to
91 //   "db_port": 3306, // the_
92 //   port to reach the stor_db
93 //   "db_name": "cgrates", // stor_
94 //   database name
95 //   "db_user": "cgrates", // username_
96 //   to use when connecting to stor_db
97 //   "db_password": "", //
98 //   password to use when connecting to stor_db
99 //   "max_open_conns": 100, // maximum_
100 //   database connections opened, not applying for mongo
101 //   "max_idle_conns": 10, // maximum_
102 //   database connections idle, not applying for mongo
103 //   "conn_max_lifetime": 0, // maximum amount_
104 //   of time in seconds a connection may be reused (0 for unlimited), not applying for_
105 //   mongo
106 //   "string_indexed_fields": [], // indexes on cdrs_
107 //   table to speed up queries, used in case of *mongo and *internal
108 //   "prefix_indexed_fields": [], // prefix_
109 //   indexes on cdrs table to speed up queries, used in case of *internal
110 //   "query_timeout": "10s",
111 //   "sslmode": "disable", // sslmode in_
112 //   case of *postgres
113 //   "items":{
114 //     "session_costs": {"ttl": ""},
115 //     "cdrs": {"ttl": ""},
116 //     "tp_timings":{"ttl": ""},
117 //     "tp_destinations": {"ttl": ""},
118 //     "tp_rates": {"ttl": ""},
119 //     "tp_destination_rates": {"ttl": ""},
120 //     "tp_rating_plans":{"ttl": ""},
121 //     "tp_rating_profiles":{"ttl": ""},
122 //     "tp_shared_groups": {"ttl": ""},
123 //     "tp_actions":{"ttl": ""},
124 //     "tp_action_plans":{"ttl": ""},
125 //     "tp_action_triggers":{"ttl": ""},
126 //     "tp_account_actions": {"ttl": ""},

```

(continues on next page)

(continued from previous page)

```

113 //         "tp_resources":{"ttl": ""},
114 //         "tp_stats":{"ttl": ""},
115 //         "tp_thresholds": {"ttl": ""},
116 //         "tp_filters": {"ttl": ""},
117 //         "tp_suppliers": {"ttl": ""},
118 //         "tp_attributes":{"ttl": ""},
119 //         "tp_chargers":{"ttl": ""},
120 //         "versions": {"ttl": ""},
121 //         "tp_dispatcher_profiles":{"ttl": ""},
122 //         "tp_dispatcher_hosts":{"ttl": ""},
123 //     },
124 // },
125
126
127 // "listen": {
128 //     "rpc_json": "127.0.0.1:2012",           // RPC JSON listening_
129 ↪address
130 //     "rpc_gob": "127.0.0.1:2013",         // RPC GOB listening_
131 ↪address
132 //     "http": "127.0.0.1:2080",           // HTTP listening_
133 ↪address
134 //     "rpc_json_tls" : "127.0.0.1:2022",   // RPC JSON TLS_
135 ↪listening address
136 //     "rpc_gob_tls": "127.0.0.1:2023",     // RPC GOB TLS listening_
137 ↪address
138 //     "http_tls": "127.0.0.1:2280",       // HTTP TLS listening_
139 ↪address
140 // },
141
142 // "tls": {
143 //     "server_certificate" : "",           // path to server_
144 ↪certificate
145 //     "server_key": "",                   // path to server_
146 ↪key
147 //     "client_certificate" : "",         // path to client_
148 ↪certificate
149 //     "client_key": "",                  // path to client_
150 ↪key
151 //     "ca_certificate": "",              // path to CA_
152 ↪certificate (populate for self-signed certificate otherwise let it empty)
153 //     "server_policy":4,                 // server_policy_
154 ↪determines the TLS Client Authentication (0-NoClientCert, 1-RequestClientCert, 2-
155 ↪RequireAnyClientCert, 3-VerifyClientCertIfGiven, 4-RequireAndVerifyClientCert)
156 //     "server_name": "",
157 // },
158
159 // "http":
160 ↪{
161 ↪HTTP server configuration
162 //     "json_rpc_url": "/jsonrpc",        // JSON_
163 ↪RPC relative URL (" to disable)
164 //     "ws_url": "/ws",                  // _
165 ↪WebSockets relative URL (" to disable)
166 //     "freeswitch cdrs_url": "/freeswitch_json", // Freeswitch CDRS_
167 ↪relative URL (" to disable)

```

(continues on next page)

(continued from previous page)

```

152 //      "http_cdrs": "/cdr_http",                                // CDRS_
↳relative URL (" " to disable)
153 //      "use_basic_auth": false,                                // use_
↳basic authentication
154 //      "auth_users": {},                                        //
↳ basic authentication usernames and base64-encoded passwords (eg: { "username1":
↳"cGFzc3dvcmQ=", "username2": "cGFzc3dvcmQy "})
155 // },
156
157
158 // "schedulers": {
159 //      "enabled": false,                                        // start Scheduler_
↳service: <true/false>
160 //      "cdrs_conns": [],                                       // connections to CDRs_
↳for *cdrlog actions <*internal/x.y.z.y:1234>
161 //      "filters": [],                                           // only execute_
↳actions matching these filters
162 // },
163
164
165 // "caches":{
166 //      "*destinations": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // destination caching
167 //      "*reverse_destinations": {"limit": -1, "ttl": "", "static_ttl": false,
↳"precache": false},                                       // reverse destinations index caching
168 //      "*rating_plans": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // rating plans caching
169 //      "*rating_profiles": {"limit": -1, "ttl": "", "static_ttl": false, "precache
↳": false},                                       // rating profiles caching
170 //      "*actions": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // actions caching
171 //      "*action_plans": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // action plans caching
172 //      "*account_action_plans": {"limit": -1, "ttl": "", "static_ttl": false,
↳"precache": false},                                       // account action plans index caching
173 //      "*action_triggers": {"limit": -1, "ttl": "", "static_ttl": false, "precache
↳": false},                                       // action triggers caching
174 //      "*shared_groups": {"limit": -1, "ttl": "", "static_ttl": false, "precache
↳": false},                                       // shared groups caching
175 //      "*timings": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // timings caching
176 //      "*resource_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳"precache": false},                                       // control resource profiles caching
177 //      "*resources": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // control resources caching
178 //      "*event_resources": {"limit": -1, "ttl": "", "static_ttl": false},
↳
↳                                     // matching resources to_
↳events
179 //      "*statqueue_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳"precache": false},                                       // statqueue profiles
180 //      "*statqueues": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // statqueues with metrics
181 //      "*threshold_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳"precache": false},                                       // control threshold profiles caching
182 //      "*thresholds": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // control thresholds caching
183 //      "*filters": {"limit": -1, "ttl": "", "static_ttl": false, "precache":_
↳false},                                       // control filters caching

```

(continues on next page)

(continued from previous page)

```

184 //      "*supplier_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳ "precache": false},          // control supplier profile caching
185 //      "*attribute_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳ "precache": false},          // control attribute profile caching
186 //      "*charger_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳ "precache": false},          // control charger profile caching
187 //      "*dispatcher_profiles": {"limit": -1, "ttl": "", "static_ttl": false,
↳ "precache": false},          // control dispatcher profile caching
188 //      "*dispatcher_hosts": {"limit": -1, "ttl": "", "static_ttl": false,
↳ "precache": false},          // control dispatcher hosts caching
189 //      "*resource_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
↳                               // control resource filter indexes caching
190 //      "*stat_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},↳
↳                               // control stat filter indexes caching
191 //      "*threshold_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false}
↳ ,                               // control threshold filter indexes caching
192 //      "*supplier_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},
↳                               // control supplier filter indexes caching
193 //      "*attribute_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false}
↳ ,                               // control attribute filter indexes caching
194 //      "*charger_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl": false},↳
↳                               // control charger filter indexes caching
195 //      "*dispatcher_filter_indexes" : {"limit": -1, "ttl": "", "static_ttl":↳
↳ false},                          // control dispatcher filter indexes caching
196 //      "*dispatcher_routes": {"limit": -1, "ttl": "", "static_ttl": false},↳
↳                               // control dispatcher routes caching
197 //      "*diameter_messages": {"limit": -1, "ttl": "3h", "static_ttl": false},
↳                               // diameter messages caching
198 //      "*rpc_responses": {"limit": 0, "ttl": "2s", "static_ttl": false},
↳                               // RPC responses caching
199 //      "*closed_sessions": {"limit": -1, "ttl": "10s", "static_ttl": false},
↳                               // closed sessions cached for CDRs
200 //      "*cdr_ids": {"limit": -1, "ttl": "10m", "static_ttl": false},
↳                               // protects↳
↳ CDRs against double-charging
201 //      "*load_ids": {"limit": -1, "ttl": "", "static_ttl": false, "precache":↳
↳ false},                          // control the load_ids for items
202 //      "*rpc_connections": {"limit": -1, "ttl": "", "static_ttl": false},
↳                               // RPC connections caching
203 // },
204
205 // "filters": {
↳ Filters configuration (*new)
207 //      "stats_conns": [],
↳ connections to StatS for <*stats> filters, empty to disable stats functionality: <"
↳ |*internal/x.y.z.y:1234>
208 //      "resources_conns": [],
↳ connections to ResourceS for <*resources> filters, empty to disable stats↳
↳ functionality: <"|*internal/x.y.z.y:1234>
209 //      "apiers_conns": [],
↳ connections to RALs for <*accounts> filters, empty to disable stats functionality: <
↳ "|*internal/x.y.z.y:1234>
210 // },
211
212 // "rals": {

```

(continues on next page)

(continued from previous page)

```

214 //      "enabled": false,                                     // enable_
↳Rating/Accounting service: <true/false>
215 //      "thresholds_conns": [],                               //
↳connections to ThresholdS for account/balance updates, empty to disable thresholds_
↳functionality: <"|*internal|x.y.z.y:1234>
216 //      "stats_conns": [],                                   //
↳connections to StatS for account/balance updates, empty to disable stats_
↳functionality: <"|*internal|x.y.z.y:1234>
217 //      "caches_conns":["*internal"],                       // connections to_
↳CacheS for account/balance updates
218 //      "rp_subject_prefix_matching": false,                // enables prefix matching for_
↳the rating profile subject
219 //      "remove_expired":true,                              // enables_
↳automatic removal of expired balances
220 //      "max_computed_usage": {                               // do not_
↳compute usage higher than this, prevents memory overload
221 //          "*any": "189h",
222 //          "*voice": "72h",
223 //          "*data": "107374182400",
224 //          "*sms": "10000"
225 //      },
226 //      "max_increments": 1000000,
227 //      "balance_rating_subject":{                            // default_
↳rating subject in case that balance rating subject is empty
228 //          "*any": "*zerolns",
229 //          "*voice": "*zerols",
230 //          "*data": "*zerolns",
231 //          "*sms": "*zerolns",
232 //          "*monetary": "*zerolns",
233 //          "*generic": "*zerolns",
234 //      },
235 // },
236
237 // "cdrs": {                                                 //
↳ CDRs config
238 //      "enabled": false,                                     // start_
↳the CDR Server: <true/false>
239 //      "extra_fields": [],                                  //
↳extra fields to store in CDRs for non-generic CDRs (ie: FreeSWITCH JSON)
240 //      "store_cdrs": true,                                  //
↳store cdrs in StorDB
241 //      "session_cost_retries": 5,                           // number of_
↳queries to session_costs before recalculating CDR
242 //      "chargers_conns": [],                                 // connection_
↳to ChargerS for CDR forking, empty to disable billing for CDRs: <"|*internal|x.y.z.
↳y:1234>
243 //      "rals_conns": [],                                    //
↳ connections to RALs for cost calculation: <"|*internal|x.y.z.y:1234>
244 //      "attributes_conns": [],                              //
↳connection to AttributeS for altering *raw CDRs, empty to disable attributes_
↳functionality: <"|*internal|x.y.z.y:1234>
245 //      "thresholds_conns": [],                              //
↳connection to ThresholdS for CDR reporting, empty to disable thresholds_
↳functionality: <"|*internal|x.y.z.y:1234>
246 //      "stats_conns": [],                                   //
↳connections to StatS for CDR reporting, empty to disable stats functionality: <"
↳"|*internal|x.y.z.y:1234>

```

(continues on next page)

(continued from previous page)

```

248 //      "online_cdr_exports": [],                                // list of CDRE_
↳ profiles to use for real-time CDR exports
249 // },
250
251
252 // "cdre":
↳ {
↳ / CDRE config
253 //      "*default": {
254 //          "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|*sqs_
↳ json_map>
255 //          "export_path": "/var/spool/cgrates/cdre",            // path_
↳ where the exported CDRs will be placed
256 //          "filters" : [],
↳
↳ // filters_
↳ for this export
257 //          "tenant": "",
↳
↳ // tenant_
↳ used in filterS.Pass
258 //          "synchronous": false,
↳
↳ // block processing until_
↳ export has a result
259 //          "attempts": 1,
↳
↳ // export_
↳ attempts
260 //          "field_separator": ",",
↳
↳ // used field separator in_
↳ some export formats, eg: *file_csv
261 //          "attributes_context": "",
↳
↳ // attributes context - empty_
↳ disables attributes processing
262 //          "fields": _
↳
↳ // template of the_
↳ exported content fields
263 //          {"path": "*exp.CGRID", "type": "*composed", "value": "~
↳ *req.CGRID"},
264 //          {"path": "*exp.RunID", "type": "*composed", "value": "~
↳ *req.RunID"},
265 //          {"path": "*exp.ToR", "type": "*composed", "value": "~*req.
↳ ToR"},
266 //          {"path": "*exp.OriginID", "type": "*composed", "value": "~
↳ *req.OriginID"},
267 //          {"path": "*exp.RequestType", "type": "*composed", "value":
↳ "~*req.RequestType"},
268 //          {"path": "*exp.Tenant", "type": "*composed", "value": "~
↳ *req.Tenant"},
269 //          {"path": "*exp.Category", "type": "*composed", "value": "~
↳ *req.Category"},
270 //          {"path": "*exp.Account", "type": "*composed", "value": "~
↳ *req.Account"},
271 //          {"path": "*exp.Subject", "type": "*composed", "value": "~
↳ *req.Subject"},
272 //          {"path": "*exp.Destination", "type": "*composed", "value":
↳ "~*req.Destination"},
273 //          {"path": "*exp.SetupTime", "type": "*composed", "value": "~
↳ *req.SetupTime", "layout": "2006-01-02T15:04:05Z07:00"},

```

(continues on next page)

(continued from previous page)

```

274 // {"path": "*exp.AnswerTime", "type": "*composed", "value":
↳ "~*req.AnswerTime", "layout": "2006-01-02T15:04:05Z07:00"},
275 // {"path": "*exp.Usage", "type": "*composed", "value": "~
↳ *req.Usage"},
276 // {"path": "*exp.Cost", "type": "*composed", "value": "~*req.
↳ Cost", "rounding_decimals": 4},
277 // ],
278 // },
279 // },
280
281 // "ers": { //
282 ↳ EventReaderService // starts
283 // "enabled": false, // RPC Connections
284 ↳ the EventReader service: <true/false> // RPC Connections
285 // "sessions_conns": ["*internal"], // IDs
286 ↳ IDs
287 // "readers": [ //
288 // { //
289 // "id": "*default", //
290 ↳ identifier of the EventReader profile //
291 // "type": "*file_csv", // reader type
292 ↳ <*file_csv> // separator used
293 // "field_separator": ",", // separator used
294 ↳ in case of csv files //
295 // "run_delay": "0", //
296 ↳ sleep interval in seconds between consecutive runs, -1 to use automation via
297 ↳ inotify or 0 to disable running all together //
298 // "concurrent_requests": 1024, // maximum simultaneous requests/
299 ↳ files to process, 0 for unlimited //
300 // "source_path": "/var/spool/cgrates/cdrc/in", //
301 ↳ // read data from this path //
302 // "processed_path": "/var/spool/cgrates/cdrc/out", //
303 ↳ move processed data here // path towards one
304 // "xml_root_path": "", // path towards one
305 ↳ event in case of XML CDRs //
306 // "tenant": "", //
307 ↳ tenant used by import //
308 // "timezone": "", //
309 ↳ timezone for timestamps where not specified <"|UTC|Local|$IANA_TZ_DB> //
310 // "filters": [], //
311 ↳ limit parsing based on the filters //
312 // "flags": [], //
313 ↳ flags to influence the event processing //
314 // "fields //
315 ↳ ":[ //
316 ↳ / import fields template, tag will match internally CDR field, in case of .csv
317 ↳ value will be represented by index of the field value //

```

(continues on next page)

(continued from previous page)

```

300 // {"tag": "ToR", "path": "*cgreg.ToR", "type":
↳"*variable", "value": "~*req.2", "mandatory": true},
301 // {"tag": "OriginID", "path": "*cgreg.OriginID",
↳"type": "*variable", "value": "~*req.3", "mandatory": true},
302 // {"tag": "RequestType", "path": "*cgreg.RequestType
↳", "type": "*variable", "value": "~*req.4", "mandatory": true},
303 // {"tag": "Tenant", "path": "*cgreg.Tenant", "type":
↳"*variable", "value": "~*req.6", "mandatory": true},
304 // {"tag": "Category", "path": "*cgreg.Category",
↳"type": "*variable", "value": "~*req.7", "mandatory": true},
305 // {"tag": "Account", "path": "*cgreg.Account", "type
↳": "*variable", "value": "~*req.8", "mandatory": true},
306 // {"tag": "Subject", "path": "*cgreg.Subject", "type
↳": "*variable", "value": "~*req.9", "mandatory": true},
307 // {"tag": "Destination", "path": "*cgreg.Destination
↳", "type": "*variable", "value": "~*req.10", "mandatory": true},
308 // {"tag": "SetupTime", "path": "*cgreg.SetupTime",
↳"type": "*variable", "value": "~*req.11", "mandatory": true},
309 // {"tag": "AnswerTime", "path": "*cgreg.AnswerTime",
↳"type": "*variable", "value": "~*req.12", "mandatory": true},
310 // {"tag": "Usage", "path": "*cgreg.Usage", "type":
↳"*variable", "value": "~*req.13", "mandatory": true},
311 // ],
312 // "cache_dump_fields": [],
313 // },
314 // ],
315 // },
316
317
318 // "sessions": {
319 // "enabled": false, // starts_
↳the session service: <true/false>
320 // "listen_bijson": "127.0.0.1:2014", // address where to_
↳listen for bidirectional JSON-RPC requests
321 // "chargers_conns": [], //_
↳connections to ChargerS for session forking <*internal|x.y.z.y:1234>
322 // "rals_conns": [], //_
↳connections to RALs for rating/accounting <"|*internal|127.0.0.1:2013>
323 // "cdrs_conns": [], //_
↳connections to CDRs for CDR posting <*internal|x.y.z.y:1234>
324 // "resources_conns": [], //_
↳connections to ResourceS for resources monitoring <"|*internal|127.0.0.1:2013>
325 // "thresholds_conns": [], //_
↳connections to ThresholdS for reporting session events <"|*internal|127.0.0.1:2013>
326 // "stats_conns": [], //_
↳connections to StatS for reporting session events <"|*internal|127.0.0.1:2013>
327 // "suppliers_conns": [], //_
↳connections to SupplierS for querying suppliers for event <"|*internal|127.0.0.
↳1:2013>
328 // "attributes_conns": [], //_
↳connections to AttributeS for altering event fields <"|*internal|127.0.0.1:2013>
329 // "replication_conns": [], // replicate_
↳sessions towards these session services
330 // "debit_interval": "0s", // interval_
↳to perform debits on.
331 // "store_session_costs": false, // enable storing of_
↳the session costs within CDRs

```

(continues on next page)

(continued from previous page)

```

332 //      "min_call_duration": "0s", // only
    ↳authorize calls with allowed duration higher than this
333 //      "max_call_duration": "3h", // maximum call
    ↳duration a prepaid call can last
334 //      "session_ttl": "0s", // time after
    ↳a session with no updates is terminated, not defined by default
335 //      //"session_ttl_max_delay": "", // activates session_
    ↳ttl randomization and limits the maximum possible delay
336 //      //"session_ttl_last_used": "", // tweak LastUsed
    ↳for sessions timing-out, not defined by default
337 //      //"session_ttl_usage": "", // tweak Usage
    ↳for sessions timing-out, not defined by default
338 //      "session_indexes": [], // index
    ↳sessions based on these fields for GetActiveSessions API
339 //      "client_protocol": 1.0, // version
    ↳of protocol to use when acting as JSON-PRC client <"0","1.0">
340 //      "channel_sync_interval": "0", // sync channels to
    ↳detect stale sessions (0 to disable)
341 //      "terminate_attempts": 5, // attempts to get
    ↳the session before terminating it
342 //      "alterable_fields": [], // the
    ↳session fields that can be updated
343 // },
344
345
346 // "asterisk_agent": {
347 //      "enabled": false, // starts
    ↳the Asterisk agent: <true/false>
348 //      "sessions_conns": ["*internal"],
349 //      "create_cdr": false, // create CDR
    ↳out of events and sends it to CDRS component
350 //      "asterisk_conns": [ //
    ↳instantiate connections to multiple Asterisk servers
351 //          {"address": "127.0.0.1:8088", "user": "cgrates", "password":
    ↳"CGRateS.org", "connect_attempts": 3, "reconnects": 5}
352 //      ],
353 // },
354
355
356 // "freeswitch_agent": {
357 //      "enabled": false, // starts
    ↳the FreeSWITCH agent: <true/false>
358 //      "sessions_conns": ["*internal"],
359 //      "subscribe_park": true, //
    ↳subscribe via fsock to receive park events
360 //      "create_cdr": false, // creates CDR
    ↳out of events and sends them to CDRS component
361 //      "extra_fields": [], //
    ↳extra fields to store in auth/CDRs when creating them
362 //      //"min_dur_low_balance": "5s", // threshold which
    ↳will trigger low balance warnings for prepaid calls (needs to be lower than debit_
    ↳interval)
363 //      //"low_balance_ann_file": "", // file to be played
    ↳when low balance is reached for prepaid calls
364 //      "empty_balance_context": "", // if defined, prepaid
    ↳calls will be transferred to this context on empty balance
365 //      "empty_balance_ann_file": "", // file to be played
    ↳before disconnecting prepaid calls on empty balance (applies only if no context
    ↳defined)

```

(continues on next page)

(continued from previous page)

```

366 //      "max_wait_connection": "2s",                // maximum duration to
↳wait for a connection to be retrieved from the pool
367 //      "event_socket_conns":[                        //
↳instantiate connections to multiple FreeSWITCH servers
368 //          {"address": "127.0.0.1:8021", "password": "ClueCon", "reconnects":
↳5, "alias":""}
369 //      ],
370 // },
371
372 // "kamailio_agent": {
373 //      "enabled": false,                            // starts
↳Kamailio agent: <true/false>
374 //      "sessions_conns": ["*internal"],
375 //      "create_cdr": false,                        // create CDR
↳out of events and sends them to CDRS component
376 //      "timezone": "",                             //
↳timezone of the Kamailio server
377 //      "evapi_conns":[                             //
↳instantiate connections to multiple Kamailio servers
378 //          {"address": "127.0.0.1:8448", "reconnects": 5}
379 //      ],
380 // },
381
382 // "diameter_agent": {
383 //      "enabled": false,
384 //      "listen": "127.0.0.1:3868",
385 //      "listen_net": "tcp",
386 //      "listen_net": "tcp",                        // address
↳where to listen for diameter requests <x.y.z.y/x1.y1.z1.y1:1234>
387 //      "listen_net": "tcp",                       //
↳transport type for diameter <tcp|sctp>
388 //      "dictionaries_path": "/usr/share/cgrates/diameter/dict/", // path
↳towards directory holding additional dictionaries to load
389 //      "sessions_conns": ["*internal"],
390 //      "origin_host": "CGR-DA",                    // diameter
↳Origin-Host AVP used in replies
391 //      "origin_realm": "cgrates.org",              // diameter Origin-
↳Realm AVP used in replies
392 //      "vendor_id": 0,
393 //      "product_name": "CGRateS",                 // diameter
↳Product-Name AVP used in replies
394 //      "concurrent_requests": -1,                  // limit
↳the number of active requests processed by the server <-1|0-n>
395 //      "syncd_conn_requests": false,              // process one
↳request at the time per connection
396 //      "asr_template": "",

```

(continues on next page)

```

↳enable AbortSession message being sent to client on DisconnectSession

```

(continued from previous page)

```

397 //      "templates":
↪ {
↪ / default message templates
398 //      "err": [
399 //          {"tag": "SessionId", "path": "*rep.Session-Id",
↪ "type": "*variable",
400 //          "value": "~*req.Session-Id", "mandatory":
↪ true},
401 //          {"tag": "OriginHost", "path": "*rep.Origin-Host",
↪ "type": "*variable",
402 //          "value": "~*vars.OriginHost", "mandatory":
↪ true},
403 //          {"tag": "OriginRealm", "path": "*rep.Origin-Realm",
↪ "type": "*variable",
404 //          "value": "~*vars.OriginRealm", "mandatory
↪ ": true},
405 //      ],
406 //      "cca": [
407 //          {"tag": "SessionId", "path": "*rep.Session-Id",
↪ "type": "*variable",
408 //          "value": "~*req.Session-Id", "mandatory":
↪ true},
409 //          {"tag": "ResultCode", "path": "*rep.Result-Code",
↪ "type": "*constant",
410 //          "value": "2001"},
411 //          {"tag": "OriginHost", "path": "*rep.Origin-Host",
↪ "type": "*variable",
412 //          "value": "~*vars.OriginHost", "mandatory":
↪ true},
413 //          {"tag": "OriginRealm", "path": "*rep.Origin-Realm",
↪ "type": "*variable",
414 //          "value": "~*vars.OriginRealm", "mandatory
↪ ": true},
415 //          {"tag": "AuthApplicationId", "path": "*rep.Auth-
↪ Application-Id", "type": "*variable",
416 //          "value": "~*vars.*appid", "mandatory":
↪ true},
417 //          {"tag": "CCRequestType", "path": "*rep.CC-Request-
↪ Type", "type": "*variable",
418 //          "value": "~*req.CC-Request-Type",
↪ "mandatory": true},
419 //          {"tag": "CCRequestNumber", "path": "*rep.CC-
↪ Request-Number", "type": "*variable",
420 //          "value": "~*req.CC-Request-Number",
↪ "mandatory": true},
421 //      ],
422 //      "asr": [
423 //          {"tag": "SessionId", "path": "*diamreq.Session-Id",
↪ "type": "*variable",
424 //          "value": "~*req.Session-Id", "mandatory":
↪ true},
425 //          {"tag": "OriginHost", "path": "*diamreq.Origin-Host
↪ ", "type": "*variable",
426 //          "value": "~*req.Destination-Host",
↪ "mandatory": true},
427 //          {"tag": "OriginRealm", "path": "*diamreq.Origin-
↪ Realm", "type": "*variable",

```

(continues on next page)

(continued from previous page)

```

428 //           "value": "~*req.Destination-Realm",
↳ "mandatory": true},
429 //           {"tag": "DestinationRealm", "path": "*diamreq.
↳ Destination-Realm", "type": "*variable",
430 //           "value": "~*req.Origin-Realm", "mandatory
↳ ": true},
431 //           {"tag": "DestinationHost", "path": "*diamreq.
↳ Destination-Host", "type": "*variable",
432 //           "value": "~*req.Origin-Host", "mandatory":
↳ true},
433 //           {"tag": "AuthApplicationId", "path": "*diamreq.
↳ Auth-Application-Id", "type": "*variable",
434 //           "value": "~*vars.*appid", "mandatory":
↳ true},
435 //           {"tag": "UserName", "path": "*diamreq.User-Name",
↳ "type": "*variable",
436 //           "value": "~*req.User-Name", "mandatory":
↳ true},
437 //           {"tag": "OriginStateID", "path": "*diamreq.Origin-
↳ State-Id", "type": "*constant",
438 //           "value": "1"},
439 //       ]
440 //   },
441 //   "request_processors": [           // list of
↳ processors to be applied to diameter messages
442 //   ],
443 // },
444
445 // "radius_agent": {
446 //     "enabled": false,
↳
447 //     / enables the radius agent: <true/false>
448 //     "listen_net": "udp",           //
↳
↳ network to listen on <udp/tcp>
449 //     "listen_auth": "127.0.0.1:1812",           // address where to listen
↳
↳ for radius authentication requests <x.y.z.y:1234>
450 //     "listen_acct": "127.0.0.1:1813",           // address where to listen
↳
↳ for radius accounting requests <x.y.z.y:1234>
451 //     "client_secrets":
↳
↳ {
↳ / hash containing secrets for clients connecting here <*default|$client_ip>
452 //         "*default": "CGRateS.org"
453 //     },
454 //     "client_dictionaries":
↳
↳ {           // per
↳ client path towards directory holding additional dictionaries to load (extra to RFC)
455 //         "*default": "/usr/share/cgrates/radius/dict/",
↳
↳ // key represents the client IP or catch-all <*default|
↳ $client_ip>
456 //     },
457 //     "sessions_conns": ["*internal"],
458 //     "request_processors":
↳
↳ [           //
↳ request processors to be applied to Radius messages

```

(continues on next page)

(continued from previous page)

```

459 //     ],
460 // },
461
462 // "http_agent": [                                // HTTP Agents, ie_
463   ↪towards cnc.to MVNE platform
464 // ],
465
466 // "dns_agent": {
467 //     "enabled": false,
468   ↪
469   ↪/ enables the DNS agent: <true/false>
470 //     "listen": "127.0.0.1:2053",                // address_
471   ↪where to listen for DNS requests <x.y.z.y:1234>
472 //     "listen_net": "udp",                        //_
473   ↪network to listen on <udp/tcp/tcp-tls>
474 //     "sessions_conns": ["*internal"],
475 //     "timezone": "",
476   ↪
477   ↪/ timezone of the events if not specified <UTC/Local/$IANA_TZ_DB>
478 //     "request_processors":_
479   ↪[                                                //_
480   ↪request processors to be applied to DNS messages
481 //     ],
482 // },
483
484 // "attributes": {                                //_
485   ↪Attributes config
486 //     "enabled": false,                            // starts_
487   ↪attribute service: <true/false>.
488 //     "indexed_selects":true,                       // enable_
489   ↪profile matching exclusively on indexes
490 //     //"string_indexed_fields": [],                 // query indexes_
491   ↪based on these fields for faster processing
492 //     "prefix_indexed_fields": [],                 // query indexes based_
493   ↪on these fields for faster processing
494 //     "nested_fields": false,                       //_
495   ↪determines which field is checked when matching indexed filters(true: all; false:_
496   ↪only the one on the first level)
497 //     "process_runs": 1,                            //_
498   ↪number of run loops when processing event
499 // },
500
501 // "chargers": {                                  //_
502   ↪ChargerS config
503 //     "enabled": false,                            // starts_
504   ↪charger service: <true/false>.
505 //     "attributes_conns": [],                       //_
506   ↪connections to AttributeS for event fields altering <"|127.0.0.1:2013>
507 //     "indexed_selects":true,                       // enable_
508   ↪profile matching exclusively on indexes
509 //     //"string_indexed_fields": [],                 // query indexes_
510   ↪based on these fields for faster processing

```

(continues on next page)

(continued from previous page)

```

493 //      "prefix_indexed_fields": [],                // query indexes based
↳on these fields for faster processing
494 //      "nested_fields": false,                    //
↳determines which field is checked when matching indexed filters(true: all; false:
↳only the one on the first level)
495 // },
496
497
498 // "resources": {                                  //
↳ResourceS config
499 //      "enabled": false,                          // starts
↳ResourceLimiter service: <true|false>.
500 //      "store_interval": "",                      // dump cache
↳regularly to dataDB, 0 - dump at start/shutdown: <"|$dur>
501 //      "thresholds_conns": [],                    //
↳connections to ThresholdS for resource reporting, empty to disable thresholds
↳functionality: <"|*internal|x.y.z.y:1234>
502 //      "indexed_selects":true,                   // enable
↳profile matching exclusively on indexes
503 //      //"string_indexed_fields": [],              // query indexes
↳based on these fields for faster processing
504 //      "prefix_indexed_fields": [],              // query indexes based
↳on these fields for faster processing
505 //      "nested_fields": false,                   //
↳determines which field is checked when matching indexed filters(true: all; false:
↳only the one on the first level)
506 // },
507
508
509 // "stats": {                                      /
↳/ StatS config
510 //      "enabled": false,                          // starts
↳Stat service: <true|false>.
511 //      "store_interval": "",                      // dump cache
↳regularly to dataDB, 0 - dump at start/shutdown: <"|$dur>
512 //      "store_uncompressed_limit": 0,            //
↳used to compress data
513 //      "thresholds_conns": [],                    //
↳connections to ThresholdS for StatUpdates, empty to disable thresholds
↳functionality: <"|*internal|x.y.z.y:1234>
514 //      "indexed_selects":true,                   // enable
↳profile matching exclusively on indexes
515 //      //"string_indexed_fields": [],              // query indexes
↳based on these fields for faster processing
516 //      "prefix_indexed_fields": [],              // query indexes based
↳on these fields for faster processing
517 //      "nested_fields": false,                   //
↳determines which field is checked when matching indexed filters(true: all; false:
↳only the one on the first level)
518 // },
519
520
521 // "thresholds": {                                  //
↳ThresholdS
522 //      "enabled": false,                          // starts
↳ThresholdS service: <true|false>.
523 //      "store_interval": "",                      // dump cache
↳regularly to dataDB, 0 - dump at start/shutdown: <"|$dur>

```

(continues on next page)

(continued from previous page)

```

524 //      "indexed_selects":true,                                // enable_
    ↪profile matching exclusively on indexes
525 //      //"string_indexed_fields": [],                          // query indexes_
    ↪based on these fields for faster processing
526 //      "prefix_indexed_fields": [],                          // query indexes based_
    ↪on these fields for faster processing
527 //      "nested_fields": false,                               //_
    ↪determines which field is checked when matching indexed filters(true: all; false:_
    ↪only the one on the first level)
528 // },
529
530
531 // "suppliers": {                                           //_
    ↪SupplierS config
532 //      "enabled": false,                                     // starts_
    ↪SupplierS service: <true|false>.
533 //      "indexed_selects":true,                              // enable_
    ↪profile matching exclusively on indexes
534 //      //"string_indexed_fields": [],                          // query indexes_
    ↪based on these fields for faster processing
535 //      "prefix_indexed_fields": [],                          // query indexes based_
    ↪on these fields for faster processing
536 //      "nested_fields": false,                               //_
    ↪determines which field is checked when matching indexed filters(true: all; false:_
    ↪only the one on the first level)
537 //      "attributes_conns": [],                               //_
    ↪connections to AttributeS for altering events before supplier queries: <"
    ↪|*internal|127.0.0.1:2013>
538 //      "resources_conns": [],                               //_
    ↪connections to ResourceS for *res sorting, empty to disable functionality: <"
    ↪|*internal|x.y.z.y:1234>
539 //      "stats_conns": [],                                   //_
    ↪connections to StatS for *stats sorting, empty to disable stats functionality: <"
    ↪|*internal|x.y.z.y:1234>
540 //      "rals_conns": [],                                    //_
    ↪connections to Rater for calculating cost, empty to disable stats functionality: <"
    ↪|*internal|x.y.z.y:1234>
541 //      "default_ratio":1                                    //_
    ↪default ratio used in case of *load strategy
542 // },
543
544
545 // "loaders":_
    ↪[
    ↪/ LoaderS config
546 //      {
547 //          "id": "*default",
    ↪
    ↪
    ↪identifier of the Loader
548 //          "enabled": false,
    ↪
    ↪
    ↪as service: <true|false>.
549 //          "tenant": "",
    ↪
    ↪
    ↪tenant used in filterS.Pass
550 //          "dry_run": false,
    ↪
    ↪
    ↪do not_
    ↪send the CDRs to CDRS, just parse them

```

(continues on next page)

(continued from previous page)

```

551 //          "run_delay": 0,
↪                                     //
↪sleep interval in seconds between consecutive runs, 0 to use automation via inotify
552 //          "lock_filename": ".cgr.lck",
↪                                     // Filename containing concurrency
↪lock in case of delayed processing
553 //          "caches_conns": ["*internal"],
554 //          "field_separator": ",",
↪                                     // separator used
↪in case of csv files
555 //          "tp_in_dir": "/var/spool/cgrates/loader/in",
↪absolute path towards the directory where the TPs are stored
556 //          "tp_out_dir": "/var/spool/cgrates/loader/out",
↪absolute path towards the directory where processed TPs will be moved
557 //          "data
↪": [
↪/ data profiles to load
558 //          {
559 //              "type": "*attributes",
↪                                     // data source type
560 //              "file_name": "Attributes.csv",
↪                                     // file name in the tp_in_dir
561 //              "fields": [
562 //                  {"tag": "TenantID", "path": "Tenant", "type":
↪": "*variable", "value": "~0", "mandatory": true},
563 //                  {"tag": "ProfileID", "path": "ID", "type":
↪ "*variable", "value": "~1", "mandatory": true},
564 //                  {"tag": "Contexts", "path": "Contexts",
↪ "type": "*variable", "value": "~2"},
565 //                  {"tag": "FilterIDs", "path": "FilterIDs",
↪ "type": "*variable", "value": "~3"},
566 //                  {"tag": "ActivationInterval", "path":
↪ "ActivationInterval", "type": "*variable", "value": "~4"},
567 //                  {"tag": "AttributeFilterIDs", "path":
↪ "AttributeFilterIDs", "type": "*variable", "value": "~5"},
568 //                  {"tag": "Path", "path": "Path", "type":
↪ "*variable", "value": "~6"},
569 //                  {"tag": "Type", "path": "Type", "type":
↪ "*variable", "value": "~7"},
570 //                  {"tag": "Value", "path": "Value", "type":
↪ "*variable", "value": "~8"},
571 //                  {"tag": "Blocker", "path": "Blocker", "type":
↪": "*variable", "value": "~9"},
572 //                  {"tag": "Weight", "path": "Weight", "type":
↪": "*variable", "value": "~10"},
573 //              ],
574 //          },
575 //          {
576 //              "type": "*filters",
↪                                     // data source type
577 //              "file_name": "Filters.csv",
↪                                     // file name in the tp_in_dir
578 //              "fields": [
579 //                  {"tag": "Tenant", "path": "Tenant", "type":
↪": "*variable", "value": "~0", "mandatory": true},
580 //                  {"tag": "ID", "path": "ID", "type":
↪ "*variable", "value": "~1", "mandatory": true},

```

(continues on next page)

(continued from previous page)

```

581 // {"tag": "Type", "path": "Type", "type":
↪ "*variable", "value": "~2"},
582 // {"tag": "Element", "path": "Element", "type
↪ ": "*variable", "value": "~3"},
583 // {"tag": "Values", "path": "Values", "type
↪ ": "*variable", "value": "~4"},
584 // {"tag": "ActivationInterval", "path":
↪ "ActivationInterval", "type": "*variable", "value": "~5"},
585 // },
586 // },
587 // {
588 // "type": "*resources",
↪ // data source type
589 // "file_name": "Resources.csv",
↪ // file name in the tp_in_dir
590 // "fields": [
591 // {"tag": "Tenant", "path": "Tenant", "type
↪ ": "*variable", "value": "~0", "mandatory": true},
592 // {"tag": "ID", "path": "ID", "type":
↪ "*variable", "value": "~1", "mandatory": true},
593 // {"tag": "FilterIDs", "path": "FilterIDs",
↪ "type": "*variable", "value": "~2"},
594 // {"tag": "ActivationInterval", "path":
↪ "ActivationInterval", "type": "*variable", "value": "~3"},
595 // {"tag": "TTL", "path": "UsageTTL", "type":
↪ "*variable", "value": "~4"},
596 // {"tag": "Limit", "path": "Limit", "type":
↪ "*variable", "value": "~5"},
597 // {"tag": "AllocationMessage", "path":
↪ "AllocationMessage", "type": "*variable", "value": "~6"},
598 // {"tag": "Blocker", "path": "Blocker", "type
↪ ": "*variable", "value": "~7"},
599 // {"tag": "Stored", "path": "Stored", "type
↪ ": "*variable", "value": "~8"},
600 // {"tag": "Weight", "path": "Weight", "type
↪ ": "*variable", "value": "~9"},
601 // {"tag": "ThresholdIDs", "path":
↪ "ThresholdIDs", "type": "*variable", "value": "~10"},
602 // },
603 // },
604 // {
605 // "type": "*stats",
↪ // data source type
606 // "file_name": "Stats.csv",
↪ // file name in the tp_in_dir
607 // "fields": [
608 // {"tag": "Tenant", "path": "Tenant", "type
↪ ": "*variable", "value": "~0", "mandatory": true},
609 // {"tag": "ID", "path": "ID", "type":
↪ "*variable", "value": "~1", "mandatory": true},
610 // {"tag": "FilterIDs", "path": "FilterIDs",
↪ "type": "*variable", "value": "~2"},
611 // {"tag": "ActivationInterval", "path":
↪ "ActivationInterval", "type": "*variable", "value": "~3"},
612 // {"tag": "QueueLength", "path": "QueueLength
↪ ", "type": "*variable", "value": "~4"},
613 // {"tag": "TTL", "path": "TTL", "type":
↪ "*variable", "value": "~5"},

```

(continues on next page)

(continued from previous page)

```

614 // {"tag": "MinItems", "path": "MinItems",
↪ "type": "*variable", "value": "~6"},
615 // {"tag": "MetricIDs", "path": "MetricIDs",
↪ "type": "*variable", "value": "~7"},
616 // {"tag": "MetricFilterIDs", "path":
↪ "MetricFilterIDs", "type": "*variable", "value": "~8"},
617 // {"tag": "Blocker", "path": "Blocker", "type
↪ ": "*variable", "value": "~9"},
618 // {"tag": "Stored", "path": "Stored", "type
↪ ": "*variable", "value": "~10"},
619 // {"tag": "Weight", "path": "Weight", "type
↪ ": "*variable", "value": "~11"},
620 // {"tag": "ThresholdIDs", "path":
↪ "ThresholdIDs", "type": "*variable", "value": "~12"},
621 // },
622 // },
623 // {
624 // "type": "*thresholds",
↪ // data source type
625 // "file_name": "Thresholds.csv",
↪ // file name in the tp_in_dir
626 // "fields": [
627 // {"tag": "Tenant", "path": "Tenant", "type
↪ ": "*variable", "value": "~0", "mandatory": true},
628 // {"tag": "ID", "path": "ID", "type":
↪ "*variable", "value": "~1", "mandatory": true},
629 // {"tag": "FilterIDs", "path": "FilterIDs",
↪ "type": "*variable", "value": "~2"},
630 // {"tag": "ActivationInterval", "path":
↪ "ActivationInterval", "type": "*variable", "value": "~3"},
631 // {"tag": "MaxHits", "path": "MaxHits", "type
↪ ": "*variable", "value": "~4"},
632 // {"tag": "MinHits", "path": "MinHits", "type
↪ ": "*variable", "value": "~5"},
633 // {"tag": "MinSleep", "path": "MinSleep",
↪ "type": "*variable", "value": "~6"},
634 // {"tag": "Blocker", "path": "Blocker", "type
↪ ": "*variable", "value": "~7"},
635 // {"tag": "Weight", "path": "Weight", "type
↪ ": "*variable", "value": "~8"},
636 // {"tag": "ActionIDs", "path": "ActionIDs",
↪ "type": "*variable", "value": "~9"},
637 // {"tag": "Async", "path": "Async", "type":
↪ "*variable", "value": "~10"},
638 // },
639 // },
640 // {
641 // "type": "*suppliers",
↪ // data source type
642 // "file_name": "Suppliers.csv",
↪ // file name in the tp_in_dir
643 // "fields": [
644 // {"tag": "Tenant", "path": "Tenant", "type
↪ ": "*variable", "value": "~0", "mandatory": true},
645 // {"tag": "ID", "path": "ID", "type":
↪ "*variable", "value": "~1", "mandatory": true},
646 // {"tag": "FilterIDs", "path": "FilterIDs",
↪ "type": "*variable", "value": "~2"},

```

(continues on next page)

(continued from previous page)

```

647 // {"tag": "ActivationInterval", "path":
↳ "ActivationInterval", "type": "*variable", "value": "~3"},
648 // {"tag": "Sorting", "path": "Sorting", "type
↳ ": "*variable", "value": "~4"},
649 // {"tag": "SortingParamameters", "path":
↳ "SortingParamameters", "type": "*variable", "value": "~5"},
650 // {"tag": "SupplierID", "path": "SupplierID",
↳ "type": "*variable", "value": "~6"},
651 // {"tag": "SupplierFilterIDs", "path":
↳ "SupplierFilterIDs", "type": "*variable", "value": "~7"},
652 // {"tag": "SupplierAccountIDs", "path":
↳ "SupplierAccountIDs", "type": "*variable", "value": "~8"},
653 // {"tag": "SupplierRatingPlanIDs", "path":
↳ "SupplierRatingPlanIDs", "type": "*variable", "value": "~9"},
654 // {"tag": "SupplierResourceIDs", "path":
↳ "SupplierResourceIDs", "type": "*variable", "value": "~10"},
655 // {"tag": "SupplierStatIDs", "path":
↳ "SupplierStatIDs", "type": "*variable", "value": "~11"},
656 // {"tag": "SupplierWeight", "path":
↳ "SupplierWeight", "type": "*variable", "value": "~12"},
657 // {"tag": "SupplierBlocker", "path":
↳ "SupplierBlocker", "type": "*variable", "value": "~13"},
658 // {"tag": "SupplierParameters", "path":
↳ "SupplierParameters", "type": "*variable", "value": "~14"},
659 // {"tag": "Weight", "path": "Weight", "type
↳ ": "*variable", "value": "~15"},
660 // },
661 // },
662 // {
663 // "type": "*chargers",
↳ // data source type
664 // "file_name": "Chargers.csv",
↳ // file name in the tp_in_dir
665 // "fields": [
666 // {"tag": "Tenant", "path": "Tenant", "type
↳ ": "*variable", "value": "~0", "mandatory": true},
667 // {"tag": "ID", "path": "ID", "type":
↳ "*variable", "value": "~1", "mandatory": true},
668 // {"tag": "FilterIDs", "path": "FilterIDs",
↳ "type": "*variable", "value": "~2"},
669 // {"tag": "ActivationInterval", "path":
↳ "ActivationInterval", "type": "*variable", "value": "~3"},
670 // {"tag": "RunID", "path": "RunID", "type":
↳ "*variable", "value": "~4"},
671 // {"tag": "AttributeIDs", "path":
↳ "AttributeIDs", "type": "*variable", "value": "~5"},
672 // {"tag": "Weight", "path": "Weight", "type
↳ ": "*variable", "value": "~6"},
673 // },
674 // },
675 // {
676 // "type": "*dispatchers",
↳ // data source type
677 // "file_name": "DispatcherProfiles.csv",
↳ // file name in the tp_in_dir
678 // "fields": [
679 // {"tag": "Tenant", "path": "Tenant", "type
↳ ": "*variable", "value": "~0", "mandatory": true},

```

(continues on next page)

(continued from previous page)

```

680 // {"tag": "ID", "path": "ID", "type":
↳"*variable", "value": "~1", "mandatory": true},
681 // {"tag": "Contexts", "path": "Contexts",
↳"type": "*variable", "value": "~2"},
682 // {"tag": "FilterIDs", "path": "FilterIDs",
↳"type": "*variable", "value": "~3"},
683 // {"tag": "ActivationInterval", "path":
↳"ActivationInterval", "type": "*variable", "value": "~4"},
684 // {"tag": "Strategy", "path": "Strategy",
↳"type": "*variable", "value": "~5"},
685 // {"tag": "StrategyParameters", "path":
↳"StrategyParameters", "type": "*variable", "value": "~6"},
686 // {"tag": "ConnID", "path": "ConnID", "type
↳": "*variable", "value": "~7"},
687 // {"tag": "ConnFilterIDs", "path":
↳"ConnFilterIDs", "type": "*variable", "value": "~8"},
688 // {"tag": "ConnWeight", "path": "ConnWeight",
↳ "type": "*variable", "value": "~9"},
689 // {"tag": "ConnBlocker", "path": "ConnBlocker
↳", "type": "*variable", "value": "~10"},
690 // {"tag": "ConnParameters", "path":
↳"ConnParameters", "type": "*variable", "value": "~11"},
691 // {"tag": "Weight", "path": "Weight", "type
↳": "*variable", "value": "~12"},
692 // },
693 // },
694 // {
695 // "type": "*dispatcher_hosts",
↳ // data source type
696 // "file_name": "DispatcherHosts.csv",
↳ // file name in the tp_in_dir
697 // "fields": [
698 // {"tag": "Tenant", "path": "Tenant", "type
↳": "*variable", "value": "~0", "mandatory": true},
699 // {"tag": "ID", "path": "ID", "type":
↳"*variable", "value": "~1", "mandatory": true},
700 // {"tag": "Address", "path": "Address", "type
↳": "*variable", "value": "~2"},
701 // {"tag": "Transport", "path": "Transport",
↳"type": "*variable", "value": "~3"},
702 // {"tag": "TLS", "path": "TLS", "type":
↳"*variable", "value": "~4"},
703 // ],
704 // },
705 // ],
706 // },
707 // ],
708
709
710 // "mailer": {
711 // "server": "localhost",
↳ // the server to
↳use when sending emails out
712 // "auth_user": "cgrates",
↳ // authenticate to
↳email server using this user
713 // "auth_password": "CGRateS.org",
↳ // authenticate to email server
↳with this password

```

(continues on next page)

(continued from previous page)

```

714 //      "from_address": "cgr-mailer@localhost.localdomain"      // from address_
↪used when sending emails out
715 // },
716
717
718 // "suretax": {
719 //      "url": "",
↪/ API url
720 //      "client_number": "",
↪number, provided by SureTax
721 //      "validation_key": "",
↪key provided by SureTax
722 //      "business_unit": "",
↪Business Unit
723 //      "timezone": "Local",
↪time of the events to this timezone before sending request out <UTC|Local|$IANA_TZ_
↪DB>
724 //      "include_local_cost": false,
↪calculated cost with tax one in final cost
725 //      "return_file_code": "0",
↪Quote purposes <0|Q>
726 //      "response_group": "03",
↪determines how taxes are grouped for the response <03|13>
727 //      "response_type": "D4",
↪determines the granularity of taxes and (optionally) the decimal precision for the_
↪tax calculations and amounts in the response
728 //      "regulatory_code": "03",
↪provider type
729 //      "client_tracking": "~*req.CGRID",
↪client information out of StoredCdr; <RSRParsers>
730 //      "customer_number": "~*req.Subject",
↪customer number out of StoredCdr; <RSRParsers>
731 //      "orig_number": "~*req.Subject",
↪origination number out of StoredCdr; <RSRParsers>
732 //      "term_number": "~*req.Destination",
↪termination number out of StoredCdr; <RSRParsers>
733 //      "bill_to_number": "",
↪extracting billed to number out of StoredCdr; <RSRParsers>
734 //      "zipcode": "",
↪template extracting billing zip code out of StoredCdr; <RSRParsers>
735 //      "plus4": "",
↪template extracting billing zip code extension out of StoredCdr; <RSRParsers>
736 //      "p2pzipcode": "",
↪template extracting secondary zip code out of StoredCdr; <RSRParsers>
737 //      "p2pplus4": "",
↪template extracting secondary zip code extension out of StoredCdr; <RSRParsers>
738 //      "units": "1",
↪template extracting number of "lines" or unique charges contained within the_
↪revenue out of StoredCdr; <RSRParsers>
739 //      "unit_type": "00",
↪template extracting number of unique access lines out of StoredCdr; <RSRParsers>
740 //      "tax_included": "0",
↪extracting tax included in revenue out of StoredCdr; <RSRParsers>
741 //      "tax_situs_rule": "04",
↪extracting tax situs rule out of StoredCdr; <RSRParsers>
742 //      "trans_type_code": "010101",
↪transaction type indicator out of StoredCdr; <RSRParsers>
743 //      "sales_type_code": "R",
↪extracting sales type code out of StoredCdr; <RSRParsers>

```

(continues on next page)

(continued from previous page)

```

744 //      "tax_exemption_code_list": "",                // template_
    ↪extracting tax exemption code list out of StoredCdr; <RSRParsers>
745 // },
746
747
748 // "loader":
    ↪{                                                    // loader_
    ↪for tariff plans out of .csv files
749 //      "tpid": "",
    ↪                                                    // tariff plan_
    ↪identifier
750 //      "data_path": "./",                            // path_
    ↪towards tariff plan files
751 //      "disable_reverse": false,                    // disable_
    ↪reverse computing
752 //      "field_separator": ",",                      //
    ↪separator used in case of csv files
753 //      "caches_conns":["*localhost"],
754 //      "scheduler_conns": ["*localhost"],
755 // },
756
757
758 // "migrator": {
759 //      "out_datadb_type": "redis",
760 //      "out_datadb_host": "127.0.0.1",
761 //      "out_datadb_port": "6379",
762 //      "out_datadb_name": "10",
763 //      "out_datadb_user": "cgrates",
764 //      "out_datadb_password": "",
765 //      "out_datadb_encoding" : "msgpack",
766 //      "out_stordb_type": "mysql",
767 //      "out_stordb_host": "127.0.0.1",
768 //      "out_stordb_port": "3306",
769 //      "out_stordb_name": "cgrates",
770 //      "out_stordb_user": "cgrates",
771 //      "out_stordb_password": "",
772 //      "users_filters": [],
773 // },
774
775
776 // "dispatchers":{                                     //
    ↪DispatcherS config
777 //      "enabled": false,                             // starts_
    ↪DispatcherS service: <true/false>.
778 //      "indexed_selects":true,                       // enable_
    ↪profile matching exclusively on indexes
779 //      // "string_indexed_fields": [],               // query indexes_
    ↪based on these fields for faster processing
780 //      "prefix_indexed_fields": [],                  // query indexes based_
    ↪on these fields for faster processing
781 //      "nested_fields": false,                       //
    ↪determines which field is checked when matching indexed filters(true: all; false:
    ↪only the one on the first level)
782 //      "attributes_conns": [],                       //
    ↪connections to AttributeS for API authorization, empty to disable auth_
    ↪functionality: <"*internal|x.y.z.y:1234>
783 // },

```

(continues on next page)

(continued from previous page)

```
784
785
786 // "analyzers":{                                     //
787 ↪AnalyzerS config
787 //           "enabled":false                         //
787 ↪starts AnalyzerS service: <true/false>.
788 // },
789
790
791 // "apiers": {
792 //           "enabled": false,
793 //           "caches_conns":["*internal"],
794 //           "scheduler_conns": [],                  //
794 ↪connections to SchedulerS for reloads
795 //           "attributes_conns": [],                //
795 ↪connections to AttributeS for CDRExporter
796 // },
797
798 }
```

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

6.1 API Calls

API calls are documented in the following [GoDoc](#)

6.2 CDR Server

An important component of every rating system is represented by the CDR Server. CGRateS includes an out of the box CDR Server component, controllable 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:

6.2.1 CDR-CGR

Available as handler within http server.

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

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

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

- ToR: type of record, meta-field, should map to one of the TORs hardcoded inside the server `<*voice!*data!*sms>`
- OriginID: represents the unique accounting id given by the telecom switch generating the CDR
- OrderID: Stor order id used as export order id

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

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

Example of sample CDR generated simply using curl:

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

6.2.2 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.

6.2.3 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)
```

(continues on next page)

(continued from previous page)

```

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
}

```

6.3 CDR Client (cdrc)

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

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

Controlled within *cdrc* section of the configuration file.

Has two modes of operation:

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

Principles behind functionality:

- Monitor/process a CDR folder (IN) as outlined above.
- For every file processed, extract the information based on configuration and post it via configured mechanism to CDRS.
- The fields extracted out of each CDR row are the same ones depicted in the CDRS documentation (following primary and extra fields concept).
- Once the file processing completes, move it in it's original format in another folder (OUT) in order to avoid re-processing. Here it's worth mentioning the auto-detection of duplicated CDRs at server side based on accid and host fields.
- Advanced configuration like forking a number of simultaneous client instances monitoring different folders possible through the use of *.xml* configuration.

6.3.1 Import Templates

To specify 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.

6.3.1.1 CGR-RSR Regexp Rule

Format:

```
~path:s/regexp_search_and_capture_rule/output_template/
```

Example of usage:

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

6.3.1.2 CGR-RSR Static Rule

Format:

```
^path:static_value
```

Example of usage:

Input CDR field:

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

Capture Rule: ^account:MasterAccount

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

CDR Formats supported:

6.3.2 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 position 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>]....`

6.4 CDR Exporter

Component to retrieve rated CDRs from internal CDRs database.

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

6.4.1 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.

6.4.1.1 CGR-RSR Regexp Rule

Format:

```
~path:s/regexp_search_and_capture_rule/output_template/
```

Example of usage:

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

6.4.1.2 CGR-RSR Static Rule

Format:

```
^path:static_value
```

Example of usage:

Input CDR field:

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

Capture Rule: `^account:MasterAccount`

Result after processing: `{ "account": "MasterAccount" }`

Export interfaces implemented:

6.4.2 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 ("APIerSv1.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 $\langle *voice|*data|*sms \rangle$
- 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 $\langle prepaid|postpaid|pseudoprepaid|rated \rangle$.
- direction: matching the supported direction identifiers of the CGRateS $\langle *out \rangle$
- 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 date-time (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
```

6.4.3 CGR-FWV

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

6.4.4 Hybrid CSV-FWV

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

6.5 CDR Stats Server

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

Principles of functionality:

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

6.5.1 Configuration

Individual StatsQueue configurations are loaded inside TariffPlan definitions, 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         ActionTriggerPriorityList
}

```

6.5.2 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

6.5.3 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.

6.5.4 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": "",
```

(continues on next page)

(continued from previous page)

```

    "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"'
```

6.6 DerivedCharging

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

DerivedCharging occurs in two separate places:

- **SessionManager**: necessary to handle each derived (emulated) session in its individual 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.

6.6.1 Configuration

DerivedCharging is configured in two places:

- Platform level configured within *cgrates.cfg* file.
- Account level configured as part of *TarifPlans* definition 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_
}

```

(continues on next page)

(continued from previous page)

```

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.

6.7 Rating logic

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

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

```

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

```

When the session manager receives a call start event it will first check if the call is prepaid or postpaid. If the call is postpaid then 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 apply 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 or 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 account 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.

6.7.1 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 weight 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.

7.1 Asterisk Integration Tutorials

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

7.1.1 Software installation

We have chosen Debian Jessie as operating system.

7.1.1.1 CGRateS

CGRateS can be installed using the instructions found [here](#).

7.1.1.2 Asterisk

We got **Asterisk14** installed via following commands:

```
apt-get install autoconf build-essential openssl libssl-dev libsrtplib-dev libxml2-dev
↳ libncurses5-dev uuid-dev sqlite3 libsqlite3-dev pkg-config libedit-dev
cd /tmp
wget --no-check-certificate https://raw.githubusercontent.com/asterisk/third-party/
↳ master/pjproject/2.7.2/pjproject-2.7.2.tar.bz2
wget --no-check-certificate https://raw.githubusercontent.com/asterisk/third-party/
↳ master/jansson/2.11/jansson-2.11.tar.bz2
wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-16-current.tar.gz
tar xzvf asterisk-16-current.tar.gz
cd asterisk-16.1.0/
./configure --with-jansson-bundled
make
```

(continues on next page)

(continued from previous page)

```
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.

7.1.2 SIP UA - Jitsi

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

7.1.3 Asterisk interaction via *ARI*

7.1.3.1 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 (pre-paid/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*).

7.1.3.2 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
```

7.1.3.3 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
```

7.1.3.4 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.

7.1.3.5 CGRateS Usage

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

7.1.4 CGRateS Usage

7.1.4.1 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, followed by 20cents per minute charged in 10s increments.
 - Rate id: *RT_1CNT* having no connect fee and a rate of 1 cent per minute, chargeable in 1 minute increments.
 - Rate id: *RT_1CNT_PER_SEC* having no connect fee and a rate of 1 cent per second, chargeable in 1 second increments.
- Accounting part will have following configured:
 - Create 3 accounts: 1001, 1002, 1003.
 - 1001, 1002 will receive 10units of **monetary* balance.

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

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

- Make sure all our balances were topped-up:

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

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

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

7.1.4.2 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 stopped in the middle of debit interval).

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

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

1002 -> 1001

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

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

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

1001 -> 1003

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

7.1.4.3 CDR Exporting

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

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

7.1.4.4 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).

7.2 FreeSWITCH Integration Tutorials

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

7.2.1 Software installation

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

7.2.1.1 CGRateS

CGRateS can be installed using the instructions found [here](#).

7.2.1.2 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.

7.2.2 FreeSWITCH generating *http-json* CDRs

7.2.2.1 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*).

7.2.2.2 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
```

7.2.2.3 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
```

7.2.2.4 CDR processing

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

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

7.2.2.5 CGRateS Usage

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

7.3 Kamailio Integration Tutorials

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

7.3.1 Software installation

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

7.3.1.1 CGRateS

CGRateS can be installed using the instructions found [here](#).

7.3.1.2 Kamailio

We got **Kamailio** installed via following commands:

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

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

7.3.2 Kamailio interaction via *evapi* module

7.3.2.1 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*).

7.3.2.2 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
```

7.3.2.3 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
```

7.3.2.4 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.

7.3.2.5 CGRateS Usage

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

7.4 OpenSIPS Integration Tutorials

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

7.4.1 Software installation

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

7.4.1.1 CGRateS

CGRateS can be installed using the instructions found [here](#).

7.4.1.2 OpenSIPS

We got **OpenSIPS** installed via following commands:

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

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

7.4.2 OpenSIPS interaction via *event_datagram*

7.4.2.1 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*).

7.4.2.2 Starting OpenSIPS with custom configuration

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

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

```
opensipsctl moni
```

7.4.2.3 Starting CGRateS with custom configuration

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

Make sure that cgrates is running

```
cgr-console status
```

7.4.2.4 CDR processing

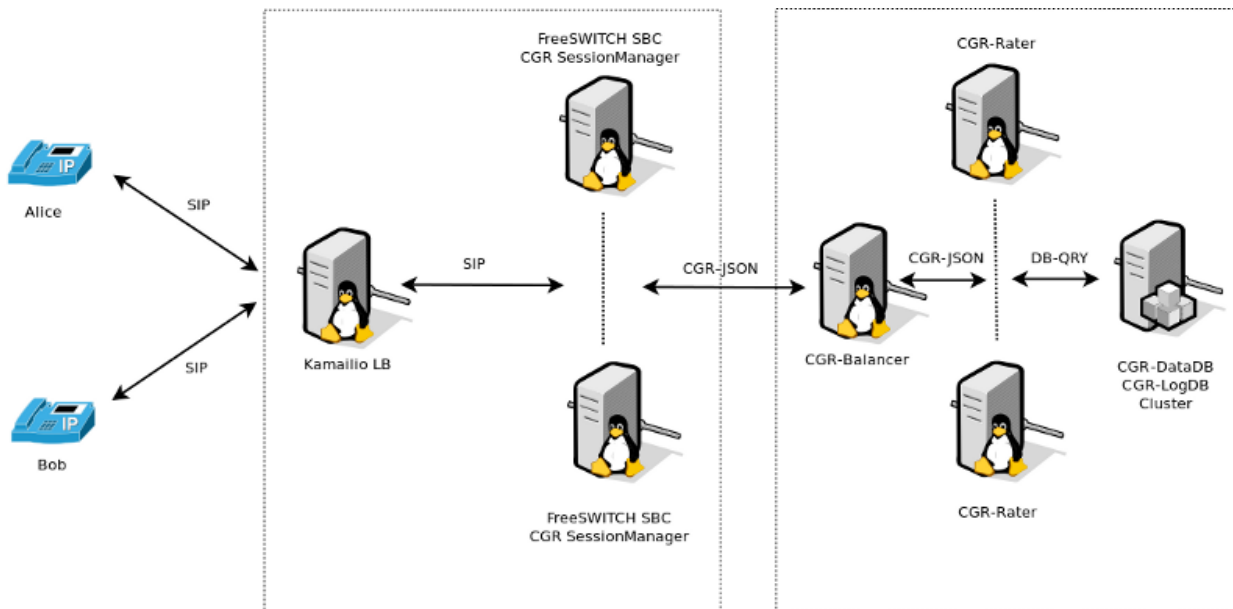
At the end of each call **OpenSIPS** will generate an CDR event and due to automatic handler registration built in **CGRateS-SM** component, this will be directed towards the port configured inside *cgrates.json*. This event will reach inside **CGRateS** through the *SM* component (close to real-time).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

7.4.2.5 CGRateS Usage

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

8.1 8.1. FreeSWITCH integration

Being the original platform supported by CGRateS, **FreeSWITCH** has the advantage of support for complete set of CGRateS features. When used as Telecom Switch it fully supports all rating modes: **pre-paid/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 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** if 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 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 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 successive mediation processes with different indexes).

8.1.2.2 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.