Garderos Documentation

GCS - Garderos Configuration Server

Garderos GmbH
PDF generated at: Thu, 17 Mar 2022 12:59:28 CET

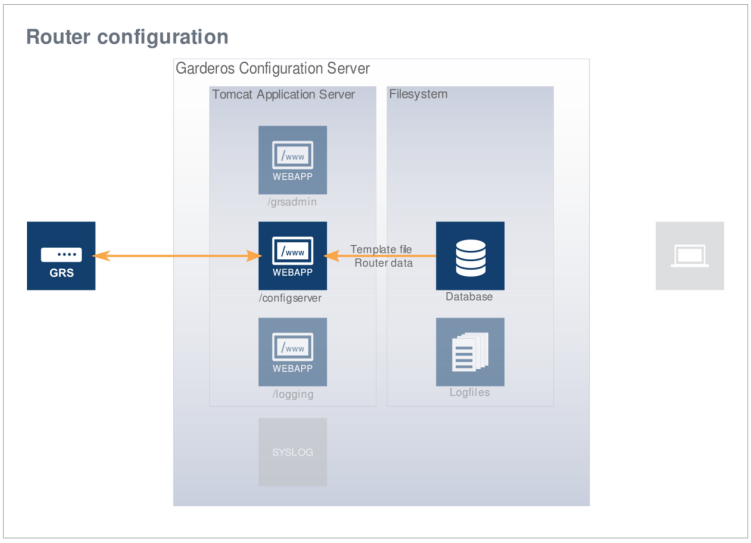
Overview

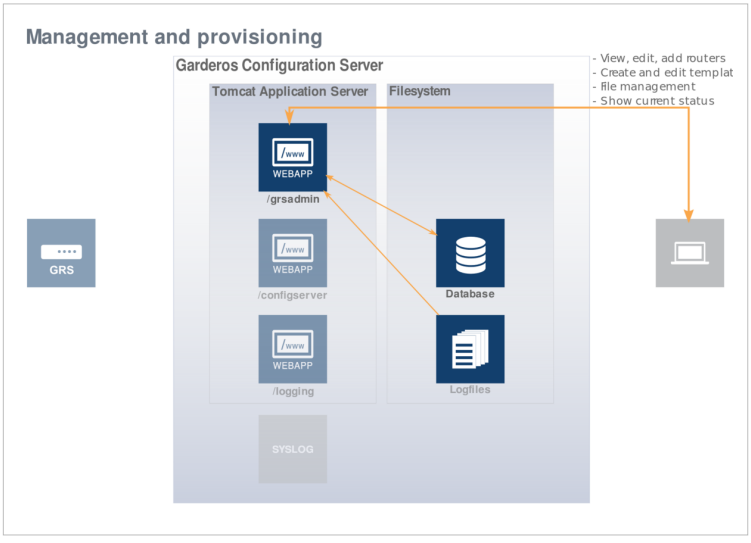
The Garderos routers are designed to be installed in large numbers in distributed networks. They use an HTTP-API for auto-configuration. Centrally stored configuration files can be downloaded and the router will check for updates of the configuration file within a configurable period of time. For remote configuration every GRS router needs a valid network configuration and the URL of at least one configuration server. The GRS then requests a configuration from the server at every startup and then periodically as defined in the configuration (typically every hour).

For this scenario a powerful centralized configuration and management tool is good to save time and cost. The *Garderos Configuration Server* is such a tool and covers the following tasks:

- Deliver configuration files to the routers
- Deliver firmware updates to the routers
- Central management system for all routers
- Collect log information from the routers
- Collect statistic data from the routers

The essential parts of the Garderos Configuration server:





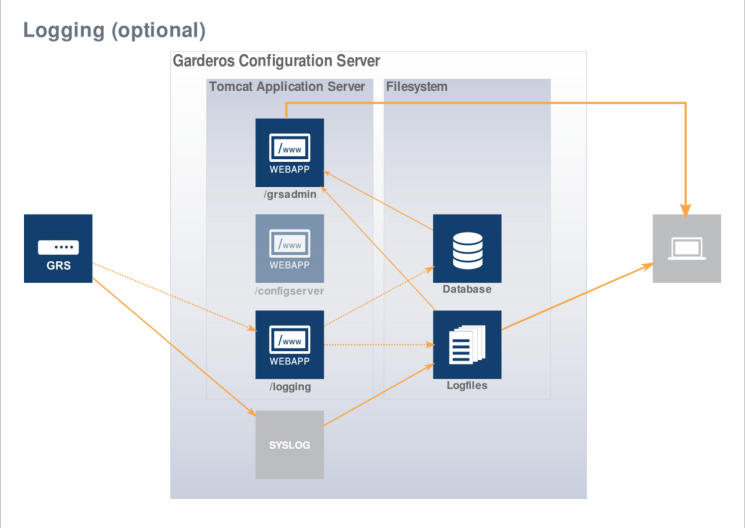
Web application *configserver* (mandatory)

The main task of the *configserver*-webapp is, to create configuration files dynamically on request and deliver them to the GRS routers. Basis for the configuration files are **template files** which contain the static configuration lines and the values which are valid on all routers, like the interfaces to activate or the addresses of NTP servers or SNMP servers. The content of the template files is combined with dynamic values taken from a database.

Beside the main task the *configserver*-webapp can provide the following functions:

- Provide a secured download of files to GRS routers, like firmware updates, scripts or certificates. Downloading a file is only possible for GRS routers which are stored in the database and if the authentication was successful

- Collect and store information about configuration requests from the GRS routers, e.g. timestamp, IP address, delivered configuration file, etc.
- Provide an interface to deliver the collected information (the *adm*-servlet)



SQL database (mandatory)

The database contains an entry for each configured router with the individual settings for this router.
In many cases routers and router locations are already managed by an existing inventory management system. In this case the Garderos Configuration Server can be integrated with an existing database.
Garderos Configserver installations are using **MySQL** or **MariaDB** database systems.

Web application *grsadmin*

If no integration with an existing database is required, the administration servlet *GRSAdmin* allows to add, change and remove router configurations from the Garderos Configuration Server's database It is a web based GUI, requiring almost no integration effort.

GRSAdmin provides the following functions:

- Router management (edit, add and delete routers in the database)
- Template management (edit, add and delete template files in the database)
- File management (management of GRS update images, scripts, certificates)
- Provide access to information stored in the configserver-webapp, that means reprocessing the information delivered by the *adm*-servlet to display it in a nice format
- Provide access to logfiles, e.g. Tomcat logfiles or GRS logfiles collected by a syslog server
- Weblogger: an integrated servlet that allows to store data sent within an HTTP request to a file or to a database

See GRS Admin documentation for details.

Web application *logging* (optional)

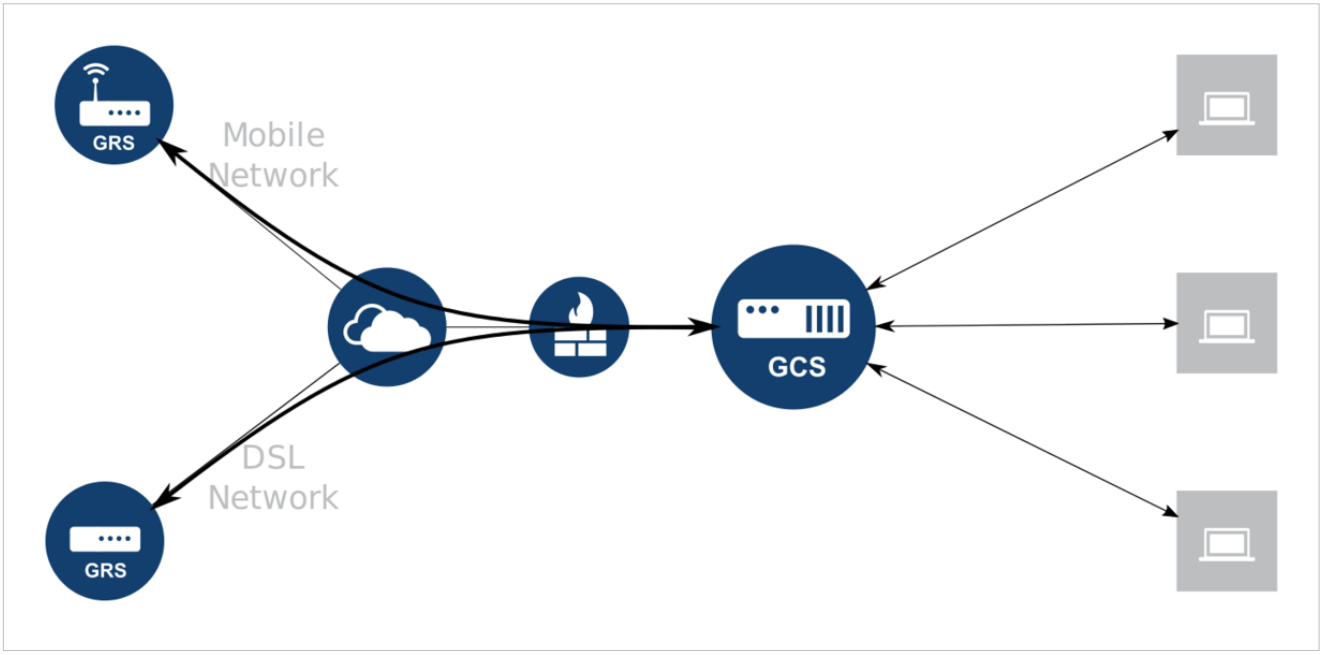
A web application that allows to store statistical data sent within an HTTP request to a database. This web application is typically used for collecting statistical data like router uptime, CPU temperature, LTE signal strength (RSSI, RSRP), LTE band information and many more.

This tool can be used for a limited number of routers (a few hundred) if no SNMP monitoring system is available.

See GRS Logging documentation for details.

Example usages

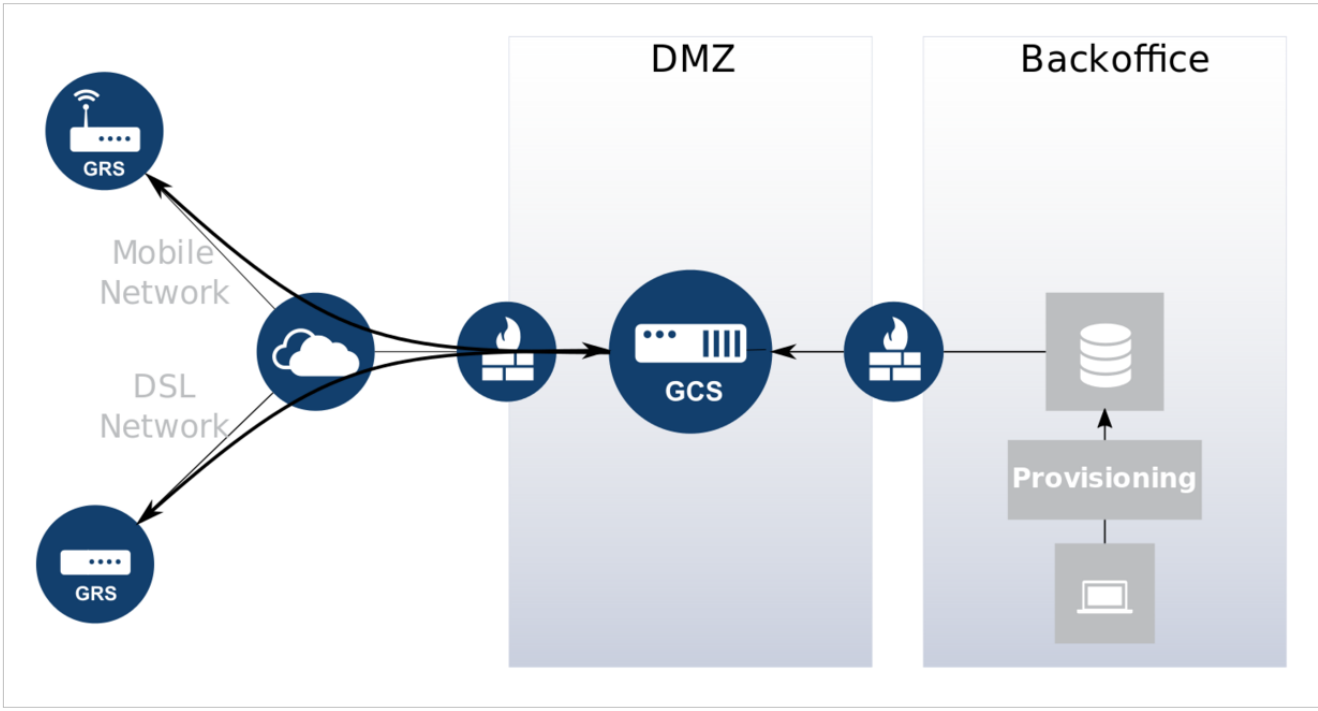
Example 1: Configserver with *GRSAdmin* web application



- Configuration data for the GRS routers are created and modified within GRSAdmin.
- Template-files and/or update files are managed in GRSAdmin.
- GRSAdmin fetches the information about the latest GRS configuration requests from the *configserver*-webapp and displays them table-formatted
- GRSAdmin displays log information from the configserver-webapp and/or GRS routers

For installation and configuration of the webapps **configserver** and **grsadmin** please see Configuration Server: Installation

Example 2: Database is managed by customer's provisioning system



- Configuration data for each GRS router is created and stored in the customer's provisioning system.
- The required information for the routers is exported from the customer's system to a database in configserver-format.
- Template-files and/or update files are managed by a 3rd party MySQL management tool (e.g. *phpmyadmin*).

Prerequisites

- Tomcat 7, Version > 7.0.43 (otherwise the logging won't work properly), Tomcat 8 or Tomcat 9
- Java 8 or newer
- MySQL or MariaDB with InnoDB database engine (otherwise CSV import and export won't work properly), version 5.7 or newer.

Terminology

Term	Definition
Configuration Server	The complete set of installed an configured components, usually all running together on a real server hardware or virtual machine.
Web application (webapp)	A set of one or more dynamic web pages and/or one or more servlets bundled in one common subdirectory of the web server.
Servlet	A servlet is a small program that runs on a server. Typically a servlet listens for HTTP requests from clients (in this case GRS routers or a browser) and then dynamically creates the response by collecting data from a database, the file system or the server's memory.
configserver	The web application which is responsible for delivering configuration files to Garderos routers and keeping data of the requests.
grsadmin (named as GRSAdmin in documentation)	A web application which provides database management features as well as displaying log files and statistics. The user interface of the Configuration Server.

config-servlet	A configurable servlet running within the configserver-webapp, which dynamically builds GRS configuration files based on a template and the result of a database query.
adm-servlet	A servlet running within the configserver-webapp, which allows access to the data storage, where the configserver-webapp keeps information about GRS connections.
GRS cache	A table in the DBMS, where the configserver webapp stores information about the configuration requests from the GRS routers. The stored data are: *timestamp*, *GRS IP address*, *GRS name*, *GRS serial number*, *GRS version*, *URL of config-servlet* and some other.
logging webapp	A separate web application, which stores data received by an HTTP request into a database.
template	A plain text file which contains key-value pairs corresponding to to the GRS configuration API line by line. If a line contains a placeholder as a value, this placeholder is replaced by the configserver webapp with a value taken from the database.

Changelog

Recent changes to the Garderos Configuration Server.

Intermittent revisions which are not mentioned here are usually minor changes, like changing log messages or optimisations to the GUI (alignment, design etc.).

Applying a patch:

The following steps have to be applied dependent on the components and hints mentioned in the table of each change entry.

For updates of *configserver*, *grsadmin* and *logging* component you can always use the latest released version, intermediate releases can be skipped.

Be aware of database changes when skipping one or more releases. All database changes made between your current running version and the destination version have to be applied.

The current valid database schema is always part of the delivered package, see the file *examples/database/schema.sql*

How to apply a configuration server patch:

- **Important: Note your current running release version.** Check:
  - *About* page which is linked at the bottom right corner of GRSAdmin. Note the higher SW-Release number of GRS Admin or Configserver.
  - The same information can be found in `/var/lib/tomcat*/webapps/configserver/WEB-INF/version.txt` or `/var/lib/tomcat*/webapps/grsadmin/WEB-INF/version.txt`
- Unpack the latest Configserver archive from Garderos(*Configserver-Cxxx-Gxxx-Lxxx.tgz*) on your server, e.g. in */home/garderos*
- If new component *configserver* has a higher release number than your installation, it has to be updated:
  - Locate the file *configurator-patch-rxxx.war*, where xxx ≥ revision number
  - `cd /var/lib/tomcat*/webapps/configserver`
  - **`rm -f WEB-INF/lib/*`**
  - `sudo jar -xvf /path/to/patch/configurator-patch-rxxx.war`
- If new component *GRSAdmin* has a higher release number than your installation, it has to be updated:
  - Locate the file *grsadmin-patch-rxxx.war*, where xxx ≥ revision number
  - `cd /var/lib/tomcat*/webapps/grsadmin`
  - **`rm -f WEB-INF/lib/*`**
  - `sudo jar -xvf /path/to/patch/grsadmin-patch-rxxx.war`
- If component *logging* is installed, and new component has a higher release number as the installed component:
  - Locate the file *logging-patch-rxxx.war*, where xxx ≥ revision number
  - `cd /var/lib/tomcat*/webapps/logging`
  - **`rm -f WEB-INF/lib/*`**
  - `sudo jar -xvf /path/to/patch/logging-patch-rxxx.war`
- **Search the first entry in this changelog, where the revision number is higher than your previously installed one.**
  Read all following entries carefully:
  - If the database schema has to be updated:

    - Log in to your database: `mysql -u<user> -p<password> config`
    - Apply mentioned changes
  - If in one of the *.../WEB-INF/web.xml* configuration options are new or changed, compare your version with the *.new* version in the same subdirectory, if you want to make changes.
- **Important: Always restart Tomcat service after patching and database updates:**
  `sudo service tomcat restart`

HINT Some files within the webapps subdirectories contains data which is probably adapted to customer requirements during customization process. This means:

- Customizations and modifications should only be made to these files. Other modifications will be lost when patching.
- If e.g. the *web.xml* file contains new options due to a new feature, an update has to be applied manually. In this case a hint is written to the changelog table.

The patch files do NOT touch these files (usually there is a *.new* copy near the original file):

```
configserver/WEB-INF/web.xml
configserver/WEB-INF/classes/log4j2.xml
...
grsadmin/WEB-INF/web.xml
grsadmin/WEB-INF/classes/log4j2.xml
grsadmin/WEB-INF/classes/forms.properties
grsadmin/WEB-INF/classes/forms_de.properties
grsadmin/custom/routermod_add_sql.jsp
grsadmin/custom/routermod_formpatterns.jsp
...
logging/WEB-INF/web.xml
logging/WEB-INF/classes/log4j2.xml
```

r619: Feature: Enable customization of *Create router*

Date	Component	Revision	DB change	Comment
2017-08-09	GRSAdmin	r619	no	Files in *grsadmin/custom*, see details

Details:

- **routermod_add_sql.jsp:** A new file where customer can add own code to be applied when inserting a router to the database. Can be used e.g. to insert random passwords or default values for any column. An example is provided within the installation.
- **routermod_formpatters.jsp:** A new file where customer can add own validators for any form field. Creating appropriate validators prevents that any user inserts data to the database which may create errors later on the routers. An example is provided within the installation.

The files located in the *custom* subdirectory are not touched when updating (patching) GRSAdmin.

r625: Feature: Page *Router status* now configurable

Date	Component	Revision	DB change	Comment
2017-09-12	GRSAdmin	r625	no	Change in *grsadmin/WEB-INF/web.xml*.

Details:

On the page *Router status* the following items are now configurable. For configuration edit the file *grsadmin/WEB-INF/web.xml*.

- Columns to be shown on status page: See context-parameter *statusheaders*.
- Time range, when values in column *Last access* are shown in red: See context-parameter *max_confi_age*.
  This value should correspond to the reconfigure-interval of the connected routers.

r646: New features for grscache: store X-AC-Status

Date	Component	Revision	DB change	Comment
2017-10-19	configserver, GRSAdmin	r646	yes	Changed in *grsadmin/WEB-INF/web.xml* necessary to enable status column.

Details:

X-AC-Status: A GRS router (GRS 3.4 and following) which already got a remote configuration, sends the status of applying the configuration within the following request. Current status codes:

- **200** Configuration successfully applied
- **400** Error in downloaded configuration. See GRS log for details.

The modified GRSAdmin (/grsadmin) is able to display the status within the routerstatus page.

Database modification:

```
ALTER TABLE grscache ADD COLUMN `laststatus` int(5) after lastaccess;
```

r656: Change: Current adminuser can not change own user level

Date	Component	Revision	DB change	Comment
2017-11-28	GRSAdmin	r656	no	

Details:

Current logged in administrator now can not change its own user level. Thus it is not possible that an admistrator locks himself out from GRSAdmin.

r664: Bugfix for handling line separators in template files

Date	Component	Revision	DB change	Comment
2017-12-12	configserver, GRSAdmin	r664	no	

Details:

When a template file contains line separator '\r\n' (instead of '\n') the *include* statements were not correctly processed. This situation could happen when template files are created on Windows machines and uploaded to the Configserver.

r665: Internal: Default language file renamed

Date	Component	Revision	DB change	Comment
2017-12-12	GRSAdmin	r665	no	Renaming of files necessary, see *Details*.

Details:

Improved handling of the default language when server has to decide which language file to use. Thus renamed *content_en.properties* to *content.properties* and *forms_en.properties* to *forms.properties*.

When applying a patch to GRSAdmin, please rename the files manually:

```
cd /var/lib/tomcat*/webapps/grsadmin/WEB-INF/classes
mv forms_en.properties forms.properties
mv content_en.properties content.properties
```

r673: Security bugfix for GRSAdmin: Block unauthorized servlet access.

Date	Component	Revision	DB change	Comment
2018-02-01	GRSAdmin	r673	no	Mantis #4285

Details:

Under certain conditions it has been possible to call the internal servlets e.g. for getting the router table without authentication before. Access is restricted now.

r675: Bugfix for GRSAdmin: Wrong date calculation in Apple Safari browser

Date	Component	Revision	DB change	Comment
2018-03-05	GRSAdmin	r675	no	

Details:

Time calculation for highlighting overdue routers on page Router Status was not working in Apple Safari browser.

r678: Change for GRSAdmin: Show only not-existing template names in create form.

Date	Component	Revision	DB change	Comment
2018-03-22	GRSAdmin	r678	no	

Details:

Show only not-existent templates in option field for create new templates.

r680: Change for GRSAdmin: Dropdown for template names modified on *Modify router*

Date	Component	Revision	DB change	Comment
2018-03-23	GRSAdmin	r680	no	

Details:

Datalist for proposal of template names is now based on real existing template files, not on the list of configured template names as before.

r685: New feature for configserver and GRSAdmin: Handling of time-scheduled template content

Date	Component	Revision	DB change	Comment
2018-04-18	Configserver and GRSAdmin	r685	no	

Details:

It is now possible to create template content, which is delivered to the routers dependent from the current time.
This shall give users the possibility to prepare a configuration, where certain parts are activated or deactivated at a certain time.

Schedule markers are defined like this:

```
#define <marker> <from-time> <to-time>
```

All parts are separated by single space. Any string is allowed for <marker>. Timestamps are in format "yyyy-MM-ddTHH:mm:ss"

Example:

```
#define T1 1970-01-01T00:00:00 2018-04-22T03:00:00
```

Later in template these defined markers can be used for individual lines:

```
   #if <marker> <config_line>
```

Example: #if T1 system.version=003_004_022

How it works:

- If the current time is within the range defined for a marker, the corresponding comments and markers are removed for any matching 'if'-line
- If the current time is out of the range for a marker, the comments are kept in file

How to use:

- Multiple markers with different ranges can be defined
- The "#if" statement has to be placed in front of each individual line which should be handled.
- To handle a block of configuration lines, use "@include" statement which is handled by a marker
- Include-files can also contain definitions and markers.
- Markers are only valid within the scope of the file.

r688: New feature for GRSAdmin: Syntax highlighting for templates

Date	Component	Revision	DB change	Comment
2018-05-25	GRSAdmin	r688	no	

Details:

Improved handling of template files by applying syntax highlighting:

- Invalid configuration parameters are shown in red.
- Comments are printed in gray.
- Variables are marked in bold blue.
- Include lines are marked
- Time definitions are marked, time based content is evaluated

Notes:

- Parameter values are not checked if they are valid for this parameter (e.g. if it is an IP-Adress etc.)
- Only GRS can evaluate if the configuration is valid.

r697: GRSAdmin and Configserver adapted for Tomcat 8 and Java 9

Date	Component	Revision	DB change	Comment
2018-06-04	GRSAdmin, Configserver	r697	no	

Details:

Minor changes to make GRSAdmin and Confiserver work with Tomcat 8 and Java 9 as uses in Ubuntu 18.04.

r698: Bugfix for handling configserver's logfile

Date	Component	Revision	DB change	Comment
2018-06-05	GRSAdmin	r698	no	

Details:

Fix bug where logfile content could not be displayed if file contained empty lines.

r700: New feature for Configserver: System variable ${ CFG_REQ_URL } for template files

Date	Component	Revision	DB change	Comment
2018-06-25	Configserver	r700	no	

Details:

Up to now URLs for file download could only be defined static in templates. When using two or more configservers in a rendundant setup, this was a restriction which made redundant setups complicated.

Now the following parameter can be used in template files: **${ CFG_REQ_URL }**

On a GRS configuration request, this variable will be replaced by the full URL which was used by the GRS to request the configuration, e.g. https://config1.garderos.com:8443/configserver/config".

Example usage:

```
system.schedule.exec.0.cron=*/5 * * * *
system.schedule.exec.0.script=${ CFG_REQ_URL }?file=script.sh
```

The configserver will deliver the following configuration:

```
system.schedule.exec.0.cron=*/5 * * * *
system.schedule.exec.0.script=https://config1.garderos.com:8443/configserver/config?file=script.sh
```

r707: Bugfix: Improve syntax highlighting for templates

Date	Component	Revision	DB change	Comment
2018-07-12	GRSAdmin	r707	no	

Details:

Improve parsing of partial config lines (while typing) and handling of 'no'-lines.

r714: Internal: Enable *forceFastReload* in default web.xml

Date	Component	Revision	DB change	Comment
2018-08-02	Configserver	r714	no	Change in configserver/WEB-INF/web.xml

Details: With this update, the feature *forceFastReload* (see Configserver->context parameters) is enabled by default and set to 66 seconds.

r719: New feature for GRSAdmin: 'About'-Page

Date	Component	Revision	DB change	Comment
2018-08-31	GRSAdmin	r719	no	

Details:

An *About*-Page is introduced and linked in the page footer (bottom right corner). The page shows:

- Current running release versions of Configserver and GRSAdmin components
- License information

r722: Internal: SQL update now writes NULL values instead of empty strings to database

Date	Component	Revision	DB change	Comment
2018-09-04	GRSAdmin	r722	no	

Details:

If any field is not set (=empty) in the form *Modify router*, *NULL* values are now written to the database for these columns (was an empty string before).

Advantage: NULL values are UNIQE in SQL, which means any column can be defined to contain only distinct values, but multiple NULL values. This allows to define columns for e.g. serial or imsi to be UNIQE in default schema.

r723: Updated CSV import for GRSAdmin

Date	Component	Revision	DB change	Comment
2018-09-05	GRSAdmin	r723	no	

Details:

Update CSV import function:

- Use updated opencsv library (4.2 instead of 3.7)
- Updated import and export code.
- Insert empty values as NULL values to DB instead of empty strings.
- Improve logging.

r730: New feature for Configserver: System variable ${ CFG_REQ_SRV } for template files

Date	Component	Revision	DB change	Comment
2018-09-07	Configserver	r730	no	

Details:

Like the previously introduced ${ CFG_REQ_URL }, this parameter returns the server URL without the path. This can be used if any files to be downloaded are not handled by the configserver process.

Now additionally the following parameter can be used in template files: **${ CFG_REQ_SRV }**

On a GRS configuration request, this variable will be replaced by the full URL which was used by the GRS to request the configuration, e.g. https://config1.garderos.com:8443".

Example usage:

```
system.file.0.name=local:certificate.crt
system.file.0.url=${ CFG_REQ_SRV }/certificates/certificate.crt
```

The configserver will deliver the following configuration:

```
system.file.0.name=local:certificate.crt
system.file.0.url=https://config1.garderos.com:8443/certificates/certificate.crt
```

r737: New feature for GRSAdmin: Upload of .tgz and .tar files now possible

Date	Component	Revision	DB change	Comment
2018-10-24	GRSAdmin	r737	no	

Details:

Archive files with extensions *.tar* and *.tgz* are now accepted to be uploaded.

r742: Bugfix for configserver: Using a non-existent variable in a commented line is now possible.

Date	Component	Revision	DB change	Comment
2018-11-15	Configserver	r742	no	

Details:

Using a non-existent variable name in a commented line of a template caused an internal error. The template file could not be delivered. Example:

```
#system.description=${ notexistent-description } This line is only for documentation.
```

After applying this patch, these variables can be used.

r743: New features for grscache: store X-AC-Last-Config-Hash

Date	Component	Revision	DB change	Comment
2018-11-16	configserver	r743	yes	

Details:

X-AC-Last-Config-Hash: A GRS router which already got a remote configuration, sends the hash-value of that configuration within the next configuration request. After enabling this feature by adding the column *lastconfighash* to the *grscache*-table within the database, this value is written to the database on each request.

Database modification:

```
ALTER TABLE grscache ADD COLUMN `lastconfighash` varchar(32) after laststatus;
```

r744: New features for GRSAdmin: Not existent variables in template files are marked.

Date	Component	Revision	DB change	Comment
2018-11-16	GRSAdmin	r744	no	

Details:

Problem: When using variables like **${ description }** in template files a column with the corresponding name must be existent in database. If any non-existent name is used, the configuration file can not be created, the requesting GRS does not receive any configuration.

After applying this patch, the variables which are not existent in the database are marked in red, so the operator can repair the template or the database before saving.

r749: Internal: Add IPv6 variant of *localhost* to default web.xml

Date	Component	Revision	DB change	Comment
2018-11-20	Configserver	r749	no	Change in configserver/WEB-INF/web.xml

Details:

Added IPv6 address `0:0:0:0:0:0:0:1` to init-parameter *allowedIP* of *adm*-servlet.

By default the only allowed host for calling the *adm*-servlet is localhost. On IPv6 capable systems calling the URL *http:///localhost/configserver/adm* caused errors. This can be fixed with this update.

r751: New features for grscache: store X-AC-Imsi and configstatus

Date	Component	Revision	DB change	Comment
2018-11-23	configserver	r751	yes	

Details:

X-AC-Imsi: If a SIM card is inserted to a GRS router, it always sends the IMSI of the SIM card within the configuration request. After enabling this feature by adding the column *lastimsi* to the *grscache*-table within the database, this value is written to the database on each request.

configstatus: A GRS router which already got a remote configuration, sends the hash-value of that configuration within the next configuration request. As the configuration server also calculates the hash value before sending the response, any configuration change can be detected. After enabling this feature by adding the column *configstatus* to the *grscache*-table within the database, the following values are written to the cache table:

- INIT: This is the very first request from that device, usually the device is newly installed at that time.
- Not-modified: The device will receive the same configuration as it had prevously received.
- Modified: A modified configuration is sent to the device.

Database modification:

```
ALTER TABLE grscache ADD COLUMN `configstatus` varchar(32) after laststatus;
ALTER TABLE grscache ADD COLUMN `lastimsi` varchar(32) after laststatus;
```

r752: Internal: For options *dbCache* and *filesInDB* the default is now *true*.

Date	Component	Revision	DB change	Comment
2018-11-29	configserver	r752	no	Change in *configserver/WEB-INF/web.xml*

Details:

The context parameters *dbCache* and *filesInDB* can now be removed from web.xml file.

If customer wants explicitly to deactivate one of these options, values have to be set to *false*.

r754: New features for GRSAdmin: Button "Show GRS view" on showconfig.jsp

Date	Component	Revision	DB change	Comment
2018-11-30	GRSAdmin	r754	no	

Details:

New button "Show GRS view" on the page which is showing the resulting configuration file.

By clicking this button, the created configuration file is displayed sorted and without comments. Lines which may cause unexpected configuration on GRS (e.g. multiple definitions of any parameter, missing values in attributes) are highlighted, so the operator can take measures to clean or repair template files.

r766: New feature for configserver and GRSAdmin: Extend handling of time-scheduled template content, allow 'day/night' settings

Date	Component	Revision	DB change	Comment
2019-03-04	Configserver and GRSAdmin	r766	no	

Details:

It is now possible to create template content, which is delivered to the routers dependent from the current **time of day**.
This shall give users the possibility to prepare a configuration, where certain parts are activated or deactivated at a certain time of day.

Schedule markers are defined like this:

```
#define <marker> <from-time> <to-time>
```

All parts are separated by single space. Any string is allowed for <marker>. Timestamps are in format "HH:mm"

Example:

```
#define day 07:00 19:00
```

Later in template these defined markers can be used for individual lines:

```
   #if <marker> <config_line>
```

Example: #if day system.description=These are day settings

See documentation or changelog for r685 to see further details.

r784: Database field (boolean) dependent content

Date	Component	Revision	DB change	Comment
2019-08-16	Configserver and GRSAdmin	r784	optional	Database change only when using this feature.

Details:

It is now possible to create template content, where it is possible to activate or deactivate parts of the configuration by a boolean (true=on/false=off) value in the database.

- First a database column of type 'boolean' has to be added to your config table:
  `ALTER TABLE config ADD COLUMN <your_column_name> boolean;`

- Now this variable can be used in templates within the *#if*-statement, similar to the time scheduled content:
  `#if ${ <your_column_name> } <config_line>`
  Example: `#if ${ wifion } @include wifi.txt`

See documentation for details.

r792: Protected (secure) files

Date	Component	Revision	DB change	Comment
2019-09-04	Configserver, GRSAdmin	r792	yes	

Details:

Secure file storage (table **securefiles** of the configdb). Files which are stored here can only be downloaded by the defined router.

These files are for example:

- Router certificate files
- Router private key files

Accessing the downloads is possible by the following URL:

```
http(s)://<configserver>:<port>/configserver/config?secfile=<filename_to_download>
```

Database modification:

```
CREATE TABLE `securefiles` (
  `filename` varchar(60) COLLATE utf8_unicode_ci NOT NULL,
  `grsname` varchar(32) COLLATE utf8_unicode_ci NOT NULL,
  `type` enum('template','include','text','script','cert','key','grsimage','other') COLLATE utf8_unicode_ci DEFAULT 'other',
  `lastaccess` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `lastadmin` varchar(32) COLLATE utf8_unicode_ci DEFAULT NULL,
  `size` int(20) DEFAULT NULL,
  `content_text` text COLLATE utf8_unicode_ci,
  `content_img` longblob,
  UNIQUE KEY `filename` (`filename`,`grsname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

r800: Database variables are now supported in *@include* and *#define* lines

Date	Component	Revision	DB change	Comment
2019-09-18	Configserver, GRSAdmin	r800	optional	Database changes only necessary when using the feature, then *grsadmin/WEB-INF/web.xml* has also to be modified.

Details:

It is now possible to use variables **${ <db_column_name> }** in ***@include***- and ***#define***-Lines. This allows finer control of connected routers and reducing the amount of templates.

When using this feature, the necessary database columns have to be created, if not yet present:

- Add column to the database

```
ALTER TABLE config ADD COLUMN <your_column_name> varchar(32);
```

e.g.

```
ALTER TABLE config ADD COLUMN includefile varchar(32);
```

or

```
ALTER TABLE config ADD COLUMN night_range varchar(32);
```

- Modify GRSAdmin for editing the value in the database.
  It is recommended to define a syntax check (especially for the time range) in the GRSAdmin form to avoid inserting illegal values which later cause errors in the templates.
- Modify the template to use the new variables:

```
@include ${ includefile }
```

or

```
#define night ${ night_range }
```

r809: Inline configuration of GRSAdmin

Date	Component	Revision	DB change	Comment
2019-12-02	GRSAdmin	r809	optional	database changes only necessary when using the feature

Details:

By default all settings of GRSAdmin are stored within the file */var/lib/tomcat*/webapps/grsadmin/WEB-INF/web.xml*. To apply changes (e.g. hide or show columns on status page) it has been necessary to edit the file on the server itself, then restart the Tomcat servlet container.

From this release, it is possible to modify the GUI settings within the GUI itself. The settings are stored in the database and when using multiple configuration servers, the changes are set on every connected server.

Using the inline configuration which is stored in the database is activated by default for new installations starting with revision r822 as of January, 2020

When using this feature, the necessary database table has to be created, if not yet present:

- Add table *webconfig* to the database

```
CREATE TABLE `webconfig` (
  `param-name` varchar(32) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL COMMENT 'parameter name for web gui',
  `param-value` varchar(500) CHARACTER SET utf8 COLLATE utf8_unicode_ci DEFAULT NULL COMMENT 'parameter value',
  UNIQUE KEY `param-name` (`param-name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

- Apply default settings to the webconfig table, **or modify the following lines to your needs**:

```
/* default values for GRSAdmin */
INSERT INTO webconfig VALUES('modfields','serial,imsi,ip,client1IP,client1DHCP,description,location');
INSERT INTO webconfig VALUES('modbooleans','wifion');
INSERT INTO webconfig VALUES('tableheaders','serial,client1IP,client1DHCP,description,location');
INSERT INTO webconfig VALUES('statusheaders','version,serial,ip,servlet,servername,count');
INSERT INTO webconfig VALUES('max_config_age','3600');
```

```
INSERT INTO webconfig VALUES('showlaststatus','true');
INSERT INTO webconfig VALUES('dbshowcolumns','uptime,cputemp,wwansignal,wwannetwork,wwanband');
INSERT INTO webconfig VALUES('dbshowgraphs','cputemp,wwansignal');
INSERT INTO webconfig VALUES('showsecurefiles','true');
INSERT INTO webconfig VALUES('adminlog','false');
INSERT INTO webconfig VALUES('syslog','false');
INSERT INTO webconfig VALUES('logdb','true');
INSERT INTO webconfig VALUES('syslog_dir','/var/log/grssyslog/');
INSERT INTO webconfig VALUES('configserver_url','http://localhost:8080/configserver/');
INSERT INTO webconfig VALUES('defaultTableLength','15');
```

After creating the database, the config file *grsadmin/WEB-INF/web.xml* has to be edited, the *<context-param>*-entries have to be commented or removed. Note the current settings and apply to the database accordingly.

See Documentation for more information.

r815: Logfile of Admin GUI can be displayed

Date	Component	Revision	DB change	Comment
2019-12-06	GRSAdmin	r815		

Details: It is now also possible to display the content of */var/log/tomcat*/grsadmin.log* on the GUI, the page is called *Admin GUI Logfile*. Content of this log is among others when administrators were logged in and which templates or routers were changed.

To enable the page, go to *Settings* and activate *Show Admin GUI Log* in common settings.

r818: Export buttons for router table and statistics

Date	Component	Revision	DB change	Comment
2020-01-16	GRSAdmin	r818		

Details:

It is now possible to export the router status table and the router statistics table by a simple click to a button, either to clipboard or as CSV or Excel file.

r819: #ifnot definition in templates

Date	Component	Revision	DB change	Comment
2020-01-17	Configserver and GRSAdmin	r819	no	

Details:

It is now possible to define a **#ifnot** line in templates, which can be used to activate configuration lines when a database boolean is **not true** (i.e. *false*), or if a defined time range is **not valid**.

- Boolean database variables can be used in templates now additionally with an **#ifnot**-statement:
  `#ifnot ${ <your_column_name> } <config_line>`
  Example:
  `#ifnot ${ wifion } no interface.wlan`

- Time based markers can also be used in templates now additionally with an **#ifnot**-statement:
  `#ifnot day <config_line>`
  Example:
  `#define day 08:00 20:00`
  `#ifnot day no interface.wlan`

See documentation for details.

r824: Improve showing statistics

Date	Component	Revision	DB change	Comment
2020-02-04	GRSAdmin	r824	no	

Details: Statistics page improved

- If statistics pages are activated, a link to the router statistics is displayed on router status page.
- Sorting improved for certain columns, like uptime and WWAN signal

r827: More flexibility for @include in templates

Date	Component	Revision	DB change	Comment
2020-02-15	GRSAdmin and configserver	r827	no	

Details:

More flexibility when using @include-statements in templates, now combinations of static text and database variables are allowed, e.g.

- `@include include.${ name }`
- `@include ${dbvar1}.${dbvar2}`
- `@include filepart.${dbvar1}.${dbvar2}`

r828: Improve syntax highlighting for templates

Date	Component	Revision	DB change	Comment
2020-02-26	GRSAdmin	r828	no	

Details: Improve the syntax highlighting in templates and created config files:

- Allow more than one variable per line
- Improved details of lines starting with #if, #ifnot and #define

r834: New GRSAdmin features: copy, rename and compare templates

Date	Component	Revision	DB change	Comment
2020-03-05	GRSAdmin	r834	no	

Details: The *Templates* subpage now has a context menu with the following options:

- Copy a template file
- Rename a template file
- Compare two template files in a new window, showing them side-by-side with highlighted differences

To use these new functions simply make a right-click to the template name.

r838: Keep history of file changes in database

Date	Component	Revision	DB change	Comment
2020-03-06	GRSAdmin	r838	yes	

Details: Introduce new table *fileshistory*, which keeps track of previous text file versions saved in the database. For example, if a modified template file is stored to the database, the previous content is automatically saved to this table.

Saving the previous content is done by MySQL trigger functions.

Changes to the database:

```
CREATE TABLE `fileshistory` (
  `filename` varchar(60) COLLATE utf8_unicode_ci NOT NULL,
  `type`
enum('template','include','text','script','cert','key','grsimage','other')
 COLLATE utf8_unicode_ci DEFAULT 'other',
  `lastmodified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
 CURRENT_TIMESTAMP,
  `lastmodifiedby` varchar(32) COLLATE utf8_unicode_ci DEFAULT NULL,
  `size` int(20) DEFAULT NULL,
  `content_text` text COLLATE utf8_unicode_ci
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

DELIMITER $$
CREATE TRIGGER `modFILE` BEFORE UPDATE ON `files` FOR EACH ROW BEGIN
INSERT INTO fileshistory SELECT
filename,type,NOW(),lastadmin,size,content_text FROM files WHERE
filename=OLD.filename COLLATE utf8_unicode_ci;END $$
CREATE TRIGGER `delFILE` BEFORE DELETE ON `files` FOR EACH ROW BEGIN
INSERT INTO fileshistory SELECT
filename,type,NOW(),lastadmin,size,content_text FROM files WHERE
filename=OLD.filename COLLATE utf8_unicode_ci;END $$
DELIMITER ;
```

r845: New GRSAdmin feature: Write admin name to files and securefiles table

Date	Component	Revision	DB change	Comment
2020-03-09	GRSAdmin	r845	yes	

Details: The name of the administrator which did the latest modifications to any file, especially template files, is now written to the database. This is an alternative way (besides the logfiles) to track the latest changes of files on the configuration server.

Changes to the database:

```
ALTER TABLE files ADD COLUMN lastadmin varchar(32) AFTER lastaccess;
ALTER TABLE securefiles ADD COLUMN lastadmin varchar(32) AFTER lastaccess;
```

r855: New feature: Configuration servlets with a 'default' template

Date	Component	Revision	DB change	Comment
2020-05-15	configserver, GRSAdmin	r855	no	

Details: It is now possible to configure a config-servlet, which ignores the device-type of the router and builds the configuration based on the same template for all routers.

Use case: You have several groups of routers which get their running configuration based on different templates, but all routers shall get the identical startup configuration. Until now a separate startup template for each device type was necessary, now it is possible to share the same startup template for all routers.

New configuration parameter:

The following parameter within one of the servlet definitions configures the default servlet:

```
    <init-param>
        <param-name>default-template</param-name>
        <param-value>default</param-value>
    </init-param>
```

r861: New feature: Function *encode* for templates

Date	Component	Revision	DB change	Comment
2020-05-19	configserver, GRSAdmin	r861	no	

Details: A new function for templates converts cleartext passwords (e.g. taken from database) to a *{sha256}*-hashed value, which is used for user- and enable passwords of Garderos routers.

Use case: You can have a cleartext password in the database, but it will be sent already encoded to the router.

Usage example in templates:

```
user.account.0.password=${ encode ( "sha256" , password    ) }   #Value from database column 'password' will be hashed
user.account.1.password=${ encode ( "sha256" , "user1pass" ) }   #String 'user1pass' will be hashed
```

See Documentation for details.

r863: Improve handling of uploads in *Protected files*

Date	Component	Revision	DB change	Comment
2020-06-22	GRSAdmin	r863	no	

Details: Improved handling of secure file uploads:

- If an upload error occurs, the router name is kept in the form field.
- A list of allowed files for update is shown.

r865: New feature for masking the router secret and customer fields

Date	Component	Revision	DB change	Comment
2020-06-24	GRSAdmin	r865	no	

Details: It is now possible to hide passwords in the database view of GRSAdmin. This is by default activated for the router secret (with a button to show the password in cleartext) and can additionally be activated for all text-based customer's fields stored in the database.

The options are in each case:

- Mask the password, show button for cleartext view
- Mask the password
- Do not mask the password

Notes for installation:

1. Apply the patch for GRSAdmin as described on top of this chapter.
2. Go to the directory */var/lib/tomcat*/webapps/grsadmin/custom*.
   - Create a copy of the existing file *routermod_formpatterns.jsp*:
     `sudo cp routermod_formpatterns.jsp routermod_formpatterns.jsp.old`
   - Replace the original file with the new one::
     `sudo mv routermod_formpatterns.jsp.new routermod_formpatterns.jsp`
   - If you do not have custom modifications in the original file, you are done.
   - If you have modifications in the original file, copy the modified content to the new file.

Check the documentation for more details:

- Mask or unmask the shared secret
- Router form validation

r874: Fix bug: Filter on *Router Status* page breaks *Protected Files* page

Date	Component	Revision	DB change	Comment
2021-03-17	GRSAdmin	r874	no	

Details: When a filter was used on *Router Status* page, a later call of the page *Protected files* just showed an error message instead of the file list. This bug is fixed with this release.

r876: New function for templates: Dynamic creation of an IPv6 IID based on a database column

Date	Component	Revision	DB change	Comment
2021-04-12	Configserver and GRSAdmin	r876	no	

Details: This function can create an individual IPv6 host address out of a database value, e.g. the serial number:

Example: This configuration in a template

```
interface.dummy.0.ip.static.0.ipv6=fd00::${ ipv6iid ( 4,1, serial )}/64
```

will create the following configuration:

```
interface.dummy.0.ip.static.0.ipv6=fd00::772:211:200:03/64
```

r893: Remove restrictions in file upload

Date	Component	Revision	DB change	Comment
2021-06-02	GRSAdmin	r893	no	

Details: Most restrictions regarding upload files are removed. Any file (even binaries) up to 64MB can now be uploaded to the files table.

It still only makes sense to upload files which can be processed or used on a GRS router.

---

r898: Advanced filters for tables

Date	Component	Revision	DB change	Comment
2021-06-24	GRSAdmin	r898	no	

Details: Introduce Datatable's SearchPanes to pages which contain tables (Router status, Routers in database, Templates...). This can ease some searching and filtering tasks when working with the admin tool.

Just click on the **Show/hide filters** text above the table.

r900: Browse in files history

Date	Component	Revision	DB change	Comment
2021-06-28	GRSAdmin	r900	depends	Check if the necessary DB change of **r838** is applied (see above).

Details: Since the update coming with r838 all changes to templates and scripts are automatically written to the table *fileshistory* in the database.

This revision adds a new page **Files history** to GRSAdmin, which allows to see previous file versions as well as comparing selected versions.

r902: Add authentication option to logging servlet

Date	Component	Revision	DB change	Comment
2021-07-09	Logging	r902	no	

Details: A new configuration option of the *logging* web app allows to require basic authentication for each logging message which shall be stored in the database. This can prevent abuse of the logging database.

Example: **Add the parameters *loguser* and *logpasswd* to *.../webapps/logging/WEB-INF/web.xml* '**

```
   <servlet>
      <servlet-name>loggerSecure</servlet-name>
      <servlet-class>com.garderos.grstools.LoggerServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
       <init-param>
           <param-name>log_to_db</param-name>
           <param-value>true</param-value>
       </init-param>
       <init-param>
           <param-name>loguser</param-name>
           <param-value>logme</param-value>
       </init-param>
       <init-param>
           <param-name>logpasswd</param-name>
           <param-value>logmepass</param-value>
       </init-param>
   </servlet>
```

When logging within a script, which uses *curl* for sending the values, the following parameter has to be added to the call

```
curl -u logme:logmepass ....
```

r936: New feature: Function *add* for templates

Date	Component	Revision	DB change	Comment
2021-12-09	configserver, GRSAdmin	r936	no	

Details: A new function to calculate an offset based on a database variable. Can be used for example to calculate different IP addresses based on a common value.

Example:

```
interface.dummy.0.ip.static.0.ipv6 = fd00:${network}::1/64
interface.dummy.1.ip.static.0.ipv6 = fd00:${ add ( "hex" , 1 , network ) }:1/64
interface.dummy.2.ip.static.0.ipv6 = fd00:${ add ( "hex" , "A" , network ) }:1/64
```

If the database field *network* contains the value `1a01`, the result would be:

```
interface.dummy.0.ip.static.0.ipv6 = fd00:1a01::1/64
interface.dummy.1.ip.static.0.ipv6 = fd00:1a02:1/64
interface.dummy.2.ip.static.0.ipv6 = fd00:1a0b:1/64
```

More information can be found in Template functions

r939: Code cleanup requires some configuration changes

Date	Component	Revision	DB change	Comment
2021-12-14	configserver, GRSAdmin	r940	no	Update of configuration files required.

Details: Some code had to be cleaned up, and some functions are moved internally for better division of getting data and presentation. This makes it necessary to update the following configuration files:

- /var/lib/tomcat*/webapps/configserver/WEB-INF/web.xml
- /var/lib/tomcat*/webapps/grsadmin/WEB-INF/web.xml

The new versions of these files are stored within the same subdirectories, named *web.xml.new*. These new files have to be compared with the current web.xml files, new lines have to be integrated in the running config file. Basically the following changes have to be made:

In *`/var/lib/tomcat*/webapps/configserver/WEB-INF/web.xml`* add the following lines after the last <servlet>-definition:

```
    <servlet>
        <servlet-name>getServerLog</servlet-name>
        <servlet-class>com.garderos.grs.config.GetServerlog</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getServerLog</servlet-name>
        <url-pattern>/getServerLog</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>getSysInfo</servlet-name>
        <servlet-class>com.garderos.grs.config.GetSysInfo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getSysInfo</servlet-name>
        <url-pattern>/getSysInfo</url-pattern>
    </servlet-mapping>
```

In *`/var/lib/tomcat*/webapps/grsadmin/WEB-INF/web.xml`* add the following lines after the last <servlet>-definition:

```
    <servlet>
        <servlet-name>allGrsConfigsJson</servlet-name>
        <servlet-class>com.garderos.grsadmin.AllGrsConfigsJson</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>allGrsConfigsJson</servlet-name>
        <url-pattern>/allGrsConfigsJson</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>getSysinfo</servlet-name>
```

```
        <servlet-class>com.garderos.grsadmin.GetSysinfo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getSysinfo</servlet-name>
        <url-pattern>/getSysinfo</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>getServerLog</servlet-name>
        <servlet-class>com.garderos.grsadmin.GetServerLog</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getServerLog</servlet-name>
        <url-pattern>/getServerLog</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>getPlotData</servlet-name>
        <servlet-class>com.garderos.grsadmin.GetPlotData</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>getPlotData</servlet-name>
        <url-pattern>/getPlotData</url-pattern>
    </servlet-mapping>
```

Then restart Tomcat.

Additionally (only when using the feature for auto-creation of database values) the following file should be updated:

- /var/lib/tomcat*/webapps/grsadmin/custom/routermod_add_sql.jsp

Just adapt the example in the file routermod_add_sql.jsp.new

r940: Important security update: Fix CVE-2021-44228 ("Log4shell")

Date	Component	Revision	DB change	Comment
2021-12-15	configserver, GRSAdmin, logging	r940	no	**Critical update**

Details:

A critical vulnerability has been discovered in the Apache Log4j2 library. Details about the vulnerability can be found at:

nvd.nist.gov/vuln/detail/CVE-2021-44228 [1]

With this update, all GCS web applications are updated to log4j **2.16.0**.

Important:

When applying the patch, it is important to remove the previous libraries (see the statement `rm -f WEB-INF/lib/*`) in each subdirectory.

To check your server after the update, call the following commands:

```
cd /
find . | grep log4j | grep jar
```

When executing this search, there should be no versions of log4j < 2.16.0 be found.

r943: Updated Log4J2 security updates: Fix CVE-2021-44832

Date	Component	Revision	DB change	Comment
2022-01-12	configserver, GRSAdmin, logging	r943	no	

Details:

Another vulnerability has been discovered in the Apache Log4j2 library. Details about the vulnerability can be found at:

[2]

With this update, all GCS web applications are updated to log4j **2.17.1**.

Important:

When applying the patch, it is important to remove the previous libraries (see the statement `rm -f WEB-INF/lib/*`) in each subdirectory.

To check your server after the update, call the following commands:

```
cd /
find . | grep log4j | grep jar
```

When executing this search, there should be no versions of log4j < 2.17.1 be found.

r946: Fix bug: Avoid unnecessary error message when creating routers when no booleans are defined.

Date	Component	Revision	DB change	Comment
2022-01-27	GRSAdmin	r945	no	

Details: If no boolean variables are used within the router fields to modify (i.e. the value for *Boolean Fields* on the settings page is empty), an unnecessary error message was displayed when creating a new router in the database, although the router creation was successful.

r947: New feature: Function *defineHeader* for templates

Date	Component	Revision	DB change	Comment
2022-02-02	configserver, GRSAdmin	r947	no	

Details: A new function to compare header values of the requesting GRS with a defined value.

Syntax:

#defineHeader <marker> <headername> <operator> <compare-value>

Dependent on the operation result the marker is either set to true or false and can later be used in *#if* or *#ifnot* statements.

Examples:

```
#defineHeader isversion GRSVersion = 003_006_044
#defineHeader hasSimCard X-AC-Imsi > 1
#
```

```
#Use defined markers
#if isversion interface.ppp.0.multi-user.0.password=abc123
#if isversion interface.ppp.0.multi-user.0.username=testuser
```

More information can be found in GRS header dependent content

r948: Fix bug: Avoid error message for first GRS connect after server restart.

Date	Component	Revision	DB change	Comment
2022-02-07	configserver	r948	no	

Details: If a device was **not** identified by the GRS name (e.g. by the serial number) and the GRS name did **not** match the name in the database, an error message was written to the log file. Example message:

```
[https-jsse-nio-8443-exec-7] ERROR [config.DBConnector]: Error while
checking cache entry for device 'grs_default': Current position is
after the last row
```

r949: Fix bug: Settings-Page can not handle database column names ending with "-name".

Date	Component	Revision	DB change	Comment
2022-02-09	GRSAdmin	r949	no	

Details: If a database-column in the config-database was ending with the letters *name* or *secret*, the field could not be added to the *Router modify fields* using the *Settings*-page.

r953: Change: Allow @includes within included files (nested includes)

Date	Component	Revision	DB change	Comment
2022-02-15	configserver, GRSAdmin	r953	no	

Details: It is now possible to create nested includes, that means any included file can contain an include itself. Maximum depth is 10 includes (to avoid infinite includes when creating a loop by mistake).

See Merge templates

r963: Fix bug: #ifnot statement not correctly processed

Date	Component	Revision	DB change	Comment
2022-03-16	configserver	r963	no	

Details: The '#ifnot' statement was not correctly processed in combination with boolean values from the database, e.g. this statement caused an error:

```
#ifnot ${ wifion } @include wifi.txt
```

See Database dependent content

References

[1] https://nvd.nist.gov/vuln/detail/CVE-2021-44228
[2] https://nvd.nist.gov/vuln/detail/CVE-2021-44832

Installation

This document describes the basic steps for installing the Garderos Configuration Server on Ubuntu or CentOS Linux distribution. After finishing these installation steps, the Configuration Server is fully functional, but the following issues are **not** covered:

- Security
- Data backup
- Redundancy

While this installation may be good enough for a test or trial phase, take care of these issues when switching to a live scenario with many connected GRS routers. Please read the corresponding chapters of this documentation after finishing this installation.

The installation on other Linux distributions is possible and usually quite similar, but some paths and configuration files might be different.

Installing and configuring the server software needs superuser-privileges on every operating system. For having these privileges there are different options:

- on **CentOS**/**Redhat**: directly log in as root user or execute "su -" followed by the root password - **not recommended**
  → it is better to enable *sudo* for your standard user, as described later.
- on **Ubuntu**: log in as standard user; all commands which need root privilege must be called with "sudo"

Be careful not to destroy your operating system, when working as superuser. We do not take responsibility for this.

Operating System

The Garderos Configuration Server should work on any Linux Distribution which provides Tomcat 7, Tomcat 8 or Tomcat 9, MySQL ≥ 5.7 or MariaDB ≥ 5.5, but installation documentation and support is only provided for the following operating systems. Other Debian or RedHat based distributions should be quite similar, also installation on SLES 12 was performed successfully.

Ubuntu 18.04 LTS / **20.04 LTS** [show/hide]

- Download a current Ubuntu-18.04 or Ubuntu-20.04 server ISO image file.
- Install Ubuntu with the standard settings.
- During Ubuntu installation it is required to define a user. In this documentation **garderos** is used.
- When installing Ubuntu Server edition, you're asked, which additional packages shall be installed to the minimum installation. Please do not select any, they will be added manually afterwards.

After finishing the installation the system should be updated by following commands:

```
sudo apt-get update && sudo apt-get dist-upgrade && sudo apt-get autoremove
```

Then install the following packages (if not automatically installed):

```
sudo apt-get install openssh-server
```

CentOS 7.x [show/hide]

- Download the current CentOS-7 minimal ISO image file.
- Install CentOS with the standard settings.
- While or after installation a valid network configuration has to be defined, to get the updates and necessary packages.

- Install updates:

```
yum update
```

- Install useful basic tools (if not present):

```
yum install vim
yum install elinks
yum install wget
yum install tcpdump
yum install ntpdate
```

CentOS 8.x [show/hide]

Changes in CentOS 8/RedHat 8 compared to previous releases:

- **Apache Tomcat is no longer available in the repositories and has to be installed manually**.
- *dnf* packet manager is the successor of *yum* and used in this manual.

First steps:

- Download the current CentOS-8 minimal ISO image file.
- Install CentOS with the standard settings.
- While or after installation a valid network configuration has to be defined, to get the updates and necessary packages.
- Install updates:

```
dnf -y update && dnf -y upgrade
```

- Install useful basic tools (if not present):

```
dnf install -y wget net-tools vim chrony tar
```

The initial configuration of all services is not covered in this installation manual, but it is in the responsibility of the customer / operator. Examples are:

- Network configuration
- System updates
- Firewall settings
- NTP configuration

Application services

Unpack the Garderos part

All necessary files are within the archive **Configserver-C<version>_G<version>_L<version>.tgz**. This archive contains

- **configurator-r<version>.war** -> the configuration server webapp
- **configurator-patch-r<version>.war** -> an update patch for configuration server webapp which does not touch customer settings
- **grsadmin-r<version>.war** -> the management-webapp
- **grsadmin-patch-r<version>.war** -> an update patch for the management-cd exwebapp which does not touch customer settings
- **logging-r<version>.war** -> the logging-webapp
- **logging-patch-r<version>.war** -> an update patch for logging webapp which does not touch customer settings

- **db_and_libs.tgz** -> this archive contains database templates, configuration templates and libraries

Untar the archive to any folder, e.g. into the folder *garderosinstall* in your home directory (the following steps will assume you have chosen this directory):

```
cd ~
mkdir garderosinstall
cd garderosinstall
tar -xvf ../Configserver-C<version>_G<version>_L<version>.tgz
```

The archive **db_and_libs.tgz** contains the database schema and some example configuration files and database content.

```
tar -xzvf db_and_libs.tgz
```

The created directories are

```
|...
|-examples
 `|-database
  |  `example_content.sql
  |   schema.sql
  |   (some helper scripts)
  `-etc
```

Basic installation

Ubuntu 20.04 LTS [show/hide]

Install Tomcat 9 application server:

```
sudo apt-get install tomcat9
```

To run the Java *jar* binary, usually the installation of the JDK is required. As the JDK is a huge package (more than 400MB) and not needed later, it is recommended to install *fastjar* instead, which is sufficient for installation.

Possible:

```
sudo apt install default-jdk
```

Better:

```
sudo apt install fastjar
```

Ubuntu 18.04 LTS [show/hide]

Install Tomcat 8 application server:

```
sudo apt-get install tomcat8
```

To run the Java *jar* binary, usually the installation of the JDK is required. As the JDK is a huge package (more than 400MB) and not needed later, it is recommended to install *fastjar* instead, which is sufficient for installation.

Possible:

```
sudo apt install default-jdk
```

Better:

```
sudo apt install fastjar
```

CentOS 7.x [show/hide]

Install Tomcat 7 application server:

```
yum install tomcat
```

Install Java8 JDK, necessary for java *jar* binary:

```
yum install java-1.8.0-openjdk-devel
```

Prepare Tomcat to be started at server startup

```
systemctl enable tomcat
```

Although not standard, it could be a good idea to put your user into the *sudoers* file, so that it is not always necessary to log in as root. Additionally the rest of the installation instructions are then the same for all distributions:

```
su -                          # become root
visudo                        # uncomment the following line:   %wheel  ALL=(ALL)       ALL
usermod -aG wheel garderos    # Add user 'garderos' to group 'wheel'
```

Then log out and log in as 'garderos' user again. *sudo* should work now.

CentOS 8.x [show/hide]

Install Tomcat 9 application server:

Since RedHat 8/CentOS 8 there is no Tomcat package available in the standard repositories anymore. This is why Tomcat 9 has to be installed from the official sources directly:

```
#Install Java
dnf install -y java-11-openjdk-devel

#Download and extract tomcat in /usr/share - Release number may change
cd /usr/share/
wget https://downloads.apache.org/tomcat/tomcat-9/v9.0.46/bin/apache-tomcat-9.0.46.tar.gz
tar -xzvf apache-tomcat-9.0.46.tar.gz
mv apache-tomcat-9.0.46 tomcat9/
rm -f apache-tomcat-9.0.46.tar.gz

#Create user and start script
useradd -r tomcat
chown -R tomcat:tomcat /usr/share/tomcat9
#Create init script with the following content:
vim /etc/systemd/system/tomcat.service
```

Content of **/etc/systemd/system/tomcat.service:**

```
[Unit]
Description=Apache Tomcat Server
After=syslog.target network.target

[Service]
```

```
Type=forking
User=tomcat
Group=tomcat

Environment=CATALINA_PID=/usr/share/tomcat9/temp/tomcat.pid
Environment=CATALINA_HOME=/usr/share/tomcat9
Environment=CATALINA_BASE=/usr/share/tomcat9

ExecStart=/usr/share/tomcat9/bin/catalina.sh start
ExecStop=/usr/share/tomcat9/bin/catalina.sh stop

RestartSec=10
Restart=always
[Install]
WantedBy=multi-user.target
```

```
#Enble Tomcat9 at server startup
systemctl daemon-reload
systemctl enable tomcat
```

Create symbolic links, so the manual installation feels like the former installation by RPM package:

```
ln -s /usr/share/tomcat9/ /var/lib/tomcat
ln -s /usr/share/tomcat9/logs/ /var/log/tomcat
ln -s /usr/share/tomcat9/conf /etc/tomcat
```

The paths in the following installation guide will only be valid, when these symbolic links are set.

Although not standard, it could be a good idea to put your user into the *sudoers* file, so that it is not always necessary to log in as root. Additionally the rest of the installation instructions are then the same for all distributions:

```
su -                          # become root
visudo                        # uncomment the following line:   %wheel  ALL=(ALL)       ALL
usermod -aG wheel garderos    # Add user 'garderos' to group 'wheel'
```

Then log out and log in as 'garderos' user again. *sudo* should work now.

Database installation (MySQL / MariaDB)

Ubuntu 20.04 LTS [show/hide]

- Install mysql server

```
sudo apt-get install mysql-server
```

- Install mysql-java connector within the Tomcat lib directory (as of May, 2020 the new Ubuntu 20.04 does not contain package for libmysql-java):

```
cd ~/garderosinstall
wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.49.tar.gz
tar -xzvf mysql-connector-java-5.1.49.tar.gz
sudo cp mysql-connector-java-5.1.49/mysql-connector-java-5.1.49.jar /usr/share/tomcat9/lib/
```

First a database user with the authentiation method *mysql_native_password* has to be created. This will be the user to be configured in Tomcat for the SQL connection.

Adapt usernames and passwords in the following example.

For local login only:

```
sudo mysql -u root                                                    # Use "sudo"
```

```
mysql> USE mysql;
mysql> CREATE USER 'garderos'@'localhost' IDENTIFIED BY 'garderos'; # Only local login possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'localhost';  # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'localhost'; # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

If additionally remote (network) login is required:

```
sudo mysql -u root                                                    # Use "sudo"
```

```
mysql> USE mysql;
mysql> CREATE USER 'garderos'@'%' IDENTIFIED BY 'garderos';         # Login from every external IP possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'%';          # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'%';         # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

Now create the database schema by copy/pasting the following file to the console. This file can be found at *garderosinstall/examples/database/schema.sql*:

```
cd ~
sudo mysql -u root -p < garderosinstall/examples/database/schema.sql (the 'garderos' user is not allowed to create triggers!)
```

The database installation is now finished.

Optional tasks:

- Insert example data for an easy start

```
cd ~
mysql -u garderos -p config < garderosinstall/examples/database/example_content.sql
```

Ubuntu 18.04 LTS [show/hide]

- Install mysql server

```
sudo apt-get install mysql-server
```

- Install mysql-java connector and create a symlink within the Tomcat lib directory:

```
sudo apt-get install libmysql-java
cd /usr/share/tomcat8/lib/
sudo ln -s ../../java/mysql.jar mysql.jar
```

On **Ubuntu** the mysqld is prepared for autostart by default.

Beginning with Ubuntu 18.04.3 the default database authentication changed. The *root* user is no longer authenticated by *mysql_native_password*, but by *auth_socket*. That means a password stored in MySQL user table is ignored, but the system password for the user is taken (as the *root*-user in Ubuntu has no

password by default, *sudo* is necessary).

First a database user with the authentiation method *mysql_native_password* has to be created. This will be the user to be configured in Tomcat for the SQL connection.

Adapt usernames and passwords in the following example.

For local login only:

```
sudo mysql -u root                                                    # Use "sudo"
```

```
mysql> USE mysql;
mysql> CREATE USER 'garderos'@'localhost' IDENTIFIED BY 'garderos'; # Only local login possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'localhost';  # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'localhost'; # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

If additionally remote (network) login is required:

```
sudo mysql -u root                                                    # Use "sudo"
```

```
mysql> USE mysql;
mysql> CREATE USER 'garderos'@'%' IDENTIFIED BY 'garderos';         # Login from every external IP possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'%';          # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'%';         # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

Now reate the database schema by copy/pasting the following file to the console. This file can be found at *garderosinstall/examples/database/schema.sql*:

```
cd ~
mysql -u garderos -p < garderosinstall/examples/database/schema.sql
```

The database installation is now finished.

Optional tasks:

- Insert example data for an easy start

```
cd ~
mysql -u garderos -p config < garderosinstall/examples/database/example_content.sql
```

CentOS 7.x [show/hide]

- Install mysql server:

```
sudo yum install mariadb-server (MariaDB is the CentOS equivalent of MySQL)
```

- Install mysql-connector package:

```
sudo yum install mysql-connector-java
cd /usr/share/tomcat/lib/
sudo ln -s /usr/share/java/mysql-connector-java.jar mysql-connector-java.jar
```

- Configuration of services
  - Start mysql:

```
sudo service mariadb start
```

- Configure mariadb for first usage, execute

```
sudo /usr/bin/mysql_secure_installation
```

Set root password and confirm all other settings.

- To enable the mysql service at every server startup execute the following command:

```
sudo systemctl enable mariadb
```

Create the database schema by copy/pasting the following file to the console. This file can be found at *garderosinstall/examples/database/schema.sql*:

```
cd ~
mysql -u root -p < garderosinstall/examples/database/schema.sql
```

Now a standard database user with access to the databases *config* and *logdata* has to be created. This will be the user to be configured in Tomcat for the SQL connection.

```
mysql -u root -p

mysql> USE mysql;
mysql> CREATE USER 'garderos'@'localhost' IDENTIFIED BY 'garderos'; # Only local login possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'localhost';  # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'localhost'; # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

If additionally remote (network) login is required:

```
sudo mysql -u root                                                    # Use "sudo"
```

```
mysql> USE mysql;
mysql> CREATE USER 'garderos'@'%' IDENTIFIED BY 'garderos';         # Login from every external IP possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'%';          # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'%';         # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

The database installation is now finished.

Optional tasks:

- Insert example data for an easy start

```
cd ~
mysql -ugarderos -p config < garderosinstall/examples/database/example_content.sql
```

CentOS 8.x [show/hide]

- Install mysql server:

```
dnf install mariadb-server
```

- Install mysql-connector package:

```
cd /usr/share/tomcat9/lib/
wget https://downloads.mariadb.com/Connectors/java/connector-java-2.7.2/mariadb-java-client-2.7.2.jar
```

- Configuration of services and load at startup
  - Start MariaDB:

```
sudo systemctl enable mariadb
sudo systemctl start mariadb
```

  - Configure MariaDB for first usage, execute

```
sudo /usr/bin/mysql_secure_installation
```

    Set root password and confirm all other settings.

Create the database schema by copy/pasting the following file to the console. This file can be found at *garderosinstall/examples/database/schema.sql*:

```
cd ~
mysql -u root -p < garderosinstall/examples/database/schema.sql
```

Now a standard database user with access to the databases *config* and *logdata* has to be created. This will be the user to be configured in Tomcat for the SQL connection.

```
mysql -u root -p

mysql> USE mysql;
mysql> CREATE USER 'garderos'@'localhost' IDENTIFIED BY 'garderos'; # Only local login possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'localhost';  # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'localhost'; # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

If additionally remote (network) login is required:

```
sudo mysql -u root                                                    # Use "sudo"
```

```
mysql> USE mysql;
mysql> CREATE USER 'garderos'@'%' IDENTIFIED BY 'garderos';         # Login from every external IP possible
mysql> GRANT ALL PRIVILEGES ON config.* TO 'garderos'@'%';          # Access to config database
mysql> GRANT ALL PRIVILEGES ON logdata.* TO 'garderos'@'%';         # Access to logdata database
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

The database installation is now finished.

Optional tasks:

- Insert example data for an easy start

```
cd ~
mysql -ugarderos -p config < garderosinstall/examples/database/example_content.sql
```

Check database installation

To show if all tables and triggers are present:

```
mysql -u garderos -p config

mysql> show tables;
+------------------+
| Tables_in_config |
+------------------+
| adminusers       |
| config           |
| files            |
| fileshistory     |
| grscache         |
| property         |
| securefiles      |
| webconfig        |
+------------------+
8 rows in set (0.00 sec)


show columns from config;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| name        | varchar(32)  | NO   |     | NULL    |       |
| secret      | varchar(32)  | YES  |     | NULL    |       |
| device_type | varchar(32)  | YES  |     | NULL    |       |
| serial      | varchar(20)  | YES  | UNI | NULL    |       |
| mac         | varchar(20)  | YES  | UNI | NULL    |       |
| imsi        | varchar(20)  | YES  | UNI | NULL    |       |
| ip          | varchar(46)  | YES  |     | NULL    |       |
| description | varchar(256) | YES  |     | NULL    |       |
| location    | varchar(256) | YES  |     | NULL    |       |
| client1IP   | varchar(46)  | YES  |     | NULL    |       |
| client1DHCP | varchar(93)  | YES  |     | NULL    |       |
| wifion      | tinyint(1)   | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
12 rows in set (0.00 sec)

mysql> show triggers;
+---------+--------+--------+-----------------------------------------------------------------
| Trigger | Event  | Table  | Statement

| Timing | Created                | sql_mode               | Definer
        | character_set_client | collation_connection | Database
Collation |
+---------+--------+--------+-----------------------------------------------------------------
```

```
| insGRS  | INSERT | config | BEGIN INSERT INTO grscache (name) VALUES
(new.name COLLATE utf8_unicode_ci ); END

| AFTER  | 2020-05-04 12:11:13.80 | NO_AUTO_VALUE_ON_ZERO |
root@localhost | utf8                 | utf8_general_ci      |
latin1_swedish_ci  |
| updGRS  | UPDATE | config | BEGIN UPDATE grscache SET
lastmodified=NOW() WHERE name=new.name COLLATE utf8_unicode_ci; END
                                                              | AFTER  |
2020-05-04 12:11:13.84 | NO_AUTO_VALUE_ON_ZERO | root@localhost | utf8
                 | utf8_general_ci      | latin1_swedish_ci  |
| delGRS  | DELETE | config | BEGIN DELETE FROM grscache WHERE
grscache.name = OLD.name COLLATE utf8_unicode_ci; END
                                                                    |
BEFORE | 2020-05-04 12:11:13.78 | NO_AUTO_VALUE_ON_ZERO |
root@localhost | utf8                 | utf8_general_ci      |
latin1_swedish_ci  |
| modFILE | UPDATE | files  | BEGIN INSERT INTO fileshistory SELECT
filename,type,NOW(),lastadmin,size,content_text FROM files WHERE
filename=OLD.filename COLLATE utf8_unicode_ci;END | BEFORE | 2020-05-04
 12:11:13.98 | NO_AUTO_VALUE_ON_ZERO | root@localhost | utf8
                 | utf8_general_ci      | latin1_swedish_ci  |
| delFILE | DELETE | files  | BEGIN INSERT INTO fileshistory SELECT
filename,type,NOW(),lastadmin,size,content_text FROM files WHERE
filename=OLD.filename COLLATE utf8_unicode_ci;END | BEFORE | 2020-05-04
 12:11:14.00 | NO_AUTO_VALUE_ON_ZERO | root@localhost | utf8
                 | utf8_general_ci      | latin1_swedish_ci  |
+---------+--------+--------+-------------------------------------------------------------
5 rows in set (0.00 sec)


quit
```

Deploy and configure web applications

Important: Dependent on the selected Linux distribution and Tomcat version, the webapps path is different. Deploy the applications to */var/lib/tomcat<version>/webapps*. On CentOS the version number is omitted. So for **<webapps_dir>** use

- **/var/lib/tomcat9/webapps/** for Ubuntu 20.04 LTS
- **/var/lib/tomcat8/webapps/** for Ubuntu 18.04 LTS
- **/var/lib/tomcat/webapps/** for CentOS 7 and CentOS 8 (if symbolic links are created as mentioned)

After setting up the application server and database, deploy the web applications:

- Deploy the *configserver* webapp:

```
cd <webapps_dir>
sudo mkdir configserver
cd configserver
sudo jar xvf ~/garderosinstall/configurator-r<version>.war
```

- Deploy the *grsadmin* webapp:

```
cd <webapps_dir>
sudo mkdir grsadmin
cd grsadmin
sudo jar xvf ~/garderosinstall/grsadmin-r<version>.war
```

- Deploy the *logging* webapp (not mandatory):

```
cd <webapps_dir>
sudo mkdir logging
cd logging
sudo jar xvf ~/garderosinstall/logging-r<version>.war
```

It is also possible to use the Tomcat autodeploy feature for installing the web applications, just by copying the .war-files into the webapps base directory. Tomcat will then automatically extract the directories, but it may overwrite any existing directory and all files inside without asking. Be sure not to loose your configuration, templates or files when using this feature for upgrading an existent installation.

Configuration of *configserver*

The configuration of the database connection and web applications is done in */etc/tomcat*/server.xml* (*/etc/tomcat8/server.xml* or */etc/tomcat9/server.xml* on Ubuntu or */etc/tomcat/server.xml* on CentOS). Add the following lines to your configuration file and modify the values according to your system.

The Garderos package is shipped with an example file which should be almost ready-to-use. Just copy this file and check the content as described in the following documentation.

```
cp ~/garderosinstall/examples/etc/server.xml /etc/tomcat*/
```

Basic security - enable HTTPS

To create a basic self-signed certificate for your server, execute the following commands:

```
cd /etc/tomcat*/
keytool -genkey -alias <your-server-name-or-IP> -keysize 4096
-keyalg RSA -keypass garderos -validity 365 -keystore config.keystore
-storepass garderos
```

More details about certificate creation and security can be found in the chapter Security.

Then enable SSL/TLS in your Tomcat configuration:

Up to Tomcat 8 use:

```
vim /etc/tomcat*/server.xml
#add the connector for port 8443
```

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true" sslProtocol="TLS"
           sslEnabledProtocols="TLSv1.2"
           keystoreFile="/etc/tomcat*/config.keystore"
           keystorePass="garderos"
           keystoreType="JKS" />
```

For Tomcat 9 and later use:

```
<Connector  port="8443" protocol="org.apache.coyote.http11.Http11'''Nio'''Protocol"
               '''sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"'''
               maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
               sslProtocol="TLS"
               sslEnabledProtocols="TLSv1.2+TLSv1.3" useServerCipherSuitesOrder="true"
               ciphers="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
                       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
                       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,
                       TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,
                       TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
                       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
                       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
                       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
                       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,
                       TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,
                       TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
                       TLS_AES_256_GCM_SHA384,
                       TLS_AES_128_GCM_SHA256"
               URIEncoding="UTF-8"
               keystoreFile="/etc/tomcat9/config.keystore"
               keystorePass="garderos"
               keystoreType="JKS"
           />
```

Database connection: define the global database resource for all webapps

```
...
<GlobalNamingResources>
    ...
    ...
    <Resource name="jdbc/config" auth="Container"
              type="javax.sql.DataSource"
              maxActive="100"
              maxIdle="30"
              minIdle="10"
              maxWait="10000"
              timeBetweenEvictionRunsMillis="30000"
              minEvictableIdleTimeMillis="120000"
              validationQuery="SELECT 1"
```

```
                validationInterval="6000"
                testOnBorrow="true"
                username="garderos"
                password="garderos"
                removeAbandoned="true"
                removeAbondonedTimeout="60"
                logAbandoned="true"
                driverClassName="com.mysql.jdbc.Driver"
                factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"

url="jdbc:mysql://localhost:3306/config?autoReconnect=true&amp;amp;socketTimeout=10000&am
    </Resource>
</GlobalNamingResources>
...
```

Optional: Add logging database resource when using the logging features:

```
...
    <Resource name="jdbc/logdata" auth="Container"
                type="javax.sql.DataSource"
                maxActive="100"
                maxIdle="30"
                minIdle="10"
                maxWait="10000"
                timeBetweenEvictionRunsMillis="30000"
                minEvictableIdleTimeMillis="120000"
                validationQuery="SELECT 1"
                validationInterval="6000"
                testOnBorrow="true"
                username="garderos"
                password="garderos"
                removeAbandoned="true"
                removeAbondonedTimeout="30"
                logAbandoned="true"
                driverClassName="com.mysql.jdbc.Driver"
                factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"

url="jdbc:mysql://localhost:3306/logdata?autoReconnect=true&amp;amp;socketTimeout=60000&a
 >
    </Resource>
...
```

Modify the username, password and url-String to your needs, as shown in the following template:

```
url="jdbc:mysql://<IP_MYSQL_DBMS>:3306/<DB_NAME>?autoReconnect=true&amp;socketTimeout=60000&amp;connectTimeout=600000&useSSL=false">
```

For CentOS 8 change the driverClassName!
Change:
`driverClassName="com.mysql.jdbc.Driver"`
to
`driverClassName="org.mariadb.jdbc.Driver"`

Configure the webapp *configserver* to use this database resource

```
...
    <Host name="localhost"  appBase="webapps" unpackWARs="true" autoDeploy="true">
        ...
        ...
        <Context path="/configserver" docBase="configserver" >
                <ResourceLink name="jdbc/config" global="jdbc/config" auth="Container" type="javax.sql.DataSource" />
        </Context>
        ...
    </Host>
...
```

Configuration of grsadmin-webapp

GRSAdmin must use the same database resource as the configuration server, so add the following to */etc/tomcat*/server.xml*, straight below the context definition for the configserver-webapp (see above):

```
...
        <Context path="/grsadmin" docBase="grsadmin" >
                <ResourceLink name="jdbc/config" global="jdbc/config" auth="Container" type="javax.sql.DataSource" />
                <ResourceLink name="jdbc/logdata" global="jdbc/logdata" auth="Container" type="javax.sql.DataSource" />
        </Context>
...
```

Use the second resource *jdbc/logdata* only if the *logging* webapp is configured and in use.

For more configuration options of GRSAdmin see GRSAdmin web application

Configuration of logging-webapp (optional)

The webapp *logging* contains a servlet, which allows to send statistic data from the GRS routers by HTTP requests. If the webapp *logging* is deployed as described above, you have to configure the database connection in */etc/tomcat*/server.xml*:

```
...
<Context path="/logging" docBase="logging" >
     <ResourceLink name="jdbc/logdata" global="jdbc/logdata" auth="Container" type="javax.sql.DataSource" />
</Context>
...
```

Local firewall settings

As any other server in the Internet, the Configuration Server should be protected by a firewall, to limit access and keep the system secure. It is assumed that the server is installed in a firewall protected network, but it is possible to have an additional local firewall on the system itself.

Ubuntu

A standard installation of Ubuntu does not apply any firewall rule to the system.

CentOS

On CentOS often by default a firewall is installed and configured, allowing only very limited access to the system. To enable access to tomcat it is necessary to open the required ports, e.g. 8080 and/or 8443.

Please check the existing firewall rules:

```
sudo iptables -L
```

In case there are firewall rules set, add the following lines to **/etc/sysconfig/iptables**:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 8443 -j ACCEPT
```

SELinux settings

Typically on CentOS SELinux is enabled by default, which prevents the database access from Tomcat. To enable please execute the following:

```
# Enable Tomcat - DBMS connection in SELinux. The -P-parameter is for persistent
sudo setsebool tomcat_can_network_connect_db on -P
```

Check the installation

Before calling the grsadmin URL for the first time, a restart of the Tomcat application server might be necessary, to activate all configuration changes. Execute

```
sudo service tomcat8 restart (on Ubuntu 18.04)
sudo service tomcat9 restart (on Ubuntu 20.04)
sudo service tomcat restart  (on CentOS 7.x)
sudo systemctl restart tomcat (on CentOS 8.x)
```

Open a browser and call the following URL:

```
http://<ip_of_our_configserver>:8080/grsadmin
https://<ip_of_our_configserver>:8443/grsadmin
```

On the login page enter the credentials **admin** / **garderos** and click on **Login**.

Open the following pages:

- **Information -> Routerstatus**
- **Router Management -> Database**

If both pages are shown (the result *Nothing to display* is ok) and no error message occurs, the installation was successfully finished.

If any error occurs, check the Tomcat log files for the reason, mainly:

```
/var/log/tomcat<version>/catalina.out
/var/log/tomcat<version>/configserver.log
/var/log/tomcat<version>/grsadmin.log
```

Optional tasks

Admin user authentication

To access the administrative web pages a login is required. The user credentials are by default stored in the database, but optional a radius based authentication can be configured. It is possible to define user with different access rights on the page. Regardless to the fact that the userlevel can be set from 1 to 15, currently only three different access levels are implemented:

- Userlevel 15: Full permission for add/modify/delete routers, files and templates. Includes the right to add/modify/delete other administrators.
- Userlevel 10: Permission for adding/modifying routers, files and templates. No deletions possible.
- Userlevel 1: Only read access to the database and the routerstatus page.

Database authentication

The administrators using the admin web application are by default stored in the table 'adminusers' within the config database.

Description of the table:

Column	Description
adminuser	The username for login.
password	The password for login, stored as hashed value. The hash algorithm is executed when adding/modifying users within the GRSAdmin webapp.
name	*(optional) Name of the user.*
firstname	*(optional) Firstname of the user.*
userlevel	A value between **1** to **15** (see above).

Radius authentication

If required, the admin users can be authenticated by a radius server. The radius server connection is defined in *grsadmin/WEB-INF/web.xml*:

```
    <context-param>
        <param-name>radius-server</param-name>
        <param-value>10.10.10.135</param-value>
    </context-param>
    <context-param>
        <param-name>radius-secret</param-name>
        <param-value>geheim</param-value>
    </context-param>
    <context-param>
        <param-name>radius-port</param-name>
        <param-value>1812</param-value>
    </context-param>
    <context-param>
```

```
        <param-name>radius-method</param-name>
        <param-value>CHAP</param-value>
    </context-param>
```

Column	Description
radius-server	IP address of the radius server
radius-secret	Shared secret of the radius server
radius-port	Port to connect (default is 1812 if not set)
radius-method	Authentication method CHAP or PAP (defaults to CHAP if not configured)

If the radius server is configured, the user authentication is processed this way:

1. The user's credentials are checked against the radius server
2. If radius authentication fails, the authentication is done against the database
3. If the database authentication fails, the user can not log in

The database authentication can be deactivated completely, so the fallback is disabled then. Change the parameter *adminlogin_db* from *true* to *false* in *grsadmin/WEB-INF/web.xml*:

```
    <context-param>
        <param-name>adminlogin_db</param-name>
        <param-value>false</param-value>
    </context-param>
```

Userlevel set by optional radius attribute

Users authenticated by radius will have the userlevel = 15 (full permission) by default. This value can be overwritten if the radius server sends the radius attribute "Management-Privilege-Level" within the response, and the value is between 1 and 15.

Here is an example configuration for **freeradius** (file *users*):

```
adminrad  Cleartext-Password := "loginpw"
          Management-Privilege-Level = 10
```

logging webapp

If the optional *logging* webapp is deployed and configured, more details can be configured in **logging/WEB-INF/web.xml**:

```
<servlet>
   <servlet-name>logger</servlet-name>
   <servlet-class>com.garderos.grstools.LoggerServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
   <init-param>
       <param-name>log_to_db</param-name>
       <param-value>true</param-value>
   </init-param>

   <!-- optional: activate basic auth
      <init-param>
        <param-name>loguser</param-name>
        <param-value>logme</param-value>
```

```
      </init-param>
      <init-param>
        <param-name>logpasswd</param-name>
        <param-value>logmepass</param-value>
      </init-param>
   -->

   <!-- optional: file logging
      <init-param>
        <param-name>logpath</param-name>
        <param-value>/opt/configserver/log/grslog.txt</param-value>
      </init-param>
   -->

</servlet>
<servlet-mapping>
    <servlet-name>logger</servlet-name>
    <url-pattern>/logger</url-pattern>
</servlet-mapping>

 ...
```

Parameters:

- **log_to_db**: Has to be set to *true*. The data sent in the HTTP request is stored in the logging database (table *logdata*).
- **loguser**: Activate basic authentication for this servlet, the given username has to be sent in every logging request.
- **logpasswd**: If basic authentication is active, the given password has to be sent in every logging request.
- **logpath**: When set, the values are written to this file.

Log to database

If the database logging is activated, please check that the database and table to store the log data is present (the table is existent unless it was not deleted after the installatio procedure). If necessary adapt the columns to your needs. The columns **timestamp**, **sourceip** and **grsname** are mandatory.

```
CREATE TABLE `logdata` (
  `timestamp` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `sourceip` varchar(16) DEFAULT NULL,
  `grsname` varchar(32) DEFAULT NULL,
  `cputemp` varchar(32) DEFAULT NULL,
  `uptime` varchar(32) DEFAULT NULL,
  `dslspeedup` varchar(32) DEFAULT NULL,
  `dslspeeddown` varchar(32) DEFAULT NULL,
  `rxbytes` varchar(32) DEFAULT NULL,
  `txbytes` varchar(32) DEFAULT NULL,
  `wwan0signal` varchar(32) DEFAULT NULL,
  `wwan1signal` varchar(32) DEFAULT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

View log data within grsadmin webapp

The grsadmin webapp is prepared to display the received logging information. Be sure the settings in the logging webapp and the grsadmin webapp are matching. See *grsadmin/WEB-INF/web.xml*:

```
    <!-- standard db used for logging, path to logfile as configured in logging webapp -->
    <context-param>
        <param-name>logdb</param-name>
        <param-value>true</param-value>
    </context-param>
```

Log to file

It is also possible to write the received values to a plain text file. Values then are written in key = value format.

Logging script on GRS router

This is an example, how to collect some statistic data on a GRS router and send them to the web logger:

```
#!/bin/sh

logserver="https://config1.garderos.com:8443/logging/logger"

user=logme

pass=logmepass


#collect the data

hostname=$( hostname )

temp=$( cli sh sys temp | grep "CPU" | cut -d ":" -f 2 )

uptime=$( cli sh sys info | grep Uptime | cut -d ":" -f 2-10 )

rxbytes=$( ifconfig ppp0 | grep bytes | sed "s#\(.*RX bytes:\)\([0-9]*\) \(.*\)#\2#" )

txbytes=$( ifconfig ppp0 | grep bytes | sed "s#\(.*TX bytes:\)\([0-9]*\) \(.*\)#\2#" )

down=$( echo $modem | sed "s#\(.*Down Rate: \)\(.*\)\( Kbps Up.*\)#\2#" )

up=$( echo $modem | sed "s#\(.*Up Rate: \)\(.*\)\( Kbps SNR.*\)#\2#" )

curl -u $user:$pass --data-urlencode "grsname=$hostname" --data-urlencode "cputemp=$temp" --data-urlencode "uptime=$uptime" \

     --data-urlencode "dslspeedup=$up" --data-urlencode "dslspeeddown=$down" \

     --data-urlencode "rxbytes=$rxbytes" --data-urlencode "txbytes=$txbytes" $logserver
```

If the communication between logging server and GRS is secured by certificates (i.e. the Garderos Configuration Server requires a valid certificate for each request), these certificates have also used by *curl* to send the data. These are the relevant options:

- *--cacert*: Path to the CA certificate which signed the server's certificate. Can be ommitted by using the *-k* attribute.
- *--cert*: Path to the router certificate.
- *--key*: Path to the private key file.

A complete request might look like this:

```
curl --cacert /mnt/hd/files/ca.cer  --cert /mnt/hd/files/router.crt --key /mnt/hd/files/router.key  --data-urlencode "...." $logserver
```

Viewing log data is described in GRS Admin web application

Tomcat proxy support

When running the Garderos Configuration Server behind a proxy server, all requests to the GCS are coming from that proxy IP address, which makes the IP address information on the router status page or in the Tomcat log useless. But if the proxy server and the Tomcat instance are correctly configured, the originating IP address can be made visible.

The proxy server has to be configured, to insert the original IP address into an additional header. This header is typically called "X-Forwarded-For". As an example, for the often used NGINX proxy, the following line has to be added to the configuration:

```
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

If the header is inserted by the proxy server, the Configuration Server can read the value after a small configuration change:

- Add the *RemoteIpValve* to the host configuration. The line should be placed before the *AccessLogValve* configuration.
- Add the parameter *requestAttributesEnabled="true"* to the *AccessLogValve* configuration. This is necessary to make the original IP address also visible in Tomcat's access log file within the variables *%h* or *%a*.

The full example looks like this:

```
<Host name="localhost" appBase="webapps"
...
...
     <Valve className="org.apache.catalina.valves.RemoteIpValve" />

     <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
             prefix="localhost_access_log." suffix=".txt" '''requestAttributesEnabled="true"'''
             pattern="%a %l %u %t &amp;quot;%r&amp;quot; %s %b" />

...
...
```

After a restart of the Tomcat application server, the new configuration becomes active.

Change default web application

In the current installation, the managing web application is only available when being called with the full URL like **https://<your-configserver>:8443/grsadmin**. For a convenient usage define the grsadmin as the default web application where you're automatically redirected to, if typing only the server name, e.g. https://config1.garderos.com".

To do so, create a directory **ROOT** in the *webapps*-directory (*/var/lib/tomcat<version>/webapps/*) and create a file **index.html** with the following content:

```
<html>
  <head>
    <meta http-equiv="refresh" content=0;URL="https://<your-server-name-or-ip>:8443/grsadmin">
  </head>
  <body></body>
</html>
```

Change ports

By default the Tomcat servlet container is accessible on the ports 8080 (HTTP) or 8443 (HTTPS). If no other services like Apache web server are configured for the standard ports 80 and 443, those can be used by Tomcat.

This can be done by adding iptables forwarding rules:

CentOS 7

Add the following lines to your firewall configuration file, usually **/etc/sysconfig/iptables**:

```
# Add forwarding of https (443) to tomcat-https (8443) and 80 to 8080
*nat
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -d 213.61.82.123/32 -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A OUTPUT     -d 127.0.0.1/32     -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A OUTPUT     -d 213.61.82.123/32 -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A PREROUTING -d 213.61.82.123/32 -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
-A OUTPUT     -d 127.0.0.1/32     -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
-A OUTPUT     -d 213.61.82.123/32 -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
COMMIT
```

Troubleshooting

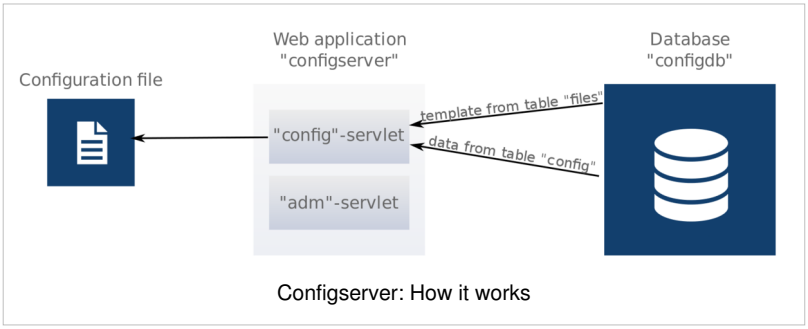
Since some errors are not visible immediately after installation but after a certain time, the look here for troubleshooting notes.

Configserver web application

Configserver web application

There are two different servlets in the configserver web application that can be configured:

1. **config**



Configserver: How it works

2. **adm**

They are described in the following chapters.

General settings

General settings for the whole configurator webapplication (*context-param*) can be set in **configserver/WEB-INF/web.xml** inside the <web-app> node:

```
<web-app .......>
    ....
    ....
    <context-param>
        <param-name>serverName</param-name>
        <param-value>configserver1</param-value>
    </context-param>
    ...
```

Optional settings for the configurator webapplication:

```
<web-app .......>
    ....
    <context-param>
        <param-name>forceFastReload</param-name>
        <param-value>66</param-value>
    </context-param>

    ...
```

List of available context-parameters

context-param	Description	Pattern / possible values	Default / example
serverName	A name which is used within the database GRS cache, used to assign the request to the corresponding server.	alphanumeric	configserver1
forceFastReload	If a GRS got an illegal configuration from the Configserver, it sends 'X-AC-Status: 400' in the following config request. If the Configserver detects status '400', it can overwrite the standard reconfigure interval (*configuration.reload-interval*) with the value given in 'forceFastReload'. By using this feature, the displayed status on router status page recovers faster from error state (400) to OK state (200).	numeric, 30-86400	0 (off)
dbCache	By default for each successful configuration request from a router some data are stored within the database table 'grscache'. This is very useful for a basic monitoring of all connected routers, especially when no external SNMP based monitoring system is used. If the configserver has to manage a high number of devices (> 10000) with a short reconfigure interval (< 1800 seconds) this feature might overload the database connection, so it can be turned off or the reconfigure interval has to be increased. The status page *Routerstatus* of the Garderos Admin GUI is only working when this value is not set to *false*.	true / false	true
filesInDB (deprecated)	By default all templates and files for download are stored within the database. This simplifies the setup when more than one server are required for delivering configurations (redundancy, load balancing) and makes data replication and backups easier. When set to *false*, then • Template files have to be put manually (by means of the operating system) to the directory **configserver/WEB-INF/templates** • Files to download have to be put to the directory **configserver/WEB-INF/files** • The template/file management pages of the Garderos Admin GUI are useless.	true / false	true

The config-servlet

Within the configserver web application the **config**-servlet is responsible for creating the configuration files and delivering them to the GRS devices. Its main tasks are:

- Check, if the incoming request comes from a Garderos GRS router and authenticate the device
- Search for the device in the database
- Load the requested template file from database
- Build the configuration and send it in the response

Servlet configuration

Minimum configuration

The servlet is configured in the file **configserver/WEB-INF/web.xml**. More instances of the servlet can be configured simultaneously, they only need different names and different paths.

The minimum configuration looks like this:

```
...
   <servlet>
      <servlet-name>config</servlet-name>
      <servlet-class>com.garderos.grs.config.Config</servlet-class>
   </servlet>
...
```

```
   <servlet-mapping>
       <servlet-name>config</servlet-name>
       <url-pattern>/config</url-pattern>
   </servlet-mapping>
...
```

With this configuration the servlet is accessible by the following URL:

```
http(s)://<configserver>:<port>/configserver/config
```

This URL must be configured as the configuration server URL in the startup configuration of each GRS router.
Example for GRS configuration:

```
configuration.remote.server.0.url=http(s)://<configserver>:<port>/configserver/config
```

The default behaviour of the config servlet:

1. Check if the request comes from a GRS router
2. Get the router name from the HTTP header and search for the router in the database column 'name'
3. If a router is found, proceed with step 6
4. If no router is found, get the serial number of the router from the HTTP header and search for this device in the column 'serial'. If a router is found, proceed with step 6
5. If again no device is found, return an *HTTP 404 not found* error
6. Read the shared secret from the matching database entry (column 'secret')
7. Authenticate the device by comparing the database secret with the received headers
8. If authentication is successful, check if the router requests a file or a configuration - either deliver file, or proceed
9. If the device requests a configuration, read the corresponding 'device_type' from the database result
10. Load the template file named *config.<device_type>*
11. If a template file is found, load the file and replace the dynamic fields with the values from the database
12. If the configuration file was successfully created, return the result to the GRS
13. Store the information of this request in the configuration server's cache.

Extended configuration options

By using the following optional configuration parameters it is possible to adapt the search order or database column names to the individual requirements. It is also possible to define more instances of the configuration servlet with different settings that can be accessed by different URLs.

Usage examples

1. Use an identifier different from *system.name* for searching the matching router in the database, e.g. serial number or IMSI.
   - Create database entries for all routers.
   - If not present, add column 'serial' or 'imsi' to the config table.
   - When installing a router, remember the serial number of the device or the IMSI of the inserted SIM card.
   - Store the serial number or IMSI in the database entry of the corresponding location.
   - The configserver now identifies the router by the serial instead of the name; the corresponding configuration will be delivered

- If a router must be replaced, replace the serial number in the database or just put the SIM card to the new router, the new device will receive the same configuration. It is not necessary to re-configure the replacement router.

2. GRS has two configuration parameters to handle configurations, one to modify the running configuration (often called *remote configuration*), and one to modify the startup configuration. Typically these configurations differ, so the configuration files to be delivered have to be based on different template files. To enable the Garderos Configuration Server to deliver different configuration files for the same router:
   - Configure 2 config-servlets and make them accessible with different names, e.g. *config* and *configStartup*
   - Create two different template files for each device type, e.g. *config.r7728* and *startup.r7728*
   - Link those templates to the corresponding paths by using the *template-pattern* option in *configserver/WEB-INF/web.xml*.
   - Configure the URLs of the servlets in the initial configuration of the GRS router, e.g.:

```
configuration.remote.server.0.url         = http(s)://<configserver-ip>:<port>/configserver/config
configuration.startup.server.0.update-url = http(s)://<configserver-ip>:<port>/configserver/configStartup
```

Available servlet configuration options

The configuration is done by modifying the *init-params* of the servlet in the file **configserver/WEB-INF/web.xml**.
Example:

```
...
   <servlet>
      <servlet-name>configStartup</servlet-name>
      <servlet-class>com.garderos.grs.config.Config</servlet-class>
       <init-param>
           <param-name>identifier</param-name>
           <param-value>name</param-value>
       </init-param>
       ...
       ...
   </servlet>
...
   <servlet-mapping>
       <servlet-name>configStartup</servlet-name>
       <url-pattern>/configStartup</url-pattern>
   </servlet-mapping>
...
```

These are the available servlet init parameters:

init-param	Description	Pattern / possible values	Default / example
identifier	For any incoming request for configuration, this identifier is picked from the received HTTP headers first for searching the device in the database.	name,mac,serial,imsi,ip	name
2nd-identifier	If the search in the database did not return a result, the second identifier is taken (optional).	none,name,mac,serial,imsi,ip	serial
db-key	Name of the column in the database, where the *identifier* is searched.	e.g. name,mac,serial,imsi,ip	name
2nd-db-key	Name of the column in the database, where the *2nd-identifier* is searched.	e.g. name,mac,serial,imsi,ip	serial
template-pattern	Modify the prefix of the template file, that the configuration file is based on.	alphanumeric string, e.g. config, startup	config
auth-type	Change the GRS authentication method: • standard: GRS devices are authenticated by comparing the router's shared secret with the secret in the database • none: authentication is deactivated • webxml: all incoming requests are authenticated against the password set in *auth-password*	none, standard, webxml	standard
auth-password	Alternative shared secret for authenticating a GRS device, used if authType = 'webxml'	alphanumeric string, special characters are allowed	s!cr3T
version-selector	Checks if different templates should be delivered depending on different GRS versions requesting the config. Format is *ddd_ddd:<suffix>* or *ddd_ddd_ddd:<suffix>*, where • ddd_ddd : Major and minor release numbers of the GRS version, like 003_005 or 003_006 • ddd_ddd_ddd: Major, minor and patch level numbers of the GRS version, like 003_005_026 or 003_005_046 • <suffix> : any string, used as extension which is appended to the template. **Notes:** • **Multiple version-selectors can be defined, separated by ',' (comma) or ' ' (space).** • **Selectors including patch-level are higher rated than ones without, e.g. when 003_005_026 and 003_005 are matching, the selector for 003_005_026 is used.**	• ddd_ddd:<suffix> or • ddd_ddd:<suffix>,ddd_dd2:<suffix2>	003_005:3five, 003_006:3six

Config database

The configserver fetches the data from a MySQL/MariaDB DBMS:

The database connection is configured in **/etc/tomcat*/server.xml**, see Configuration of configserver.

The following tables from the config-db are red by the configserver:

Table *config*

The table *config* stores data for each individual device. Mandatory columns in table *config* are:

- name
- secret
- device_type

The *name* must be a unique value, which allows to identify the router. By default, the *secret* is necessary to authenticate the router - only routers sending the correct secret will receive the configuration file. Once the requesting router is authenticated, the value of *device_type* is taken to select the corresponding template.

All other columns are optional and depend on the customer's needs. When one or more local interfaces must be configured at the routers, the following columns are suggested:

- client1IP
- client1DHCP

When managing a large number of routers, a description for each individual router makes the management easier, e.g.:

- location

If a router must be identified by the serial number instead of the name, a column for the serial can be inserted:

- serial

Table *property*

The table *property* stores values, which are valid for all GRS routers. Mandatory columns in the table *property* are:

- name
- value

Question:
Why do I need this extra *property* table for values that shall be delivered to all routers? It is easier to put these values directly into the template.

Answer:
This is correct, but there are two advantages using the *property* table:

- If you have many different templates for all of your routers, and a value should be valid for all of them (e.g. the *enable-password*), it is easier to change one value in the database than to change a value in many template files.

Important:
The content of table *property* is only loaded at server startup! When adding or changing properties, the new values will only be used after a server restart.

Database example

```
mysql> select * from config;
+---------+----------+-------------+--------------+-------------+------------------+--------------------------------+
| name    | secret   | device_type | serial       | location    | client1IP        | client1DHCP                    |
+---------+----------+-------------+--------------+-------------+------------------+--------------------------------+
| grs001  | garderos | r7700       | R77280000111 | Testlab     | 192.168.10.1     | 192.168.10.10-192.168.10.20    |
...
...


mysql> select * from property;
+-----------------+--------------------------+---------------------+
| name            | value                    | lastupdate          |
+-----------------+--------------------------+---------------------+
| enablePW        | ${enc2}0A357EF...        | 2020-02-06 16:29:00 |
```

Configuration templates

The delivered configurations are created based on template files.

Storage of template files

The templates files are stored **within the table *files* of the configserver's database**. In older versions of the configserver or if specially configured, the files are stored in the filesystem within webapps-directory */configserver/WEB-INF/templates/*.

The templates are always named **<template-prefix>.<type>[.<suffix>]**, where

- <template-prefix> (mandatory) is configurable (see above), the default is **config**
- *<type>* (mandatory) is the value taken from the database column *device-type*
- <suffix> (optional) is an extension which allows to deliver configurations based on different templates, depending on the firmware version of the requesting GRS. See *version-selector* for details.

Examples:

```
config.r7728
config.r7728.3five
startup.r7728
```

Content of template files

Templates are text files containing key/value pairs corresponding to the GRS configuration API. The values can either be static or dynamic.

Examples:

- static key/value pair:

```
system.name = garderos-router
```

- key with dynamic value

```
system.name = ${ name }
```

Dynamic values (template functions)

Dynamic values are marked by **${ var_name }**, where *var_name* is

- The name of a column of the table *config* (see database schema) or
- a *key* from the table *property* or
- a function which is defined in the configuration server.

Available functions:

The values of the variables used in this example correspond to the database example mentioned above.

Function	Explanation	Example input	Output
${ CFG_REQ_URL }	Full URL used by the client for the configuration request. Useful if the client shall download a file from the same server, e.g. when multiple configuration servers are connected to one database.	${ CFG_REQ_URL }	[1]
${ CFG_REQ_SRV }	The host and port part of the request URL, can be used for links to the same server.	${ CFG_REQ_SRV }	[2]
${ encode (method , value }	Take a cleartext password from a database column and send it already encoded to the GRS. Valid encodings are: • sha256	${ encode ("sha256" , password) }	{sha256}5O2GoQVbn$.lqvG1xievyJLrh4uaR9PeVBz7J/EoF4GdP9Vl5svq0

${ range (network, netmask) }	Calculate the network range associated to a class C network: The gateway IP, which is at the low end of the range, is not included in the return value. *network* and *netmask* are the corresponding column names in the config database.	${ range (client1IP , client1Mask) }	192.168.10.2-192.168.10.254
${ cnet (network , netmask) }	Similar to ${ range... }, but the netmask is added to the return value *network* and *netmask* are the corresponding column names in the config database.	${ cnet (client1IP , client1Mask) }	192.168.10.2-192.168.10.254/24
${ net (netmask) }	Calculate a netmask in CIDR notation (e.g. */24*) from a netmask in dotted decimal notation (e.g. *255.255.255.0*). *netmask* is the corresponding column name in the config database.	${ net (client1Mask) }	24

${ ipv6iid (parts , offset , dbcolumn) }	Calculate an IPv6 IID (host part of an IPv6 address) based on a value in the database. The base scenario is to create a uniqe IPv6 address based on a known router attribute, e.g. the serial number. • parts: Number of parts the value is split to (range: 1-6, e.g. 4) • offset: Number of characters which have to be skipped (e.g. 1, as the leading character of a serial number won't be converted)	${ ipv6iid (4,1,serial) }	792:890:912:34
${ add (base , value , dbcolumn) }	Add a number to the value of a database column. Calculation is either done in decimal or hexadecimal format. • base: Base for calculation: "hex" (for hexadecimal) or "dec" (for decimal) • value: The number to add. If number is in hex format AND contains characters (A-F), it has to be surrounded by "".	Some examples: • ${ add ("dec", 1 , network) } • ${ add ("hex", 1 , network) } • ${ add ("hex", "A" , network) }	If the filed *network* contains *1*, the result is: • 2 • 2 • B

Example:

The names of the variables used in this example correspond to the database example from above.

```
interface.eth.1.ip-assignment = static
interface.eth.1.ipv4 = ${ client1IP }
interface.eth.1.ipv6 = fd00::${ ipv6iid ( 6,1,serial ) }/64
interface.eth.1.name = eth1
service.dhcp.ipv4.subnet.1.interface-ref = eth1
service.dhcp.ipv4.subnet.1.option.dns.server.0.ipv4 = 10.10.10.2
service.dhcp.ipv4.subnet.1.option.domain = garderos.com
service.dhcp.ipv4.subnet.1.option.netmask = ${ client1Mask }
service.dhcp.ipv4.subnet.1.option.router = ${ client1IP }
service.dhcp.ipv4.subnet.1.range.0.definition = ${ range ( client1IP , client1Mask ) }
system.name = ${ name }
system.secret = garderos
user.account.0.level = 15
user.account.0.name = root
user.account.0.password = ${ enablePW }
user.enable.0.password = ${ encode ( "sha256" , enablePW ) }
```

This will result in the following configuration file:

```
interface.eth.1.ip-assignment = static
interface.eth.1.ipv4 = 192.168.10.1
interface.eth.1.ipv6 = fd00::792:890:912:34/64
interface.eth.1.name = eth1
service.dhcp.ipv4.subnet.1.interface-ref = eth1
service.dhcp.ipv4.subnet.1.option.dns.server.0.ipv4 = 10.10.10.2
service.dhcp.ipv4.subnet.1.option.domain = garderos.com
service.dhcp.ipv4.subnet.1.option.netmask = 255.255.255.0
service.dhcp.ipv4.subnet.1.option.router = 192.168.10.1
service.dhcp.ipv4.subnet.1.range.0.definition = 192.168.10.2-192.168.10.254
system.name = grs001
system.secret = garderos
user.account.0.level = 15
user.account.0.name = root
user.account.0.password = garderos
user.enable.0.password = {sha256}$5$O2GoQVbn$.lqvG1xievyJLrh4uaR9PeVBz7J/EoF4GdP9Vl5svq0
```

Merge templates

It is possible for large templates to split the template into two or more files and merge the parts by the **@include** instruction in the main template. This feature might be useful if you have large configuration blocks that are identical for all your routers, e.g. many ACL filter rules.

Example:

Content of **acls.txt**

```
acl.ipv4.input.0.action = ACCEPT
acl.ipv4.input.0.dest-ports = 22
acl.ipv4.input.0.protocol = tcp
acl.ipv4.input.0.source-network = 212.18.16.41/32
```

```
acl.ipv4.input.1.action = DROP
acl.ipv4.input.1.source-network = 0.0.0.0/0
acl.ipv4.snat.0.action = MASQUERADE
acl.ipv4.snat.0.source-network = ${ client1IP}/${ net ( client1Mask ) }
```

Content of config.r7700

```
@include acls.txt
system.name = grs001
system.secret = garderos
user.account.0.level = 15
user.account.0.name = root
user.account.0.password = garderos
user.enable.0.password = garderos
```

Resulting configuration:

```
acl.ipv4.input.0.action = ACCEPT
acl.ipv4.input.0.dest-ports = 22
acl.ipv4.input.0.protocol = tcp
acl.ipv4.input.0.source-network = 212.18.16.41/32
acl.ipv4.input.1.action = DROP
acl.ipv4.input.1.source-network = 0.0.0.0/0
acl.ipv4.snat.0.action = MASQUERADE
acl.ipv4.snat.0.source-network = 192.168.10.0/24
system.name = grs001
system.secret = garderos
user.account.0.level = 15
user.account.0.name = root
user.account.0.password = garderos
user.enable.0.password = garderos
```

As of February 2020 it is also possible to use database variables in @include-definitions, as well as combinations of text and variables. Examples:

- `@include ${ includefile }`
- `@include include.${ name }`
- `@include ${dbvar1}.${dbvar2}`
- `@include filepart.${dbvar1}.${dbvar2}`

In these cases first the variables are taken from the database fields and the filename is built, then the content of the matching file is inserted.

As of **February 2022** (release r953) it is possible to create **nested includes**, i.e. an include file can include other files.

Scheduled content

With time-based template content it is possible to activate or deactivate parts of the **delivered configuration file** at a certain time.

Important note:
This feature only controls the content which is delivered to the router by the Configuration Server, not the router itself. To activate the changes, the router has to download the updated/modified configuration within his regular reconfigure interval.
Example:

- The router's reconfigure-interval is 3600 seconds, the router usually loads the config at 08:38, 09:38, 10:38...
- The template in the server includes the configuration for activating WIFI only from 08:00 to 18:00

  →In this case WIFI will be activated at 8:38 and deactivated at 18:38

How to use:

- First a marker with a certain name, a start time and an end time has to be defined in the template file.
  Format: `#define <marker> <from-time> <to-time>`
  All parts are separated by single space. Any string is allowed for <marker>. Timestamps are can be in format
  - ***yyyy-MM-ddTHH:mm:ss*** for an absolute date/time
    Example: `#define T1 1970-01-01T00:00:00 2018-04-22T03:00:00`
    or
  - ***HH:mm*** for a day/night like definition
    Example: `#define night 21:00 06:00`
- Later in template these defined markers can be used for one or more individual configuration lines:
  `#if <marker> <config_line>`
  Example:
  `#if T1 system.version=003_004_022`
  or
  `#if night no interface.wlan`

Since January 2020, the inverted logic is also available via the #ifnot descriptor:

```
#ifnot <marker> <config_line>
```

Example:

```
#ifnot T1 system.version=003_005_026
```

or

```
#ifnot night @include wifi.txt
```

How it works:

- If the current time is within the range defined for a marker, the corresponding comments and markers are removed for any matching '#if'-line, '#ifnot' works the opposite way.
- If the current time is out of the range for a marker, the comments are kept in file
- Multiple markers with different ranges can be defined
- The "#if" or "#ifnot" statement has to be placed in front of each individual line which should be handled.
- To handle a block of configuration lines, use an "@include" statement which is handled by a marker

- Include-files can also contain definitions and markers.
- Markers are only valid within the scope of the file.

HINT Since September 2019 also database variables are allowed in the time definitions, e.g. `#define day ${ dayrange }`

The content of the field *dayrange* must be of course a valid time definition, like `08:00 20:00`

Example of a template file:

```
#define T1 1970-01-01T00:00:00 2018-05-01T03:00:00
#define T2 2018-05-01T03:00:01 9999-01-01T00:00:00
#define day ${ dayrange }                          #e.g. the content of variable 'dayrange' is '08:00 20:00'


#if T1 system.version=003_004_020
#if T1 system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_020.img
#if T1 @include acl.txt


#if T2 system.version=003_004_022
#if T2 system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_022.img
#if T2 @include acl_neu.txt
#if day @include wlan_on.txt
#ifnot day no interface.wlan
```

Database field (boolean) dependent content

- Introduced with Configserver / GRSAdmin release r784 (August 2019).
- Inverted values (#ifnot) introduced with release r819 (January 2020).

With database dependent template content it is possible to activate or deactivate parts of the configuration by a boolean (true=on/false=off) value in the database.

How to use:

- First a database column of type 'boolean' (or tinyint(1) for MySQL) has to be added to your config table:
  `ALTER TABLE config ADD COLUMN wifion boolean;`
  Note: By just adding the column, all values are set to *NULL* which is the same as *false*.
- Now this variable can be used in templates within an *#if-* or *#ifnot*-statement, similar to the time scheduled content:
  `#if ${ column_name } <config_line>`
  `#ifnot ${ column_name } <config_line>`
  Example:
  `#if ${ wifion } @include wifi.txt`
  `#ifnot ${ wifion } no interface.wlan`

To make the boolean values visible (and editable) in GRSAdmin, go to the *Settings*-menu of GRSAdmin and add the columns to the *Boolean Fields*, or when GRSAdmin is still configured via the *web.xml* file, add these lines to the configuration of *../grsadmin/WEB-INF/web.xml* (ideally after the context-parameter *modfields*):

```
<context-param>
    <param-name>modbooleans</param-name>
    <param-value>wifion</param-value>
</context-param>
```

How it works:

- The "#if" or "#ifnot" statement has to be placed in front of each individual line which should be handled.
- When evaluating a line starting with "#if" or "ifnot", the boolean value (true or false) for the requesting router is taken from the database
  - If the value is *true*, the comment (including the 'if' and the variable name) is removed from the line, so the line can be interpreted by the GRS
  - If the value is *false*, the line is not modified, and as it is starting with a *#*-character, it will be ignored by the GRS
  - For the "#ifnot"-statement, it behaves the exact opposite way.
- To handle a block of configuration lines, use an "@include" statement which is handled by a boolean variable.

```
#if ${ wifion } interface.wlan.0.hardware-ref=wifi0
#if ${ wifion } interface.wlan.0.ip-assignment=none
#if ${ wifion } interface.wlan.0.name=wlan0
#if ${ wifion } interface.wlan.0.ssid=test_wifi

#ifnot ${ wifion } no interface.wlan
```

For every router where the value of the column *wifion* is set to *true*, the configuration server will deliver the following content:

```
interface.wlan.0.hardware-ref=wifi0
interface.wlan.0.ip-assignment=none
interface.wlan.0.name=wlan0
interface.wlan.0.ssid=test_wifi
#ifnot $_{ wifion } no interface wlan
```

HINT **Please check your templates carefully for both values (true/false).**
The templates must result in legal configuration files in both cases, otherwise the GRS might show undesired behaviour!

GRS-header dependent content

- Introduced with Configserver / GRSAdmin release r947 (February 2022).

With GRS header dependent template content it is possible to activate or deactivate parts of the configuration by defining an operation based on values the GRS sends within each configuration request.

Example usage:

- Activate or deactivate parts of the configuration only, if the GRS version is higher than defined
- Activate WWAN configuration only, if an SIM card is inserted (X-AC-IMSI is set)

Definition:

```
#defineHeader <marker> <headername> <operator> <compare-value>
```

Where:

- marker: any string which can later be used as reference in #if-statement
- headername: Either the string *GRSVersion* (which extracts the version from the User-Agent) or defined GRS headers like *X-AC-Imsi* or *X-AC-Serial* (see GRS documentation for more details)
- operator: = or == (equals), > (greater than), < (less than), <> (not equals), >= or => (greater or equal than) , <= or =< (less or equal than)

- compare-value: any string which shall be compared with the header-value using the operator

How it works:

- For each request, the header value sent by the GRS is compared with the defined value using the given operator.
  The result is either *true* or *false*.
- The result is stored in the defined *marker*
- An "#if" or "#ifnot" statement using these marker has to be placed in front of each individual line which should be handled.
- When evaluating a line starting with "#if" or "#ifnot", the boolean value (true or false) of the operation result is taken.
  - If the value is *true*, the comment (including the 'if' and the variable name) is removed from the line, so the line can be interpreted by the GRS.
  - If the value is *false*, the line is not modified, and as it is starting with a *#*-character, it will be ignored by the GRS.
  - For the "#ifnot"-statement, it behaves the exact opposite way.
- To handle a block of configuration lines, use an "@include" statement which is handled by a boolean variable.

Examples:

```
#defineHeader isVer36 GRSVersion >= 003_006
#defineHeader min3_6_44 GRSVersion >= 003_006_044
#
#Use the old (GRS up to 3.5) or the new (from GRS 3.6) IP configuration style
#ifnot isVer36 interface.eth.0.ip-assignment=static
#ifnot isVer36 interface.eth.0.ipv4=10.0.0.1/24
#if isVer36 interface.eth.0.ip.static.0.ipv4=10.0.0.1/24

#if min3_6_44 interface.ppp.0.multi-user.0.password=123456
#if min3_6_44 interface.ppp.0.multi-user.0.username=user
```

HINT **Please check your templates carefully for possible values (true/false).**
The templates must result in legal configuration files in all cases, otherwise the GRS might reject the configuration or show undesired behavior!

Downloading files

Files to download to the GRS router are stored within the table **files** of the configserver's database. In older versions of the configserver or if specially configured, the files are stored in the filesystem within webapps-directory */configserver/WEB-INF/files/*

These files are for example:

- Firmware update files for GRS
- authorized_keys file for ssh login
- Certificate files
- Scripts

Downloads are only processed if the GRS is successfully authenticated at the configuration server.

Accessing the downloads is possible by the following URL:

http(s)://<configserver>:<port>/configserver/config?file=<filename_to_download>

Downloading proteced (secure) files

For certain files (e.g. router certificates and keys) it should be avoided, that the file can be downloaded by any GRS device. Therefore a new storage and methods were introduced in September 2019 for a secure file download. Files which are stored here are bound to a GRS name and can only be downloaded by the mentioned router. These files are stored within the table **securefiles** of the configserver's database.

These files are for example:

- Router certificate files
- Router private key files
- Any other allowed file type which shall be protected

Downloads are only processed if the GRS is successfully authenticated at the configuration server **and the router name found in database matches the given name for that file.**

Accessing the downloads is possible by the following URL:

```
http(s)://<configserver>:<port>/configserver/config?secfile=<filename_to_download>
```

Management functions - the *adm*-servlet

The configuration server optionally provides some features which are useful for GRS management and troubleshooting. These features are accessible by calling the *adm*-servlet, but only if they are enabled and configured. When using the GRSAdmin for managing routers, turning on these features is required as otherwise the GUI does not work properly and has limited functions.

Configuration

The adm-servlet can be (de)activated and configured by modifying the file **configserver/WEB-INF/web.xml**. The following example shows the default configuration after installation:

```
....
   <servlet>
      <servlet-name>adm</servlet-name>
      <servlet-class>com.garderos.grs.config.Admin</servlet-class>
      <init-param>
          <param-name>isSecured</param-name>
          <param-value>true</param-value>
      </init-param>
      <init-param>
          <param-name>allowedIP</param-name>
          <param-value>127.0.0.1,0:0:0:0:0:0:0:1</param-value>
      </init-param>
      <init-param>
          <param-name>showConfig</param-name>
          <param-value>true</param-value>
      </init-param>
   </servlet>
....
   <servlet-mapping>
      <servlet-name>adm</servlet-name>
      <url-pattern>/adm</url-pattern>
   </servlet-mapping>
```

```
....
```

Mandatory configuration

The *adm*-servlet is only activated, if at least the basic servlet configuration (servlet-name, servlet-class and servlet-mapping) is present in the web.xml configuration file (see the bold lines above).

Optional configuration (modify or disable access restrictions)

As the default configuration makes the adm-servlet with all features accessible from everywhere, this might be a security risk. To apply access restrictions there are three optional init-parameters:

```
...
          <param-name>isSecured</param-name>
          <param-value>true</param-value>
...
          <param-name>allowedIP</param-name>
          <param-value>127.0.0.1,0:0:0:0:0:0:0:1</param-value>
...
          <param-name>showConfig</param-name>
          <param-value>true</param-value>
...
```

- **isSecured**: Limits the access to the adm-servlet to defined source IP addresses.
  - *true*: access is limited to requests from the IP given in the parameter *allowedIP*, default is *127.0.0.1*
  - *false*: Requests are allowed from any IP address

    default is *true*
- **allowedIP**: If *isSecured* is set to *true*, the source IP addresses from where the requests are allowed can be set here. A comma separated list of IP addresses is possible. By default this IP is set to *localhost* (127.0.0.1 and IPv6 0:0:0:0:0:0:0:1)
- **showConfig**: Switch for showing complete configuration files by the *adm*-servlet
  - *true*: the adm-servlet returns the configuration files if the request is valid
  - *false*: the adm-servlet does not show any configuration files, only access to the GRS cache is possible

    default is *false*

Be aware of the potential security risk when enabling the *showConfig*-Feature

- The *adm*-servlet is designed as a debug- and management feature. It delivers the configuration file to every client (e.g. browser, wget) without authorization, whereas the GRS config-servlet only returns the configuration to authorized GRS devices.
- As your configuration file may contain sensitive data (your internal network structure, IP addresses, tunnel configuration and passwords....), this feature should only be activated, if the general *adm*-access is strictly limited or if the whole configuration server and GRS clients are in a protected network.

The GRS cache

Each request of a GRS device is stored in the configserver's database within the table **grscache**. For high-load scenarios (>20000 routers) this feature can be switched off by the *dbCache*-parameter, see above. This information is useful for monitoring the routers, e.g. just by checking if the routers fetched the configuration files within the configured reload interval.

Getting information for all routers from database

The page *Routertatus* of the GRSAdmin displays the information stored in the database within a table which can be sorted/filtered. When GUI access is not possible, the content can be directly read from the database:

```
select * from grscache [ where <column_name> = '<filter_value>' ];
```

The following values are stored in the database:

Column	Description	Released in Version
name	Name of router. Used as unique identifier	
serial	Serial number of router	
mac	MAC address of router (first ethernet interface)	
ip	IP address, where config request came from.	
lastaccess	Timestamp of latest configuration request	
laststatus	Value of X-AC-Status, indicates if previously downloaded configuration was correct (200) or not (400)	SVN r646
lastconfighash	Value of X-AC-Last-Config-Hash, the hash value of the previously downloaded configuration	SVN r743
lastmodified	Timstamp of the lastest modification of the router in database	
version	Current GRS version of the router	
servlet	Servlet path of the latest configuration request	
template	Template name which was used at the latest configuration request	
servername	Name of configserver which was called at the latest request.	
count	Number of configuration requests. Counter is reset every 24 hours.	
lastimsi	IMSI of currently inserted SIM card (if present)	SVN r751
configstatus	Hint, if the latest configuration request was the very first of the device (INIT), a request after a modification (Modified) or a normal request (Not-modified).	SVN r751

Getting information for all routers from RAM - if dbCache is turned off:

```
http(s)://<configserver>:<port>/configserver/adm?status
```

Calling this URL shows the latest successful configuration requests from GRS devices with the timestamp and number of requests sent to the server (this value is reset every 24 hours):

Example (some values are shortened to make the example more readable):

```
#R77511100001, name='grs001', mac='00:d0:...', ip='1.2.3.4',
at='2016-04...' version='grs-gwuz-armel/003_002_006, servlet='config',
template='config.r7728'~8
#R77289010105, name='grs002', mac='08:00:...', ip='1.2.3.5',
```

```
at='2016-04...' version='grs-gwuz-armel/003_001_012, servlet='config',
template='config.r7728.threetwo'~5
#R15510000111, name='grs003', mac='00:d0:...', ip='1.2.3.6',
at='2016-04...' version='grs-avila-armeb/003_000_050,
servlet='confStartup', template='startup.r1551' ~6
```

The result shows the following values:

- Serial number of the router
- System name of the router
- MAC-address of the WAN interface
- Public IP address
- Last access time
- Current GRS version of the router
- The called servlet
- The template file which the latest delivered configuration is based on
- Number of requests

As the internal sorting is done by the serial number of the device, there might be more than one entry for a system if the hardware has been replaced.

Getting information for single routers:

Calling this URL shows only the line for the device with the given serial number:

```
http(s)://<configserver>:<port>/configserver/adm?status=<serial>
```

The following request simply returns the last known IP address of the device with the given serial numer.

```
http(s)://<configserver>:<port>/configserver/adm?lookup=<serial>
```

The following request cleans the complete GRS cache. After that the list is empty and will be filled again with each configuration request from the GRS routers. Therefore it takes at least the time of the configured **system.wakeup-interval** until the list is completed with all active routers.

```
http(s)://<configserver>:<port>/configserver/adm?clear=all
```

The *showConfig*-feature

If this management feature is enabled (see configuration), it is possible to create requests with any client (browser, wget, curl) to return the same configuration file as a request from a GRS device would do. This is useful for:

- Checking the database, if the device is stored and if all necessary fields are filled
- Analyzing the templates
- Getting a valid configuration file, which can be put to a device by copy & paste

The function *[show] config* of GRSAdmin uses this feature.

Possible requests:

```
http(s)://<configserver>:<port>/configserver/adm?name=<value>
http(s)://<configserver>:<port>/configserver/adm?serial=<value>
http(s)://<configserver>:<port>/configserver/adm?mac=<value>
http(s)://<configserver>:<port>/configserver/adm?imsi=<value>
http(s)://<configserver>:<port>/configserver/adm?ip=<value>
http(s)://<configserver>:<port>/configserver/adm?<any_above_request>&template=<string>
```

```
http(s)://<configserver>:<port>/configserver/adm?<any_above_request>&suffix=<string>
http(s)://<configserver>:<port>/configserver/adm?<any_above_request>&template=<string>&suffix=<string>
```

Calling the configserver with one of the requests from above will search the given column (name, serial, ...) of the config-database for a device with the identifier *<value>*.
If the parameter *template* is not present, the standard-prefix *config* is used, otherwise the given template-prefix.
If the parameter *suffix* is present, the given suffix will be added to the template file name.

Examples:

Let's assume there is a router in the database, which has the name *grs001*, and the device-type is *r7700*.
The following request will return the complete configuration file based on the template file *configserver/WEB-INF/templates/config.r7700*.

```
http(s)://<configserver>:<port>/configserver/adm?name=grs001
```

The next request would return the file based on the template *configserver/WEB-INF/templates/remote.r7700*.

```
http(s)://<configserver>:<port>/configserver/adm?name=grs001&template=remote
```

This request would return the configuration based on the template file *configserver/WEB-INF/templates/config.r7700.three*.

```
http(s)://<configserver>:<port>/configserver/adm?name=grs001&suffix=three
```

Important note:
As this feature returns the complete configuration file (including e.g. tunnel-parameter, certificate-locations etc.) it might be a security risk to make it publicly available. Therefore this feature is switched off by default and must be turned on during configuration. When turning this feature on, make sure that your configuration server is not accessible from the public Internet or at least restrict access to this servlet to a single IP address.

Appendix

Internal functions for corresponding with GRSAdmin

Although often installed on the same server, the web applications *configserver* and *grsadmin* can be installed on different servers. Additionally one instance of GRSAdmin can access and display some information and log files of two or more different *configserver* instances, e.g. in a redundant or load balancing setup. These information are e.g.

- The configserver's log file
- System information (installed version, database access)

Keep this in mind, when designing your network and firewalls, that the server running GRSAdmin should reach the server running the *configserver* application.

Logging options for the configserver web application

The log of processing the incoming requests is written to the file *var/log/tomcat*/configserver.log*. The logfile is rotated daily, the previous files are renamed to contain the corresponding date, e.g. */var/log/tomcat*/configserver.2019-04-22.log*.

The behaviour of the logging (mainly the minimum log level) can be configured within the file *configserver/WEB-INF/classes/log4j2.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>

<Configuration status="WARN">

    <Properties>
        <Property name="logdir">${sys:catalina.base}/logs</Property>
        <Property name="layout">%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level [%logger{2}]: %msg%n</Property>
    </Properties>

    <Appenders>
      <Console name="Console">
          <PatternLayout pattern="${layout}"/>
      </Console>
      <RollingFile name="CONFIGSERVER"
                   fileName="${logdir}/configserver.log"
                   filePattern="${logdir}/configserver.%d{yyyy-MM-dd}.log">
          <PatternLayout pattern="${layout}"/>
          <Policies>
              <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
          </Policies>
      </RollingFile>
    </Appenders>

    <Loggers>
        <logger name="com" level="INFO" additivity="false">
            <AppenderRef ref="CONFIGSERVER" />
        </logger>
        <Root level="INFO">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>

</Configuration>
```

This are the possible log levels:

- OFF When no events will be logged
- FATAL When a severe error will prevent the application from continuing
- ERROR When an error in the application, possibly recoverable
- WARN When an event that might possible lead to an error
- INFO When an event for informational purposes
- DEBUG When a general debugging event required

- TRACE When a fine grained debug message, typically capturing the flow through the application
- ALL When all events should be logged

Setting the logging to a finer level may help finding errors or configuration mismatches, but leads to more server load and larger log files.

Directory structure of web application *configserver*

The web application has the following structure:

```
configserver
 |-- META-INF
 |     `-- MANIFEST.MF
 |
 |-- WEB-INF
 |     |-- classes ...
 |     |
 |     |-- files
 |     |       deprecated
 |     |
 |     |-- templates
 |     |       deprecated
 |     |
 |     `-- web.xml
 |
 |-- images
 |     |-- ...
 |
 |-- about.jsp
 |-- error.html
 |-- getServerLogJSON.jsp
 |-- getServerLog.jsp
 |-- sysInfoJSON.jsp
 `-- welcome.jsp
```

Sample configuration template

```
acl.ipv4.input.0.action=ACCEPT
acl.ipv4.input.0.connection.0.state=established
acl.ipv4.input.0.connection.1.state=related
acl.ipv4.input.0.protocol=tcp
acl.ipv4.input.0.source-network=213.62.82.123/32
acl.ipv4.input.1.action=ACCEPT
acl.ipv4.input.1.dest-ports=22
acl.ipv4.input.1.protocol=tcp
acl.ipv4.input.1.source-network=213.62.82.123/32
acl.ipv4.input.9.action=DROP
acl.ipv4.input.9.source-network=0.0.0.0/0
configuration.reload-interval=3600
configuration.remote.server.0.url=https://config1.garderos.com/configserver/config
```

```
interface.eth.2.description=PPPOE-ref
interface.eth.2.ip-assignment=none
interface.eth.2.name=eth2
interface.ppp.0.interface-ref=eth0
interface.ppp.0.ip-assignment=auto-ipv4
interface.ppp.0.name=ppp0
interface.ppp.0.password=${ dsl_pw }
interface.ppp.0.username=${ dsl_user }
service.ssh.enable=true
system.name=${ name }
system.secret={enc2}F082AB2D3D5D4637635D0E7F04980D1B41
system.timezone=Europe/Berlin
user.account.0.level=15
user.account.0.name=root
user.account.0.password={sha256}$5$SRK5cn6T$6z8k.kn06BWlQSnhzuJ5wzuMq8YZTJ0JQlO08IlN6vD
user.enable.0.password={sha256}$5$SRK5cn6T$6z8k.kn06BWlQSnhzuJ5wzuMq8YZTJ0JQlO08IlN6vD
```

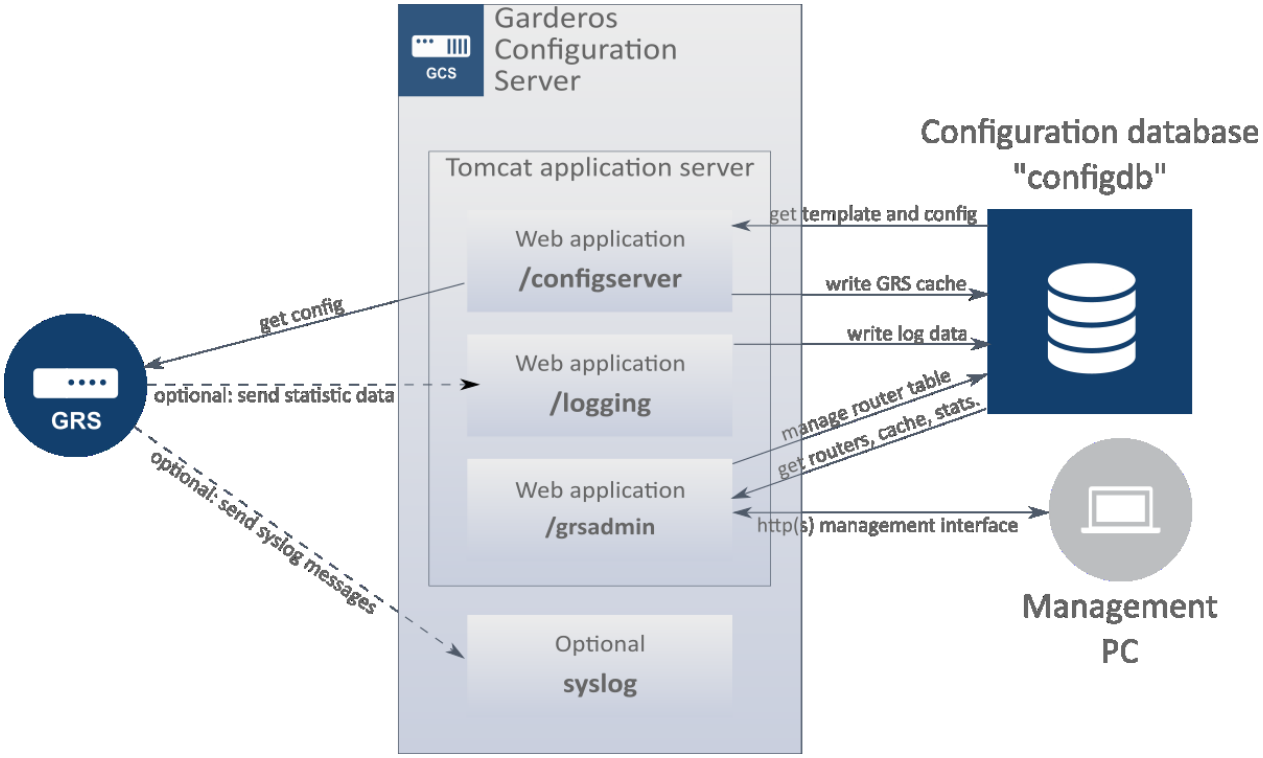
References

[1] https://config.garderos.com:8443/configserver/config
[2] https://config.garderos.com:8443/

GRS Admin web application

grsadmin is a web application, which can provide some or all of the following features:

- GRS router management (add, modify and delete routers from database)
- Show system status (show information about GRS routers that have connected to the configuration servlet)
- Collect statistics information from GRS routers and store them in a database or file
- Create and display statistic information from logged data
- Display information from syslog server



The installation and configuration of the grsadmin web application is described in Installation.

Working with grsadmin

Login

The application is available at the following URL:

http(s)://<configserver>:<port>/grsadmin

The application is protected, you need valid credentials for accessing it. After the standard installation procedure there are some users installed by default:

- **admin** / **garderos** (Userlevel: 15)
- **grsmanager** / **garderos** (Userlevel: 10)
- **grsreader** / **garderos** (Userlevel: 1)

User management

These users are stored within the database (table *adminusers*) and can be modified from the admin page itself.

User levels can reach from *1* (very limited access) to *15* (full permission).

The following table shows the existing assignment of user rights:

Action	Minimum level	grsreader	grsmanager	admin
Show router status	1	×	×	×
Show statistics	1	×	×	×
Show server logfile	5	-	×	×
Show syslog	5	-	×	×
Show routers in database	1	×	×	×
Create router in database	9	-	×	×
Modify router in database	9	-	×	×
Delete router from database	14	-	-	×
List templates	1	×	×	×
View templates	5	-	×	×
Download template	5	-	×	×
Create new template	14	-	-	×
Modify template	14	-	-	×
Delete template	14	-	-	×
List files	1	×	×	×
View file	5	-	×	×
Download file	5	-	×	×
Upload new file	14	-	-	×
Modify file (scripts only)	14	-	-	×
Delete file	14	-	-	×
List secure files	10	-	×	×
Download secure file	10	-	×	×
Upload new secure file	14	-	-	×
Delete secure file	14	-	-	×
CSV export	9	-	×	×
CSV import	14	-	-	×
Database export (dump)	14	-	-	×
User mangement (create/modify/delete/set password)	15	-	-	×
Change own password	1	×	×	×
GRSAdmin configuration via GUI (stored in database)	15	-	-	×

- **Create admin user:**
  Choose *User management -> Create adminUser* from the left menu, fill the form and then click on *save*.

- **Modify admin user**
  Choose *User management -> Modify adminUser* from the menu, the select the user to change from the dropdown and click on *search*. The following form allows to modify the user's data or delete the user from the database.
- **Change password**
  To change the password click on the username on the top right of the title bar, then click on *change password*.
  Users with privilege level < 15 only can change their own password.

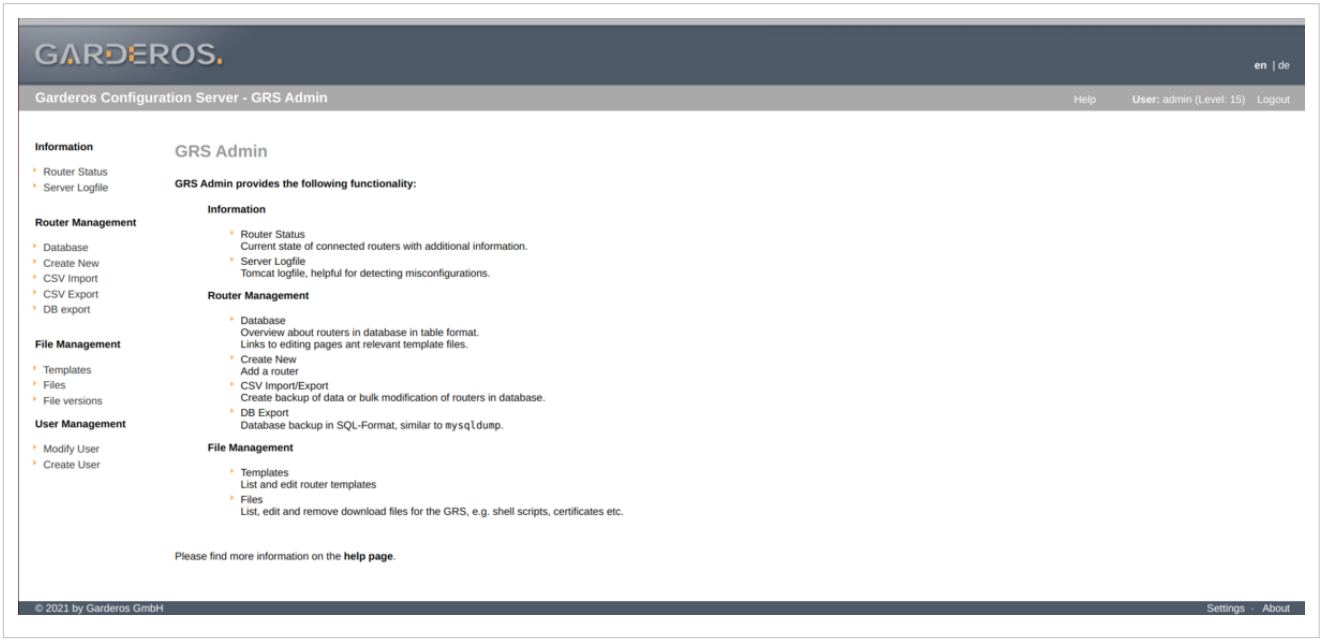
Alternatively it is possible to authenticate admin users against a radius server. Please see Configuration Server Installation -> Radius authentication for details.

Screenshots

The following screenshots give a short overview about the features of the grsadmin web application:

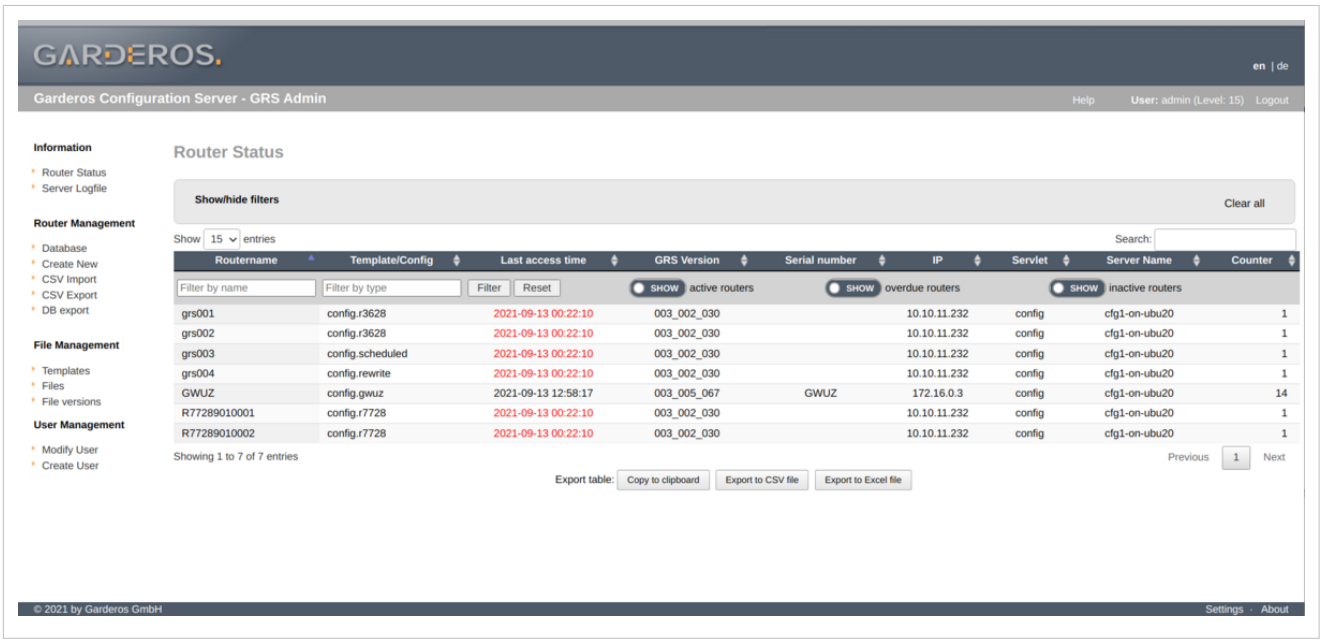
GRS Admin home

This is the start page after login. There are options to change the language on all pages, call a help page or to log out on the right side of the header.
The footer contains a link to the *About* page, where version information about the Garderos components are displayed.



Routerstatus

This page fetches the current status of routers that have connected to the web application *configserver*. This is done by displaying the data stored in the table *grscache* of the configuration database. If database caching is not enabled, the status page will not show the status details.



The database filter option allows to filter for routers either by the router name or the device type (input elements on top of the columns). In both cases it is a substring-filter, so e.g. a filter for device type *r77* will show all entries where the device_type contains the text *r77*, like *r7728*, *r7722*.

The page search (top right corner of the table) does a Javascript based 'live'-filtering of the result in the browser itself, without sending a new request to the server. The search string applies to all columns.

Additionally it is possibly to **hide/show rows** for

- Active routers (i.e. routers which are fetching the configuration regurarly)
- Overdue routers (routers which have been online, but did not connect to the configuration server within the wakeup interval, which is typically one hour)
- Inactive routers (routers which are stored to the database, but have never been active)

More features/filters for Router Status:

- It is possible to export the table content by special export buttons, either to clipboard or as CSV or Excel file (since revision r818 of GRSAdmin, January 2020).
- A direct link to the statistics page for each router is shown, if the router statistics are activated. The link-button is integrated in the column 'Routername'. If no statistics are collected for this router, an empty table is shown. (since revision r824 of GRSAdmin, February 2020).
- By clicking on *Show/hide filters* above the table, an extended filter panel can be shown (since revision r898, June 2021).

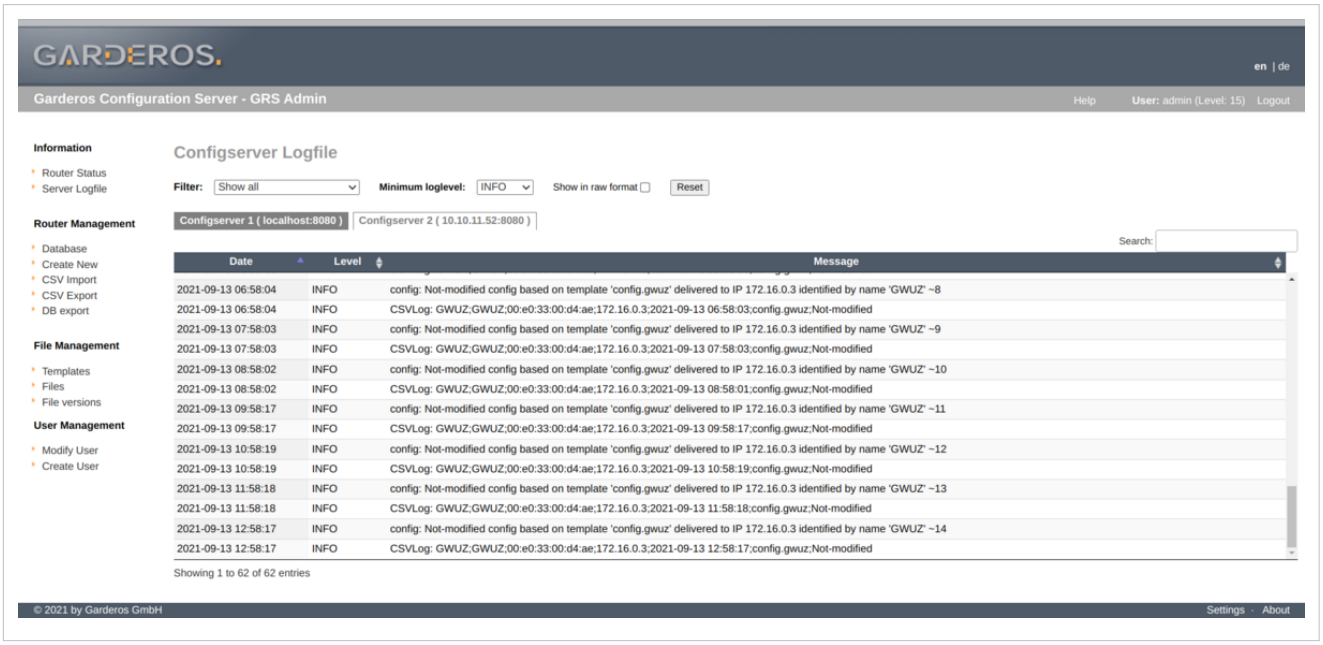
Server Logfile

This page displays the content of the configserver logfile (usually */var/log/tomcat/configserver.log*). As all requests from routers are logged in this file, this is helpful for troubleshooting, e.g. to find the reason, why a device does not get a configuration file.

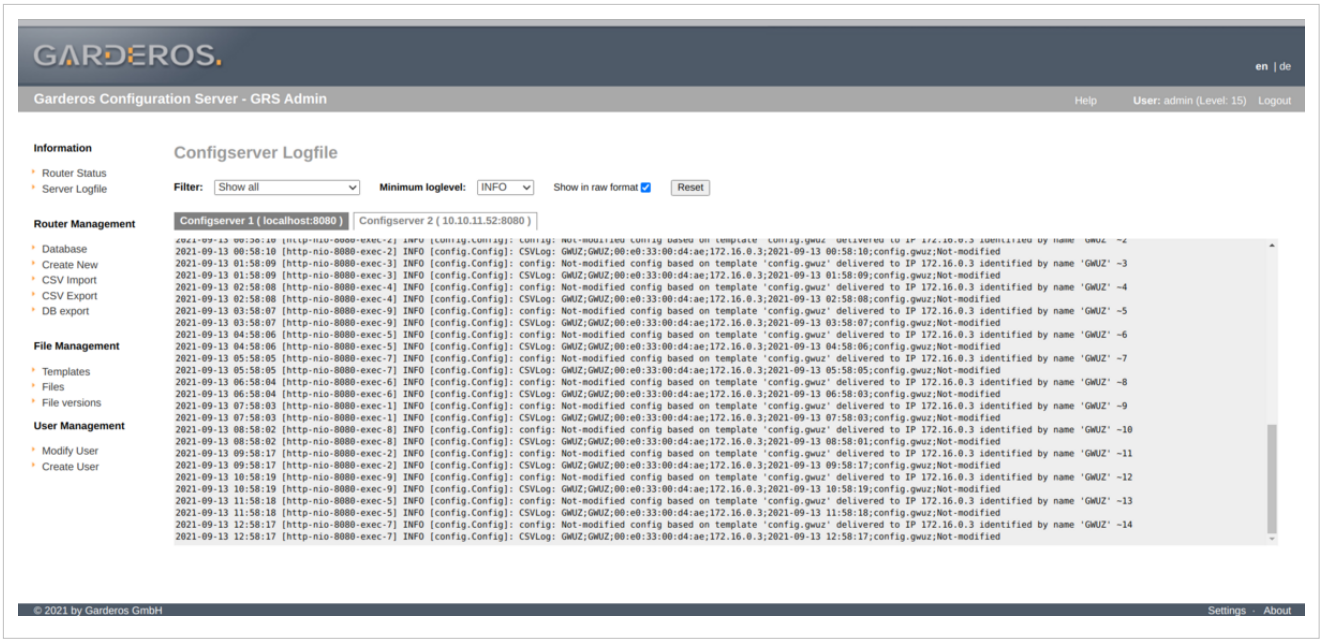
By default a not filtered table of log entries is shown; the following filtering options are available:

- Show entries where a configuration or file was successfully delivered (*Configuration/file delivered*).
- Show only entries where a router or a template was not found in the database (*No device/template found*).
- Show only *CSVLog* entries; these are lines where information about the requesting routers is collected, e.g. serial number, MAC address.
- Show only messages with at least a specified log-level (ALL - DEBUG - INFO - WARN - ERROR - FATAL).

/ If more than one configuration server is in use (e.g. for redundancy or load balancing reasons) it is possible to display the logfile of the other log server(s) on the same page, see configuration instructions.



It is also possible to display the data in 'raw' format as it is stored on disk.



Database

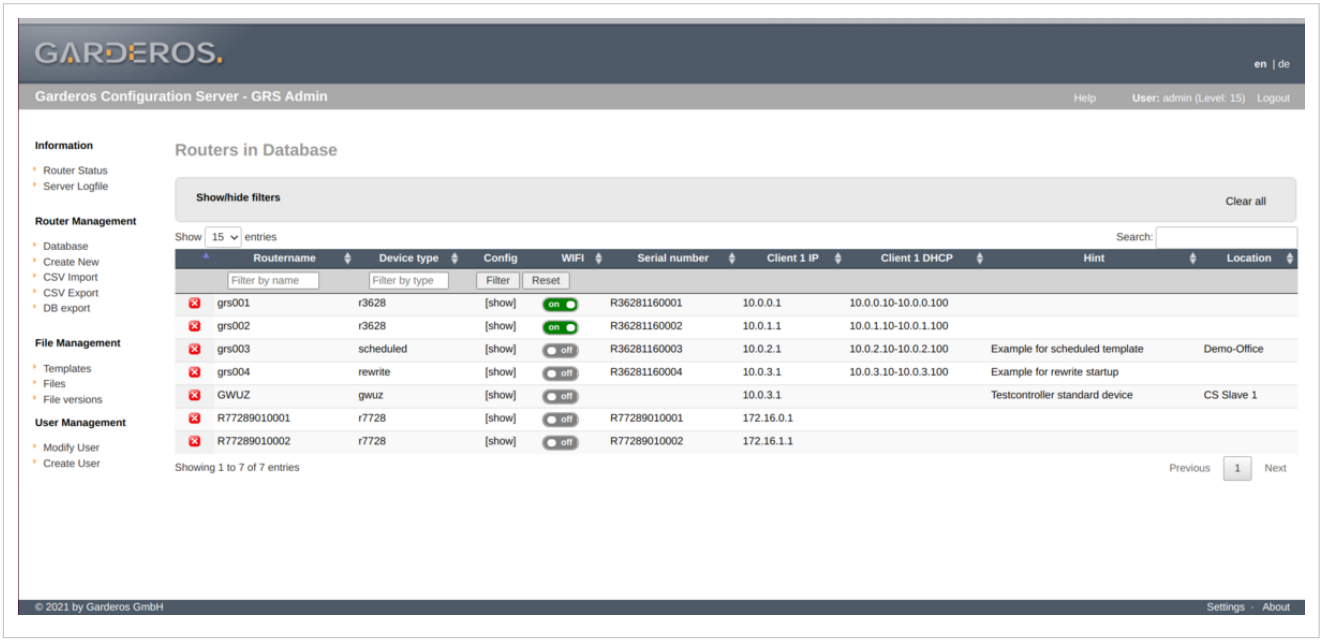
This page shows the content of the *config* database, so all router which are managed by this Configuration Server are listed here.

From this page you have the following options:

- Modify or copy database entries -> click on a router name in the second column
- Delete a router from the database -> click on the red button in the first column
- View the template file which is stored for this device -> click on the entry in the third column
- Show the configuration file as it will be delivered to the router -> click on *[show]* in the column *Config*

Additionally the most important settings of all routers are listed in the table, giving a quick overview over all routers.

The search and filter options are the same as for the page *Router Status*.



Router Management

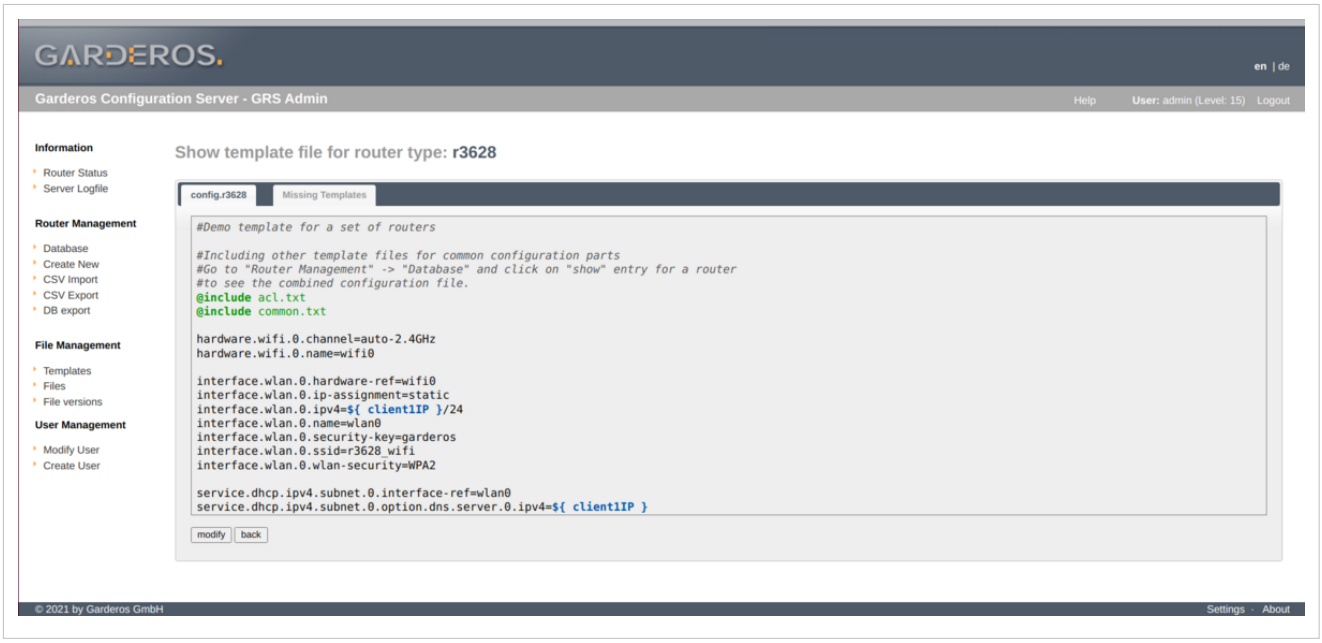
Create New or *Routers in Database -> click on name to edit router*

The *Create Router* page is shown when clicking on *Create New*, the form is empty then and can be filled from scratch. The page is also shown as *Modify Router* when clicking on the routername in the *Routers in Database* page, the form is then pre-filled with the current router settings and can be modified.

- With forward ('>>') and backward ('<<') buttons on top of the form it is possible to browse through the routers.
- Any change is saved to the database by clicking the *save* button.
- If the Routername is modified, the *save* button is invalidated and the *copy & new* button becomes active. In this case a new entry in the database is created with the currently shown values.
- The value for "Shared Secret" is hidden by default and can be made readable by clicking on the lock/unlock symbol within the form field. This behavior is configurable, see Configuration

Database -> show template

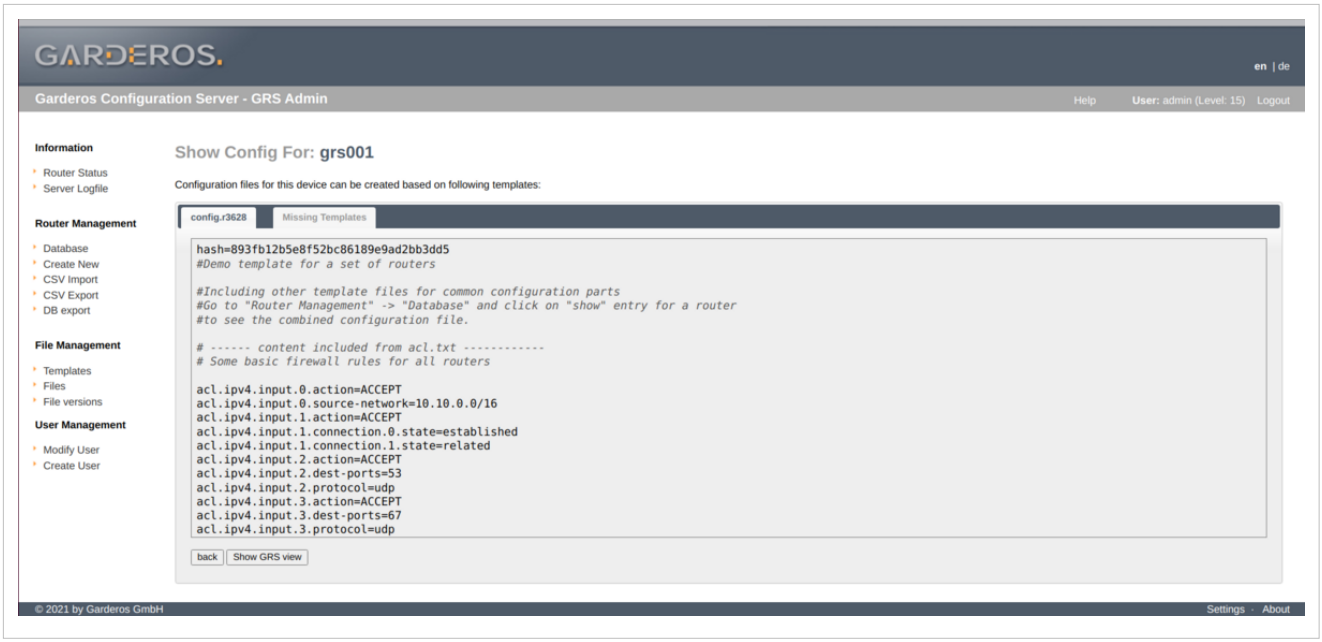
This page is shown when clicking on the template name in the *Database* page.



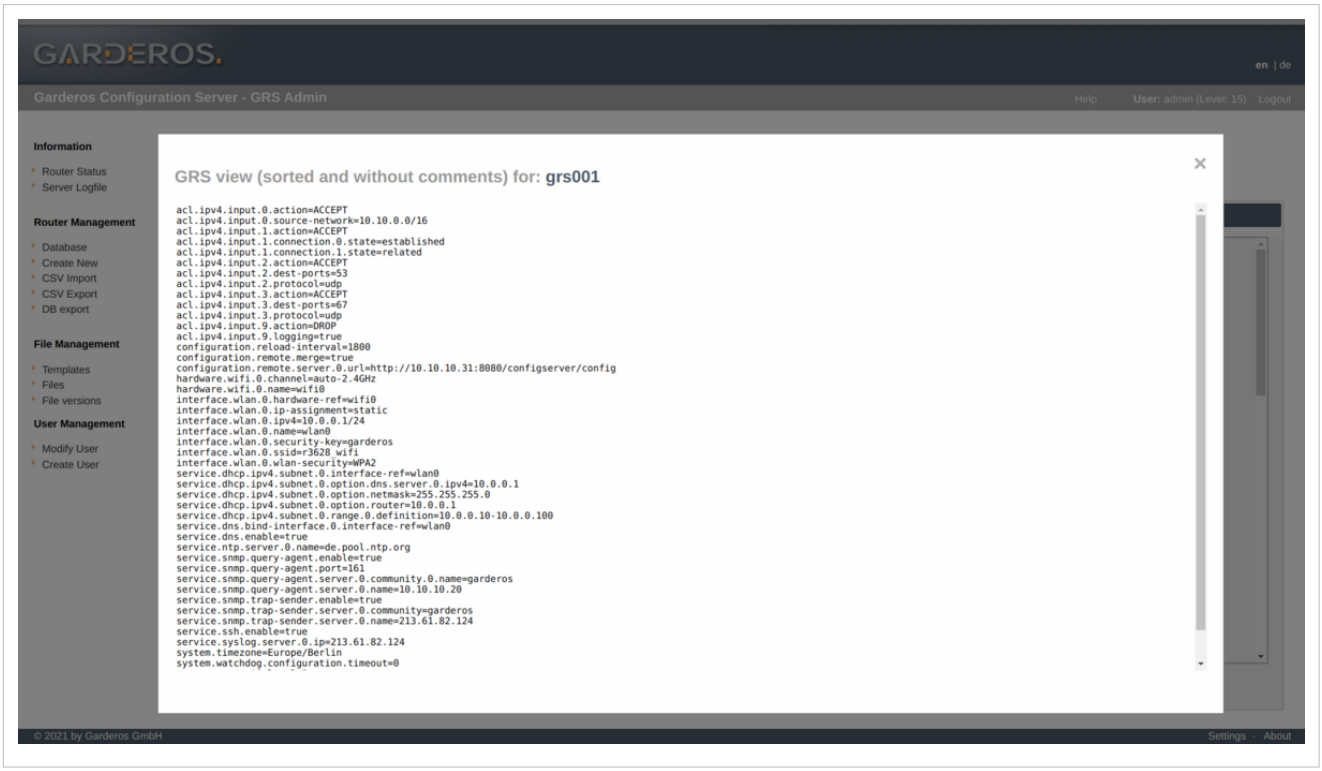
Database -> show configuration

This page is shown when clicking on the **[show]**-link in the column *Config* in the *Routers in database* page. This page displays the content of the configuration file as it would be delivered to that device in this moment.

- If the Configserver is configured to deliver different files for one device type (e.g. with different prefixes defined for different servlets or suffixes dependent on GRS versions), the possible created configuration files are shown in different tabs.
- The tab **Missing templates** shows a list of files which *may* be necessary due to the servlet configuration.
- Syntax highlighting is applied to the file content. Please take care if any line is marked in red (e.g. due to a missing value), because it is very likely that this configuration file might cause an error when delivered to the router.



The button *Show GRS view* below the text area brings up a new window where the file content is displayed 'clean' (without comments and empty lines) and sorted. This can help to identify e.g. lines which are defined twice within the template (they are marked in bold).



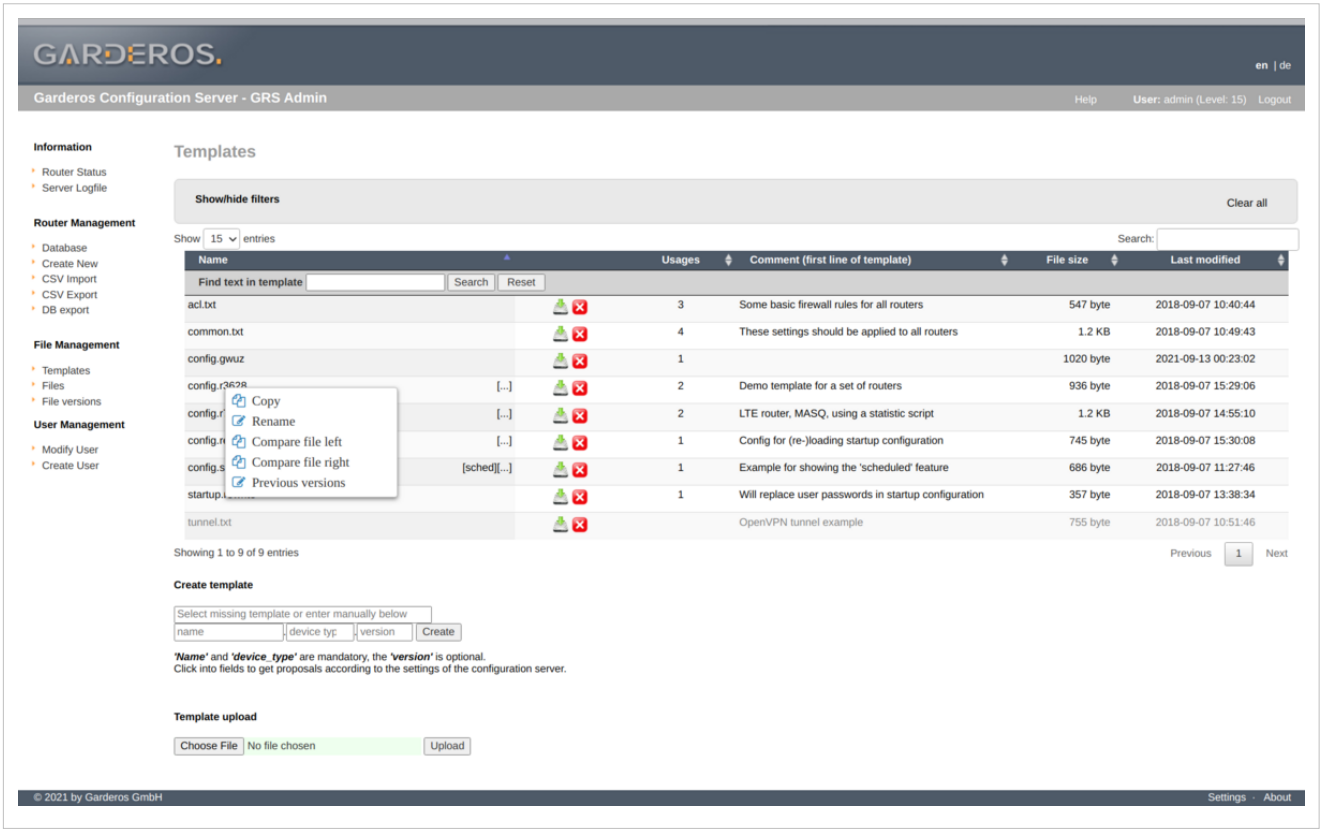
Templates

This page displays all *template*-entries within the files-table of the configuration database. It shows information about the usage of templates as well as the options for modifying or managing template files.

So called *include-files*, i.e. files which are included in other template files are also shown on that page.

Column description:

- **Name:** Shows the filename.
  - Click on the filename to edit the file.
  - Right-click on the filename offers more functions like copy, rename and compare template files.
  - If the marker *[...]* is present on the right side of the name column, it shows that this template includes other files. Hovering the marker shows a list of included files. If the marker is red, the at least one *include* refers a non-existent file.
  - If the marker *[sched]* is present on the right side of the name column, it shows that this template contains scheduled configuration, that means lines are delivered to routers only in a defined period. Hovering the marker shows the schedules and if they are currently valid (green) or not (gray).
- **Usages** shows:
  - For templates: How many routers are using this template. Clicking the number shows the correspondent routers in database.
  - For include-files: How many template files are referencing this include file. When hovering the number, a tooltip shows the corrspondent templates.
- **Comment:** If the first line of any template file contains a comment (line starts with '#'), the text is displayed in this column. Useful for giving hints on template usage or content.
- **File size**: Size of file in bytes or KBytes.
- **Last modified:** Date and time of last modification of that file.



Search and filter functions

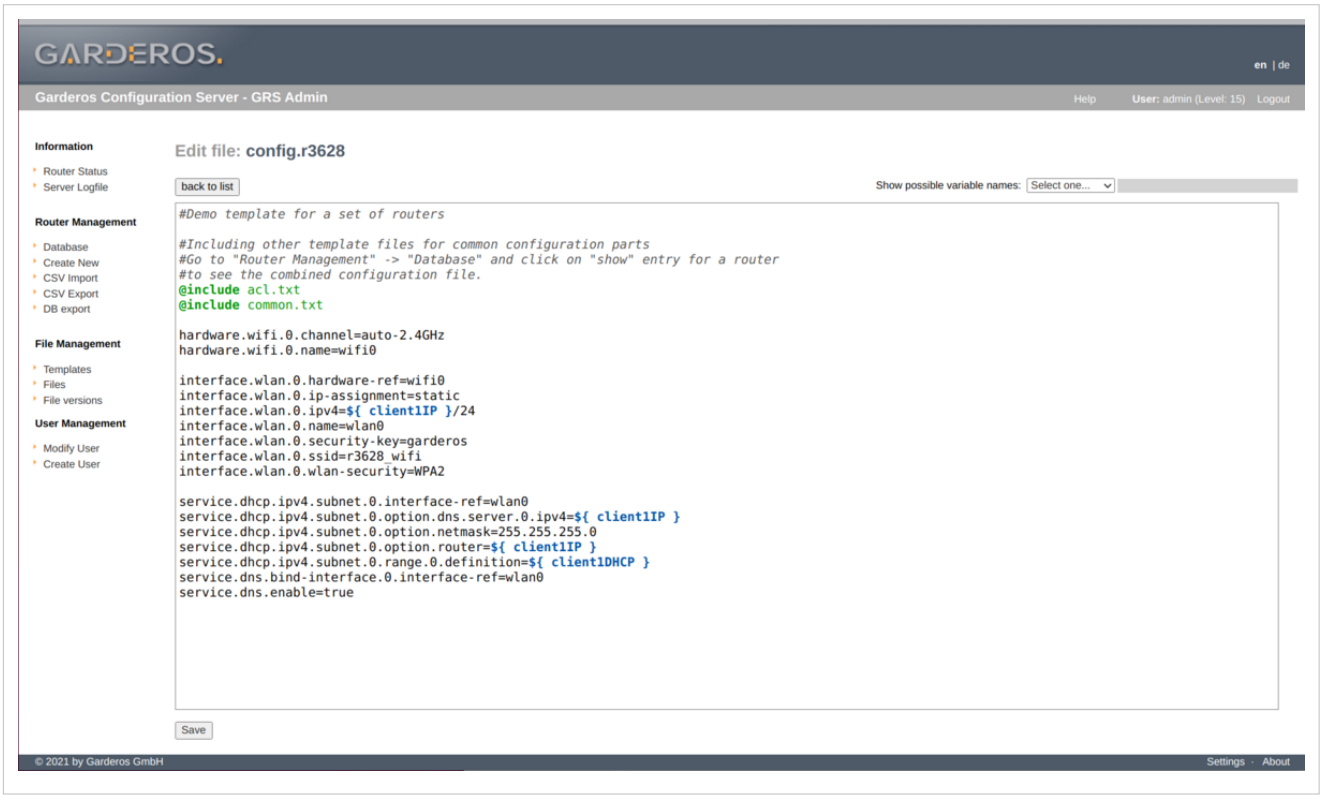
- The *Search* field on top right corner of the page performs a JavaScript based filtering of the displayed table. The search string reduces the table to show only lines where the string matches to any part, like the file name, the comment or even the last modified date. The filtering is performed locally in the browser without page reload.
- The form *Find text in template* performs a search in the file content of the template files. The request is sent to the server, which returns the matching files.

Edit an existing template file

After clicking a filename the **Edit file** page is shown.

The textarea supports syntax highlighting to give hints if the template file contains illegal lines:

- Standard config lines are written in black.
- If a lines is printed in red, either the parameter name is not valid or the value is missing.
  Note: The configserver does **not** check if parameters are valid for the aimed devices or if values are correct.
- Variable names (e.g. *${ name }*) are highlighted in bold blue. If the variable does not exist in the database, it is marked in red.
  To show the possible variable names click on the drop down list on the top right corner of the edit window. The variable name can directly be copied to the editor window.
- Comments (which will be ignored by GRS) are written in grey italic.
- Include files are printed in green.
- Scheduled content (definitions and usages) are also marked with different blue colors.

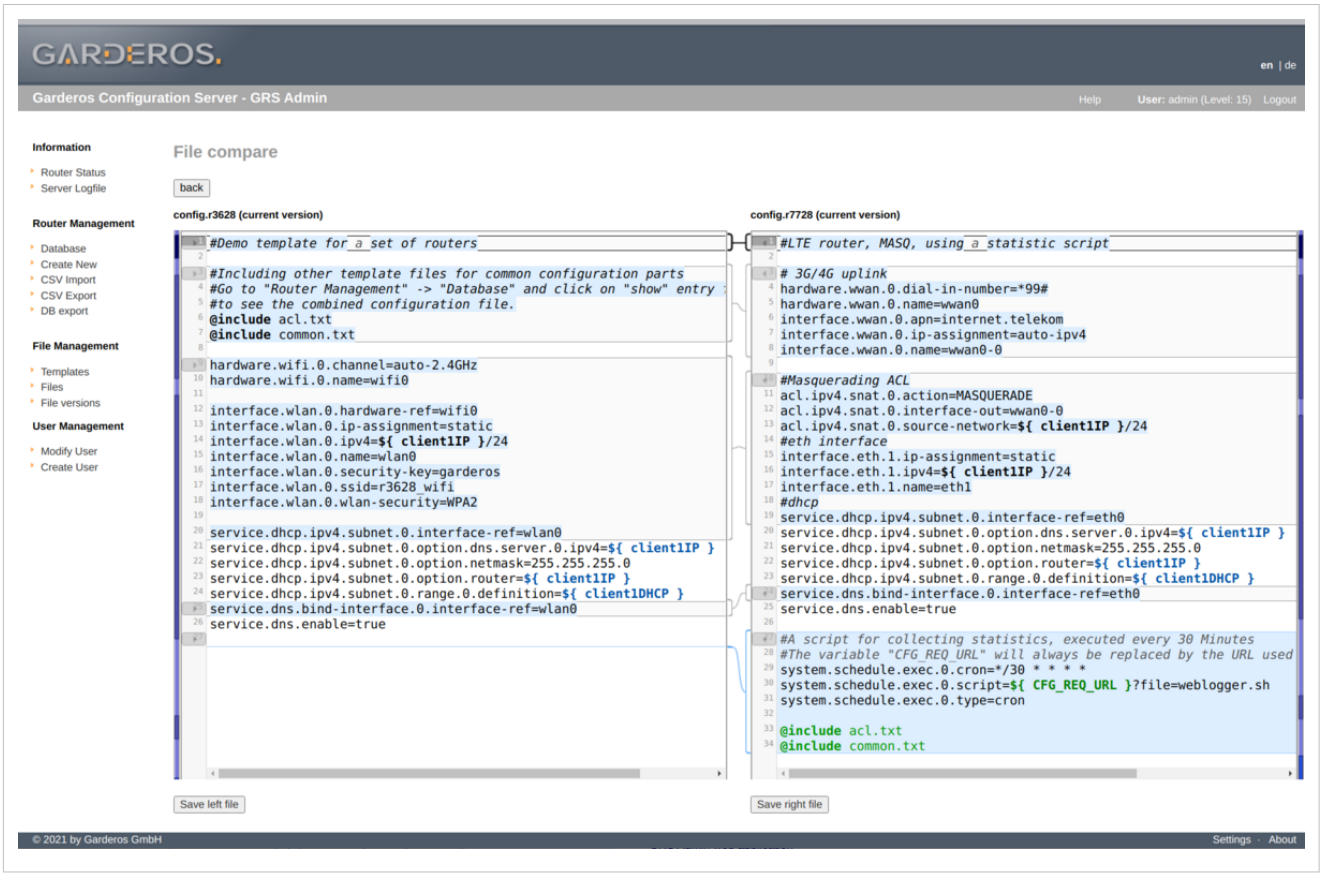


Compare template files

A new feature in GRSAdmin, released in Q1/2020 is a graphical file compare function for template files.
To compare two template files:

- Go to *File Management* → *Templates*
- Right-click on the first filename to compare and select *Compare file left* from the context menu
- Right-click on the second filename to compare and select *Compare file right* from the context menu

→ The *File compare* page is shown with the selected files.



Features:

- Files are shown side-by-side, differences are highlighted.
- With the arrows within the line numbers, a selected area can be merged to the right side or to the left side.
- Both text areas can be edited inline.
- Both text areas can be saved separately.

Create a new template file

To create a new template file, use the form *Create template* below the templates table.

The templates are always named `<template-prefix>.<type>[.<suffix>]`, where

- `<template-prefix>` (mandatory) is configurable (see template-pattern), the default value is *config*.
- `<type>` (mandatory) is the value taken from the database column device-type
- `<suffix>` (optional) is an extension which allows to deliver configurations based on different templates, depending on the firmware version of the requesting GRS. See version-selector for details.

You have two options to define the file name:

1. Enter the full filename in the first input field. The auto-completion (also accessible by clicking in the input field) makes file name proposals based on the current configuration and database content.
2. Enter the file name parts as described above in the designated input fields. Proposals based on the current configuration and database content are shown.

Alternative method: You can also create new (missing) template files by clicking the accordant link, e.g. on the tab *Missing templates* when viewing the existent template files.

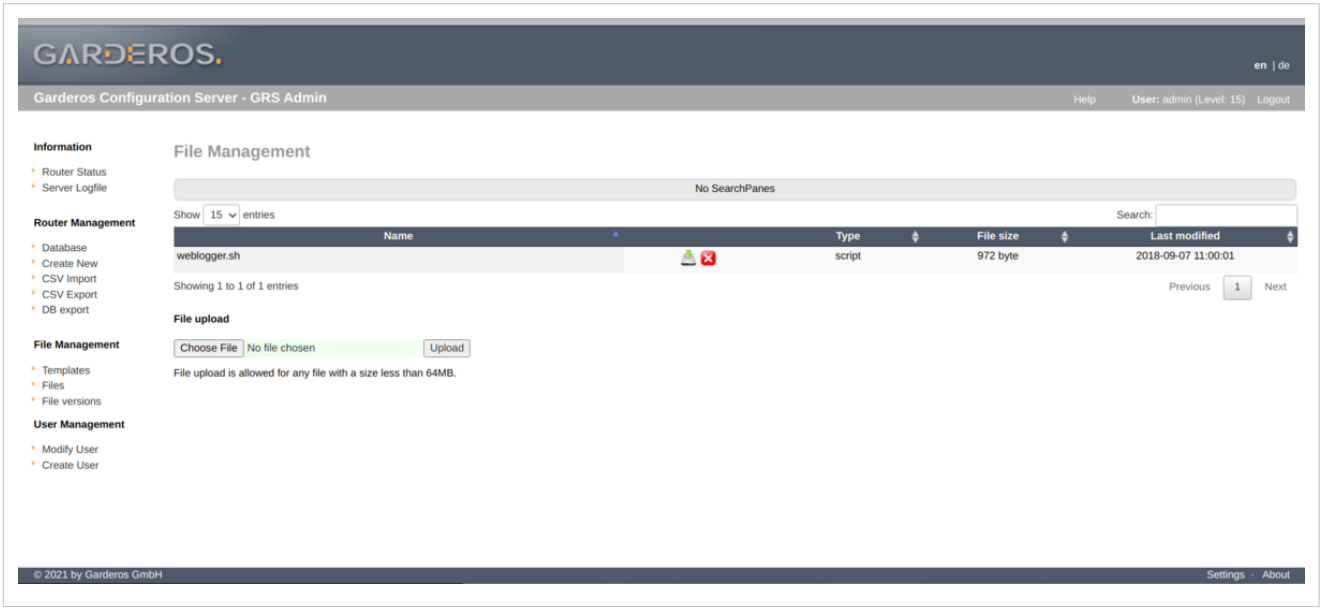
Example:

- Create a new router with a non-existent template name, e.g. *newtemplate*.
- View the router table. In the row of the new router click to the name *newtemplate* in the column *Template*.
- The next window shows a list of template files which should be created. This list depends on the configuration of your Configserver. Choose a file name from the list, e.g. *config.newtemplate*.
- The editor window appears. Enter your content and click on *Save*.

Files

This page displays the content of the *files*-table where the filetype is **not** "template", i.e. it is either a "grsimage", "script", "certificate", "key" or any other file type. It allows up- and download of files.

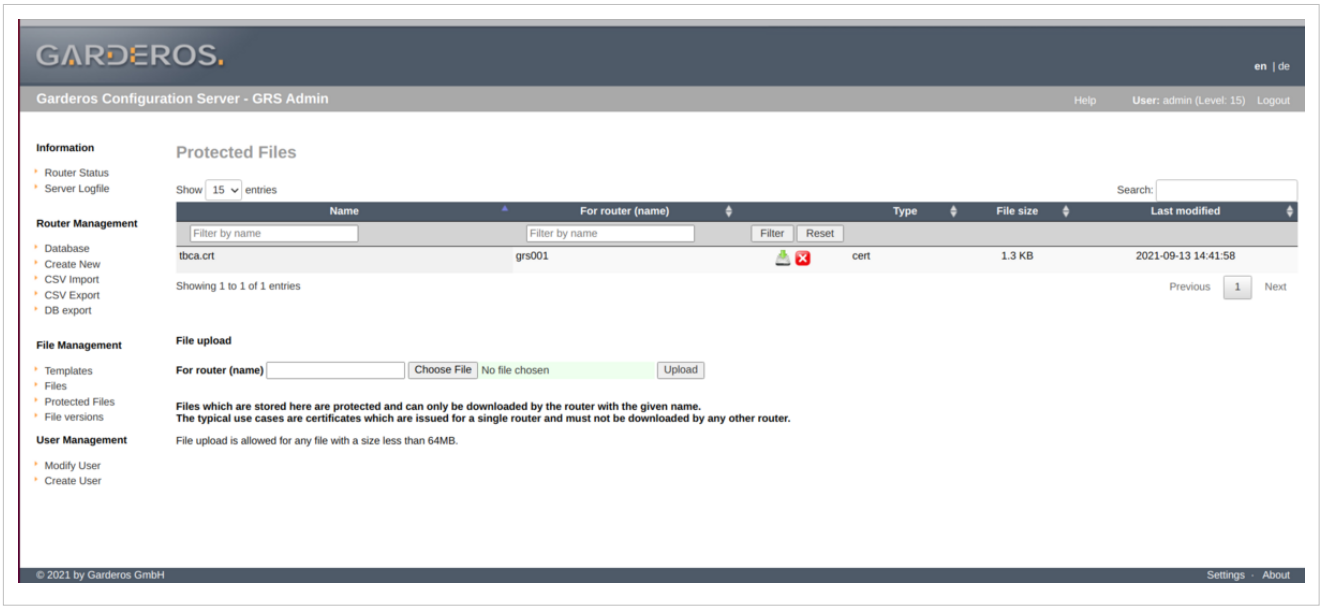
- Script files can be edited directly by clicking on the filename.
- When hovering with the mouse cursor over the filename, a file-link is displayed. This link can be used e.g. in template files.
- Buttons in the second column are for *download* (to your computer) or *delete* (from database).



Protected Files

This page displays the content of the *securefiles*-table and allows up- and download of files to that table. For these files a router name has to be defined, and any uploaded file can only be downloaded by that router with the defined name.

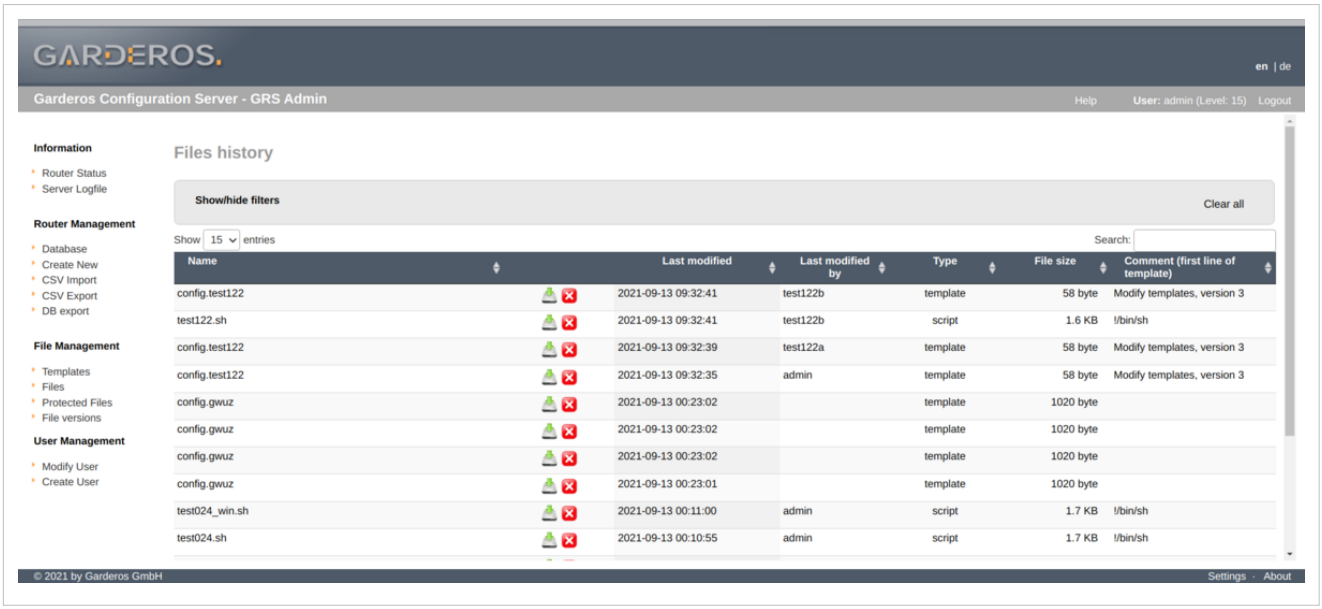
The intended use case of this protected file storage are certificates, which must not be accessible for other routers than the destination router.



Files history

If templates or files (not securefiles!) are modified or deleted, the previous version of that file is stored in a separate table *fileshistory*. The page *File versions* allows:

- Show overview of the history table, with modification date and user.
- Compare different versions of a file, or compare version with the current version (only if file is still existent).
- Filter by files, users or file types.



First steps

The following example will give you an idea about how the Garderos Configuration Server helps managing routers.

Create a template file

The following steps describe the proposed way to create the template files, add the routers to the database and how to check if everything is working as expected.

Usually there are already a few example routers in the database, as well as a template file.

To start from scratch we suggest to follow this HowTo. The order of the following steps is not mandatory.

Create a valid configuration file directly on the router

The best way to create a valid configuration file is, to write it directly on the GRS router using the CLI. In the CLI you have all the advantages of

- Tab extension
- In-line help
- Syntax check

If it is possible to write or commit the configuration to the GRS without getting an error message, it is a valid configuration. See the GRS Documentation [1] for details.

Convert the GRS configuration to a template file

For creating a template file from your working configuration:

- Go to the CLI console of the GRS and execute the command **show configuration running**.
- Copy all configuration lines to the clipboard.
- In the *grsadmin*-webapp go to *Templates* -> *Create template*: Enter the template name and click *Create*.
  The template almos always has the base name *config*, followed by the device type, e.g. *r7722* or *r3628*, so the filename is then *config.r7722* or *config.r3628*.
- Paste the clipboard content in the following window and click *Save*.
- When calling the *Templates* page again you see the new template file.

Until now this template file does not contain any dynamic value from the database, but it is already possible to deliver a configuration based on this template to a GRS router.

Router management

Create an entry for a router in the database

- In the *grsadmin* go to *Database -> Create new* and fill the form:
  - As *Routername* enter the name of the GRS router (as defined in the startup configuration).
  - As *Secret* enter the shared secret of the GRS router (as defined in the startup configuration).
  - As *Device type* enter the first extension of the newly created template file, e.g. *r7722* or *r3628*.
  - If any other field is necessary, fill it with pseudo-data (e.g. Client 1 IP = 1.2.3.4).
- Click *create*.
- Go to the page *Routers in Database* to see if the router was successfully created.

The first three fields are mandatory for every router. These are the columns **name**, **secret** and **device_type** in the database. All other fields are optional (but might be necessary due to the implementation of the web page).

Connect the GRS router to the configuration server

You can now connect the GRS router to the configuration server for the first time. Of course you need the (public) IP address or a DNS name of the machine where your configuration server is running, and this IP address must be reachable from the router.

- Log in to the router
- Call *configuration terminal startup*
  - Add the following configuration line: *configuration.remote.server.0.url = http://<your_ip>:<your_port>/configserver/config*
  - *Commit* the new configuration

The router now tries to get the configuration from the Configuration Server the first time. View the log file of the router to see if this works or if an error occurs.

It is possible to see the connection status of the router in the *grsadmin* application. Go to *Routerstatus*. The table shows one line for every router that is configured in the database. If a router successfully fetched a configuration file, the other columns for this router are filled. Each time a GRS router successfully connects to the configuration server, the columns are updated.

Customize the template

Until now there are no dynamic fields in the template. All routers configured to that template get the same configuration. To avoid this you have to identify the parameters in the configuration template, which must be unique for each GRS router.

If you have a typical interface configuration like this:

```
interface.eth.1.description=client-interface
interface.eth.1.ip-assignment=static
interface.eth.1.ipv4=10.10.0.1/24
interface.eth.1.name=eth1
service.dhcp.ipv4.subnet.0.interface-ref=eth1
service.dhcp.ipv4.subnet.0.option.dns.server.0.ipv4=10.10.0.1
service.dhcp.ipv4.subnet.0.option.netmask=255.255.255.0
service.dhcp.ipv4.subnet.0.option.router=10.10.0.1
service.dhcp.ipv4.subnet.0.range.0.definition=10.10.0.10-10.10.0.100
```

You probably need an individual IP configuration for each of the routers. In this case replace all values which must be dynamic by a variable in the following syntax:

```
${ <var_name> }
```

The identifier *<var_name>* between the brackets must correspond to a column name in the configuration database. You can either use the predefined columns of the sample database or just add a new column to the database.

The sample database has (among others) the following columns:

```
TABLE "config":
  name
```

```
secret
device_type
...
client1IP
client1DHCP
...
...
```

For this example we need an IP address which is used for the interface, the DHCP gateway IP and the DHCP DNS server IP. Additionally a value for the DHCP range is needed.

Modify the template as follows, to take those values from the database:

```
interface.eth.1.description=client-interface
interface.eth.1.ip-assignment=static
interface.eth.1.ipv4=${ client1IP }/24
interface.eth.1.name=eth1
service.dhcp.ipv4.subnet.0.interface-ref=eth1
service.dhcp.ipv4.subnet.0.option.dns.server.0.ipv4=${ client1IP }
service.dhcp.ipv4.subnet.0.option.netmask=255.255.255.0
service.dhcp.ipv4.subnet.0.option.router=${ client1IP }
service.dhcp.ipv4.subnet.0.range.0.definition=${ client1DHCP }
```

If the database contains values for these fields, they will be replaced by the configuration server in the moment when the GRS router asks for the configuration file.

Functions within templates

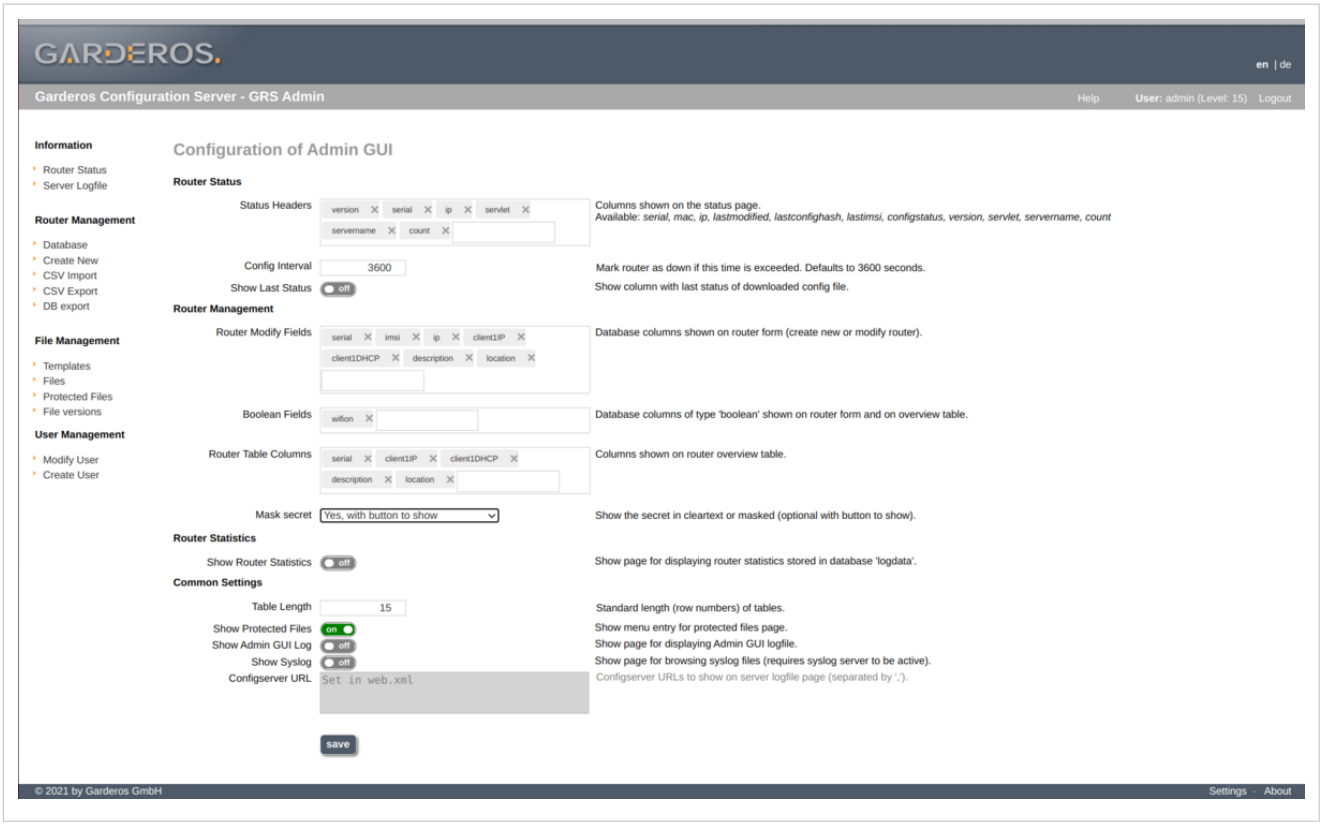
Built-in template functions might be helpful for writing templates, for example in a redundant configuration server setup or to automate network definitions and/or ip-addresses:

- **${ CFG_REQ_URL }**
- **${ CFG_REQ_SRV }**
- **${ encode (method , value }**
- **${ cnet(network,netmask) }**
- **${ range(network,netmask) }**
- **${ net(netmask) }**
- **${ ipv6iid (parts , offset , dbcolumn) }**

See Configuration Server: Configserver web application -> Dynamic values for details.

Configuration of GRSAdmin

From GRSAdmin versions released 2020 (release r810) and later, the settings of the GUI are adapted within the GUI itself by default (see the *Settings* page). These settings are stored in the database (table *webconfig*).



Some notes to GRSAdmin settings:

- To make all settings configurable within this page, remove or comment all *<context-param>*-entries from *grsadmin/WEB-INF/web.xml* file and restart Tomcat.
- All changes are applied immediately after clicking the *save* button.
- If you add columns to the router database, you can immediately make the columns available to the forms with these settings. But for adding meaningful form labels and descriptions to the input fields, still the files *grsadmin/WEB-INF/classes/forms.properties* and *forms_de.properties* have to be modified and Tomcat has to be restarted.

In case you do not want to use the inline configuration option, the settings can also be applied by editing the following file:

```
/var/lib/tomcat*/webapps/grsadmin/WEB-INF/web.xml
```

Each un-commented configuration option in this file will be set and overrule the inline-setting of the GUI.

Options **not** available in *web.xml* but only in GUI:

Parameter	Description	Pattern / possible values	Default / example
Table length	Set the default number of table rows displayed in the initial call of several pages (e.g. router status).	Any number	15
Mask secret	Introduced in r865, June 2020. Allows to override the new feature of masking the secret in the router form.	*Yes, with button to show* (Secret is masked, but can be show by clicking the unlock button), *Yes*, *No* (secret is not masked)	*Yes, with button to show*

The other names and possible values of the parameters are identical for the database and *web.xml*. Here is a full list of possible parameters:

context-param	Description	Pattern / possible values	Default / example
modfields	A comma separated list of fields which shall to be editable on routermod form.	Any column name from config table which type is NOT boolean.	serial,imsi,ip,client1IP,client1DHCP,description,location
modbooleans	A comma separated list of boolean values which shall to be editable on routermod form.	Any column name from config table which type IS boolean.	wifion
tableheaders	These fields are shown on page *Routers in Database*, which gives an overview about all configured routers.	Modbooleans are always shown, other fields like values of *modfields*.	serial,client1IP,client1DHCP,description,location
statusheaders	These columns are shown on page *Router Status*, which displays the current status of connected routers.	A comma separated list. Possible values are: *serial, mac, ip, version, servlet, template, servername, count, lastimsi, configstatus, lastconfighash, lastmodified*	version,serial,ip,servlet,servername,count
max_config_age	Max config age in seconds. This value should match the parameter *configuration.reload-interval* in your templates. If a router did not fetch config for more than this seconds, the latest timestamp is shown in red on the page *Router Status*.	3600	Default: *3600*
showlaststatus	Routers with GRS > 3.4 will send a value "X-AC-Status" which indicates, if the last received configuration was OK (Status=200) or not OK (Status=400)	true/false	Default: *false*
logdb	Shall link to statistics page be displayed in menu.	true/false	Default: *false*

dbshowcolumns	If statistics database ('logdata') is used, which columns shall be displayed on overview page.	Any column name from table 'logdata' in database 'logdata'	uptime,cputemp,wwansignal,wwannetwork,wwanband
dbshowgraphs	If statistics database ('logdata') is used, which columns shall be used to draw graphs.	Any column name from table 'logdata' in database 'logdata' which contains numeric data.	cputemp,wwansignal
showsecurefiles	Show menu entry for *Protected Files*.	true/false	Default: *true*
configserver_url	It is possible to add more than one configserver_url as comma-separated list. **This value is still by default kept in web.xml, not in the database**.	A comma-separated list of URLs where another configserver instance is running. Example: `http://127.0.0.1:8080/configserver/, http://10.10.11.12:8080/configserver/`	The default is: `http://localhost:8080/configserver/`
adminlog	Show the menu entry for displaying Admin GUI log files.	true/false	Default: *false*
syslog	Show the menu entry for displaying syslog files. Only useful, if a syslog server is running and collecting GRS syslog data on the same server hardware. For demonstration purpose only.	true/false	Default: *false*
syslog_dir	Only necessary when *syslog* is set to *true*	The directory where syslog files are stored. Example: */var/log/grssyslog/*	

Customization of database and grsadmin webapp

The originally delivered database and *grsadmin* webapp are prepared for a basic configuration with one local interface and one DHCP subnet. As the Garderos Configuration Server is very flexible, it is quite easy to adapt the database and the *grsadmin* webapp to any customized setup, where e.g. more client devices, IPSec- or OpenVPN-tunnels, etc. are needed.

Alter the database

When more dynamic values shall be taken from the database (e.g. a second client interface, individual wireless settings, etc.), add the corresponding columns to the database first.

To modify the table log in to the database and call the appropriate MySQL command. Example:

```
# log in to database 'config':
mysql -u <db_user> -p config

#add column 'test' to table 'config'
ALTER TABLE config ADD COLUMN `test` varchar(100);

#remove column 'test' from table 'config'
```

```
ALTER TABLE config DROP 'test';

#add columns storing booleans (true/false) which can be used as on/off switch in templates
ALTER TABLE config ADD COLUMN `wifion` boolean;
```

For more details refer to the MySQL documentation ([2]). Alternatively any web based database management tool like *phpMyAdmin* or *adminer* can be used.

If new columns are added to the configuration database, they are immediately available to be used in the template files.

Example:

- A new column *client2IP* was added to the table *config* of the configuration database.
- The column was filled by using the same tools as for editing the database, e.g. the command line.
- When adding the variable **${ client2IP }** to any template file, the configserver web application will immediately deliver the corresponding value from the database.

Adapt *grsadmin* to the modified database

If a column is added or deleted from the database, the *grsadmin* web application must be modified:

- When adding or modifying routers, this database column must be editable.
- The column should be shown in the database overview.

No knowledge about HTML or JSP/Java programming is required to apply these changes. Changes can be made within the GUI or are made by editing text files (the 'old' way).

Add a database column to the modify page

Via GUI (if not part of web.xml):

- Log in to the GUI with admin privileges (min. level=15).
- Call the *Settings* page (the link is at the right side of the page footer).
- Edit the list *Router Modify Fields*, add or remove values.
- Click the *save* button.

Via web.xml file: Edit this file:

```
grsadmin/WEB-INF/web.xml
```

Search the following lines:

```
  <!-- these fields can be edited on routermod form -->
  <!-- the attributes name, secret and device_type are mandatory and not part of this list -->
  <context-param>
      <param-name>modfields</param-name>
      <param-value>serial,mac,imsi,ip,client1IP,client1DHCP,description,location</param-value>
  </context-param>
```

Add the name of the new column to the line containing *<param-value>*, e.g.

```
    <param-value>serial,mac,imsi,ip,client1IP,client1DHCP,client2IP,description,location</param-value>
```

The order in the comma-separated list corresponds to the order in the modify page.

Add or remove database columns on the router table page

To keep the router table clear the columns to be shown can be defined separately from the columns to be edited.

Via GUI (if not part of web.xml):

- Log in to the GUI with admin privileges (min. level=15).
- Call the *Settings* page (the link is at the right side of the page footer).
- Edit the list *Router Table Columns*, add or remove values.
- Click the *save* button.

Via web.xml file:

Edit this file:

```
grsadmin/WEB-INF/web.xml
```

Search the following lines:

```
<!-- these fields are shown on page routers in database, which gives an overview about all configured routers-->
<!-- not all fields which can be edited are necessary on the overview page -->
<context-param>
    <param-name>tableheaders</param-name>
    <param-value>serial,client1IP,client1DHCP,description,location</param-value>
</context-param>
```

Add the name of the new column to the line containing *<param-value>*, e.g.

```
     <param-value>serial,client1IP,client1DHCP,client2IP,description,location</param-value>
```

The order in the comma-separated list corresponds to the order in the modify page.

Add a boolean database column to the router table and the modify page

From Version r784 (August 2019) of GRSAdmin it is possible, to store and edit boolean values in the database. As they should not be handled the same way as the standard text columns, a new configuration parameter is introduced:

Via GUI (if not part of web.xml):

- Log in to the GUI with admin privileges (min. level=15).
- Call the *Settings* page (the link is at the right side of the page footer).
- Edit the list *Boolean Fields*, add or remove values.
- Click the *save* button.

Via web.xml file:

Edit this file:

```
grsadmin/WEB-INF/web.xml
```

Search the following lines:

```
<context-param>
    <param-name>modbooleans</param-name>
    <param-value>wifion</param-value>
</context-param>
```

Add the name(s) of your boolean columns to the *param-value*.

Create meaningful text modules

To avoid that placeholders (like *??? client2IP ???*) are shown as the description for the new form field, the suitable text modules must be added to the language files of the web application.
Edit this file for the english version:

```
grsadmin/WEB-INF/classes/forms.properties
```

Add the following lines:

```
client2IP=Client 2 IP
client2IPInfo=IP (with netmask) of client interface 2
wifion=WIFI
wifionInfo=WIFI on or off (true or false)
```

Edit this file for the german version:

```
grsadmin/WEB-INF/classes/forms_de.properties
```

Add the following lines:

```
client2IP=Client 2 IP
client2IPInfo=IP (mit Netzmaske) von Interface 2
wifion=WIFI
wifionInfo=WIFI an oder aus (true or false)
```

To activate these changes, **restart the Tomcat application server**:

```
sudo /etc/init.d/tomcat* restart
```

Router form validation

In many cases it is recommended to limit the possible values which can be inserted into the database for the devices. Thereby can be avoided to store illegal values, which may later corrupt the configuration file for any router. Examples:

- Check if the input is a valid IP address (with or without netmask)
- Check if the input is a number in a given range (e.g. for WLAN signal strength)

The implementation of GRSAdmin is designed in a way, that customers can add/modify form validation for all parameters. No HTML or JavaScript programming skills are necessary, only the knowledge of regular expressions for validating values.

The rules for validating forms are set in the following file:

```
/var/lib/tomcat*/webapps/grsadmin/custom/routermod_formpatterns.jsp
```

The customer's settings are not overwritten when applying update patches to GRSAdmin.

The file contains descriptions and examples, which can easily be adapted to your needs.

Change a form field to be required

If a field is set to 'required', it is not possible to submit the form without entering some data. This avoids empty entries in the database, and therefore an illegal configuration file.

Add a line with the following pattern to the file and save it:

```
<c:set property="<db_column_name>"      target="${required}" value="true" />
```

Example:

```
<c:set property="serial"        target="${required}" value="true" />
```

On the router modify page you will see:

- The name of the correspondig field is written in bold text.
- It is not possible to submit the form without entering any text into that field.

Add a validator to a form field

If a special pattern is defined for an input field, it is not possible to submit the form with illegal values. This avoids entries in the database which may corrupt the configuration file.

Add a line with the following pattern to the file and save it:

```
<c:set property="<db_column_name>"      target="${patterns}"   value="<a_regular_expression>" />
```

Example:

```
<c:set property="serial"        target="${patterns}" value="^[OR]{1}[0-9]{11}$" />
```

On the router modify page you will see:

- It is not possible to submit the form with illegal values.

Add a tool tip to a form field

If a special pattern is defined for an input field, a tool tip shall give the user a hint, which values are expected, especially if the validator does not accept the value. Add a line with the following pattern to the file and save it:

```
<c:set property="<db_column_name>"      target="${titles}"   value="<any string>" />
```

Example:

```
<c:set property="serial"        target="${titles}" value="GRS format, e.g. R77229010001" />
```

On the router modify page you will see:

- When hovering the corresponding form field, a tool tip containing the value is shown.

Add a default value to a form field

Sometimes it is useful, if a form field is pre-set with a meaningful default value, instead of being empty.

Add a line with the following pattern to the file and save it:

```
<c:set property="<db_column_name>"      target="${defaults}"   value="<value_which_matches_required>" />
```

Example:

```
<c:set property="client1IP"         target="${defaults}" value="10.0.0.1" />
```

On the router modify page you will see:

- The corresponding form field is pre-filled with your value.
- If the value is not changed, it will be stored in the database.

Mask (hide) a value on the form

It is good practice, to mask sensitive data like passwords in forms, so anyone watching the administrator while editing the form can not read the value.

Add a line with the following pattern to the file and save it:

```
<c:set property="<db_column_name>"      target="${masked}"   value="true_ws|true|false" />
```

Possible values are:

- false The default, values are shown as text
- true Values are masked (with dots), no option to show them in cleartext on the form
- true_ws "with switch" - Values are masked like before, but a lock/unlock symbol allows to show/hide the text

Example:

```
<c:set property="presharedkey"         target="${masked}" value="true_ws" />
```

On the router modify page you will see:

- In the corresponding form field only dots are displayed, content is not readable.
- Clicking the unlock/lock symbol on the right side of the input field hides/shows the value.

Add automatic inserts when creating or modifying a router

It is sometimes be helpful to define database fields which are filled automatically when creating/modifying a router. Example:

- Create a random password for user login
- Create a unique name for a tunnel configuration
- Create a random password for a tunnel configuration

To configure these inserts, modify the following file:

- */grsadmin/custom/routermod_add_sql.jsp*

Uncomment the example tags and adapt them to your needs. Example:

```
<g:modifyRouter action="modify" routerName="${ param.name }" varToSet="VPNUser" valueToSet="VPN_${ param.name }"/>

<c:set var="randomval" value="<%= org.apache.commons.lang3.RandomStringUtils.randomAlphanumeric(13) %>"/

<g:modifyRouter action="modify" routerName="${ param.name }" varToSet="VPNPass" valueToSet="${ randomval }"/>
```

After creating a router on the page Router Management → Create New, check on that page or in the database if the auto-created values are set properly

Additional features

In addition to the standard management features, *grsadmin* provides additional features which might be helpful for managing a large amount of routers or for monitoring the routers.

CSV import and export

Managing a large amount of routers with the standard web frontend might be a time-consuming and erroneous task, especially when data like network configuration or location details are coming from an external source, e.g. another database or an Excel table. If no full integration with an existing database is possible, importing data to the config db from an externally created CSV (comma separated values) file might be a suitable alternative.

Additionally by exporting data into a CSV file a simple backup can be done.

CSV export and import functions are only working with the table *config*. All other tables are not touched.

CSV import

- Go to **Database -> CSV import**
- Choose CSV file from your disk
- Choose the delimiter which is used in your file
- Choose if the text separator ('doubletick') is used in your file. Using this text separator is highly recommended, as otherwise any occurence of the delimiter within the data fiels may corrupt your CSV file
- Choose the datahandling for the import:
  - INSERT: Data from the CSV file is inserted into the database. When a device with the same name already exists in the database, the line is ignored or the complete transaction is aborted.
  - DROP AND REPLACE: All existing data are deleted from the database, then the new data from the CSV file are inserted.
  - UPDATE: If a device with the name already exists in the database, the record is updated, otherwise the line in the CSV file is ignored.
- Errorhandling
  - Stop on errors: If any error occurs (duplicate data, wrong number of columns) the complete transaction is aborted and reverted.
  - Cleanup: If any error occurs, the error is ignored an the other lines are processed.
- Click on 'Import' to execute the CSV import

CSV export

- Go to **Database -> CSV export**
- Enter a filename how the downloaded fill will be named (default is configdb.csv).
- Destination
  - Write to file: A file download dialog is shown, the file with the given name can be stored on your computer.
  - Write to browser: The content of the CSV file is displayed in your browser window (suitable only for few devices).
- Delimiter: Choose either comma or semicolon as delimiter
- Text separator: Choose, if textseparator doubletick is inserted (highly recommended) or not.
- Click on "export" to execute the CSV export

Database export

It is possible to perform a database export like *mysqldump*. Following data can be exported:

- Table *adminusers*
- Table *config*
- Table *files* - only text data is exported, **no binary files** (e.g. GRS images)
- Table *property*

The table *grscache* is not part of the export, as this data is re-generated automatically.

To perform the export, go to the page *DB Export*. There are the following options:

- Show the exported data in the browser window or download a file (default).
- Choose the tables to be exported.
- Choose to export data only, schema only or both.

Logging options for the grsadmin web application

The log of processing the admin user interaction is written to the file *var/log/tomcat*/grsadmin.log*. The logfile is rotated daily, the previous files are renamed to contain the corresponding date, e.g. */var/log/tomcat*/grsadmin.2019-04-22.log*.

The behaviour of the logging (mainly the minimum log level) can be configured within the file *grsadmin/WEB-INF/classes/log4j2.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>

<Configuration status="INFO">

    <Properties>
        <Property name="logdir">${sys:catalina.base}/logs</Property>
        <Property name="layout">%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level [%logger{2}]: %msg%n</Property>
    </Properties>

    <Appenders>
      <Console name="Console">
          <PatternLayout pattern="${layout}"/>
      </Console>
      <RollingFile name="CONFIGSERVER"
                   fileName="${logdir}/grsadmin.log"
                   filePattern="${logdir}/grsadmin.%d{yyyy-MM-dd}.log">
          <PatternLayout pattern="${layout}"/>
          <Policies>
              <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
          </Policies>
      </RollingFile>
    </Appenders>

    <Loggers>
        <logger name="org" level="INFO" additivity="false">
            <AppenderRef ref="GRSADMIN" />
        </logger>
```

```
        <logger name="com" level="INFO" additivity="false">
            <AppenderRef ref="GRSADMIN" />
        </logger>
        <Root level="WARN">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>

</Configuration>
```

This are the possible log levels:

- OFF When no events will be logged
- FATAL When a severe error will prevent the application from continuing
- ERROR When an error in the application, possibly recoverable
- WARN When an event that might possible lead to an error
- INFO When an event for informational purposes
- DEBUG When a general debugging event required
- TRACE When a fine grained debug message, typically capturing the flow through the application
- ALL When all events should be logged

About-Page: Show versions and licence

By clicking the *About*-Link on the bottom right corner of the web page, version information of the installed components are displayed.

References

[1] https://my.garderos.com/techsup
[2] https://dev.mysql.com/doc/

Troubleshooting

Here are some notes to check, if your Configuration Server is not working properly after installation or if some functions (e.g. file upload) do not work. In many cases problems are related to the database access, which either results in error messages in your web browser or simply that the login is not possible.

SELinux

On CentOS or RedHat usually *SELinux* is activated by default. This typcially prevents a successful connect from Tomcat to the DBMS. In this case the login to GRSAdmin is not possible, the logfile */var/log/tomcat/catalina.out* contains DBMS errors like *permission denied*.

The following command disables SELinux until the next reboot, so it can used as a quick check if really SELinux is the problem:

```
setenforce 0 #Set SELinux to 'permissive' until the next reboot
```

The easiest solution to solve this, is to disable SELinux permanently by a configuration change:

```
vim /etc/sysconfig/selinux
set SELINUX=disabled
reboot
```

If you want to keep the additional security level provided by SELinux, it is necessary to change the SELinux configuration.

```
# Enable Tomcat - DBMS connection in SELinux. The -P-parameter is for persistent
sudo setsebool tomcat_can_network_connect_db on -P

# Check if the parameter is set:
sudo getsebool -a | grep tomcat
tomcat_can_network_connect_db --> on
tomcat_read_rpm_db --> off
tomcat_use_execmem --> off

# Reboot and test again
sudo reboot
```

Out Of Memory Error

When running the configuration server in Tomcat with the default settings, you might get an *Out Of Memory error*

```
java.lang.OutOfMemoryError: Java heap space or
java.lang.OutOfMemoryError: PermGen space
```

The default Tomcat memory settings are sometimes restrictive and may be too low in certain circumstances. In this case you can allocate more memory to the JVM.

Ubuntu 16.04 LTS

Edit the file */etc/default/tomcat7*, there should be a line like

```
JAVA_OPTS="-Djava.awt.headless=true -Xmx128m -XX:+UseConcMarkSweepGC"
```

Change the value of **-Xmx** to whatever suits your current memory size, e.g. on 4GB RAM systems you can use e.g. -*Xmx1024m*

Ubuntu 18.04 LTS

Edit the file */etc/default/tomcat8*, there should be a line like

```
JAVA_OPTS="-Djava.awt.headless=true -Xmx128m -XX:+UseConcMarkSweepGC"
```

Change the value of **-Xmx** to whatever suits your current memory size, e.g. on 4GB RAM systems you can use e.g. -*Xmx1024m*

CentOS 7.x

Edit the file */etc/tomcat/tomcat.conf*. If not yet present, add the following line

```
CATALINA_OPTS="-Xmx512m"
```

Change the value of **-Xmx** to whatever suits your current memory size, e.g. on 4GB RAM systems you can use e.g. -*Xmx1024m*

Problems storing (large) files in database

Database restriction

If it is not possible to store GRS image files ('.img') into the database, probably the parameter *max_allowed_packet* of the database server is set too low. The defined default value may vary depending on the Linux distribution.

Solution: Edit the file ***/etc/mysql/mysql.conf.d/mysqld.cnf*** (or on CentOS/Redhat the file ***/etc/my.cnf***), set the following parameter:

```
max_allowed_packet      = 32M
```

Webserver restriction

If the file upload via GRSAdmin breaks with an error message like "connection reset" or similar, this is due to a restriction in the Tomcat application server. This behaviour was first seen with Tomcat 9 in Ubuntu 20.04

Solution:

- Edit the file */etc/tomcat9/server.xml*, add the parameter **maxSwallowSize** in the connector definition, e.g.:

```
<Connector port="8080" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="8443" URIEncoding="UTF-8"
        '''maxSwallowSize="20971520"''' />
```

The value is in bytes, so the example allows files up to 20MB.

- When using https it also might be necessary to change the protocol handler in the configuration, so. e.g. change from

```
<Connector  port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
...
```

to

```
<Connector  port="8443" protocol="org.apache.coyote.http11.Http11Nio2Protocol"
...
```

Database replication

Repair a broken database replication

Under certain circumstances it may happen, that the replication stops at one side.

The following command shows the current replication status of the database server:

```
show slave status \G;
```

A result may look like this:

```
 mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
  ...
            Slave_IO_Running: Yes
           Slave_SQL_Running: No
             Replicate_Do_DB:
  ...
                 Last_Errno: 1050
                 Last_Error: Error 'ANY_SQL_ERROR_MESSAGE'
   ...
            Master_Server_Id: 1
                 Master_UUID: 73acb3cd-5bd2-12e7-9a0d-005056a93e8c
            Master_Info_File: /var/lib/mysql/master.info
                   SQL_Delay: 0
         SQL_Remaining_Delay: NULL
     Slave_SQL_Running_State:
          Master_Retry_Count: 86400
                 Master_Bind:
     Last_IO_Error_Timestamp:
    Last_SQL_Error_Timestamp: 180410 13:11:43
              Master_SSL_Crl:
          Master_SSL_Crlpath:
          Retrieved_Gtid_Set: 73acb3cd-5bd2-12e7-9a0d-005056a93e8c:1-20722648
           Executed_Gtid_Set: 73acb3cd-5bd2-12e7-9a0d-005056a93e8c:1-20017851
               Auto_Position: 1
        Replicate_Rewrite_DB:
                Channel_Name:
          Master_TLS_Version:
1 row in set (0.00 sec)
```

Following hints in the example above show a broken replication:

- Slave_SQL_Running: **No**
- The SQL Error shown at "Last_Error"
- The gap between "Retrieved_Gtid_Set" and "Executed_Gtid_Set"

Ways to recover the replication process:

1. Try to remove the entry causing the SQL error and restart sync
   - `STOP SLAVE;`
   - Fix database
   - `START SLAVE;`
2. Skip the replication entry which causes the conflict
   - `STOP SLAVE;`
   - Find the last executed GTID (see line "Executed_Gtid_Set")
   - Skip the next GTID by executing following commands:

     ```
     SET GTID_NEXT="73acb3cd-5bd2-12e7-9a0d-005056a93e8c:20017852";
     BEGIN; COMMIT;
     SET GTID_NEXT="AUTOMATIC";
     ```

   - `START SLAVE;`

Note:

- Try to resolve the conflict by executing one of the steps above and check the result.
- Maybe other errors occur, then execute the steps again.
- If replication was broken for a longer time it may take a while until all queued replication sets are processed.
- After all replications are done and no errors are shown anymore, compare the content of both databases.

Howto: Administration

While the other pages in this manual give a linear description of all features of the Garderos Configuration Server, this page will give you an idea how to solve a special task.

Preliminary notes:

- Whenever the GRSAdmin is required, it is assumed that the user is already logged in. The privilege level of the current user is shown at the top right corner of the web page, near the username.
- Some administration tasks need access to the server's operating system. It is assumed, that the administrator has knowledge about Linux operating systems and also has the necessary access rights (sudo).
- If any task requires access to the MySQL database - **be careful!** It is assumed, that the administrator has knowledge about database administration. In doubt, ask the Garderos Support for help. **Backup your data first!**

Backup the database

There are different ways to backup the database content. Ideally your system adminstrator has set up a scheduled task which creates a regular (e.g. daily) backup automatically. But sometimes it might be helpful to get a copy of the current database content, especially before applying major changes like adding, modifying or deleting many routers at once.

Method 1: GRSAdmin

Pro: Easy to execute, can be done by every administrator. **Con:** Limited features, binary files (e.g. GRS images) not contained in database dump.

Required	GRSAdmin
Minimum privilege level	14

- Go to: Router Management → DB export
- Choose the export destination:
  - *Write to browser*: The result is displayed in your browser window. Good for copy and paste, but only if there is not too much content in the database.
  - *Write to file*: The result will be downloaded to your computer. The filename can be modified.
- Choose which tables should be exported. Router details are stored in the table *config*, the templates are stored in *files*.
- Choose if everyting (schema and data) or only either the database structure (schema) or table content (data) should be exported.
- Click on Export database

Method 2: MySQL

Required	Shell access, MySQL knowledge
Minimum privilege level	-

- Log in into linux shell
- Execute command for MySQL dump.

Example:

```
mysqldump –u root –p config > configdb.dmp
```

Create a new admin user

Required	GRSAdmin
Minimum privilege level	15

- Go to: User Management → Create User
- Fill out the required fields: Admin username, Admin password and the Admin level (which defines the access rights of the new user)
- Optional: Enter the real name of the user
- Click on save
- Check for a success or error message

Reset a user's password

Required	GRSAdmin
Minimum privilege level	15

- Go to: User Management → Modify User
- Choose the user from the dropdown list, then click Search
- Check if the correct user is displayed
- Enter a new (temporary) password twice
- Click on save
- Check for a success or error message
- Now the user can log in again with the new (temporary) password then set a new individual password.

It is not possible for an administrator to change it's own password this way. Therefore the standard *Change password*-procedure has to be used, which requires knowledge of the current password.

Download all templates/files to filesystem of the configserver

All templates and files are stored directly in the database and can usually only edited/managed by GRSAdmin. For experienced users it might sometimes be more convenient to edit e.g. template files by the command line, especially when searching/replacing a number of values.

Garderos provides a script to get all files from the database and store them to the file system.

Required	Shell access, sudo privileges
Minimum privilege level	-

- Log in into linux shell
- Create a directory where the scripts and the files can be stored, e.g. *transferfiles*. Enter the directory and create the subdirectory files.

```
mkdir transferfiles
cd transferfiles
mkdir files
```

- Copy the script from the directory *garderosinstall* where all installation files are extracted (usually */home/garderos/garderosinstall*) to the newly created directory

```
cp /home/garderos/garderosinstall/examples/database/get_files_from_configdb.sh .
```

- Edit the script, if necessary adapt the settings for database name (*db*), database user (*user*) and database password (*MYSQL_PWD*).
- Execute the script with *sudo* rights

```
sudo ./get_files_from_configdb.sh
```

- All files which are shown in GRSAdmin under File Management → Templates and File Management → Files now can be found in the subdirectory *files*.

Upload all files from a directory to the database

If you have a set of files which have to be written into the database (e.g. template files from an older Configserver installation), it can be done with GRSAdmin but only one file by one.

Garderos provides a script to read all files from a directory and store them to the database.

Required	Shell access, sudo privileges, GRSAdmin
Minimum privilege level	1

- Log in into linux shell
- Create a directory where the insert script and the files can be stored, e.g. *transferfiles*. Enter the directory and create the subdirectory files.

```
mkdir transferfiles
cd transferfiles
mkdir files
```

- Copy the script from the directory *garderosinstall* where all installation files are extracted (usually */home/garderos/garderosinstall*) to the newly created directory

```
cp /home/garderos/garderosinstall/examples/database/write_files_into_configdb.sh .
```

- Edit the script, if necessary adapt the settings for database name (*db*), database user (*user*) and database password (*MYSQL_PWD*).
- Copy all your files to be uploaded to the subdirectory *files*.
- Execute the script with *sudo* rights

```
sudo ./write_files_into_configdb.sh
```

- Go to GRSAdmin under File Management → Templates and File Management → Files and check, if all files are correctly uploaded.

Howto: Customization

While the other pages in this manual give a linear description of all features of the Garderos Configuration Server, this page will give you an idea how to solve a special task.

Preliminary notes:

- Whenever GRSAdmin is required, it is assumed that the user is already logged in. The privilege level of the current user is shown at the top right corner of the web page, near the username.
- Some administration tasks need access to the server's operating system. It is assumed, that the administrator has knowledge about Linux operating systems and the necessary access rights (sudo).
- If any task requires access to the MySQL database - **be careful!** It is assumed, that the administrator has knowledge about database administration. In doubt, ask the Garderos Support for help. Backup your data first!
- Dependent on the linux distribution (Ubuntu or RedHat/CentOS) the installation directories may vary. In this documentation used:
  - <TC-WEBAPPS> is */var/lib/tomcat7/webapps* on Ubuntu 16.04, */var/lib/tomcat8/webapps* on Ubuntu 18.04, */var/lib/tomcat/webapps/* on RedHat/CentOS

Insert a new database column and add it to the web form

Required	Shell access, sudo privileges, MySQL access, GRSAdmin
Minimum privilege level	9

- Log in into linux shell
- Enter MySQL command line and add the new column to the table *config*.

```
mysql -u<user> -p config

( Enter password )

ALTER TABLE config ADD COLUMN `newcolumn` varchar(60) COLLATE utf8_unicode_ci DEFAULT NULL COMMENT 'new customer column';
```

  Use *column name*, *type*, *default value* and *comment* as you need.
- Go to directory *<TC-WEBAPPS>/grsadmin/WEB-INF/*
  - Edit file *web.xml*, modify the parameter *modfields* and add the column name in the comma-separated list
- Go to directory *<TC-WEBAPPS>/grsadmin/WEB-INF/classes*
  - Edit files *forms.properties* and *forms_de.properties*
  - Add two lines for the new column name, one containing the field name for GRSAdmin, the other containing a more descriptive text.

```
newcolumn=New column
newcolumnInfo=More information for the new column
```

- Restart Tomcat

```
service tomcat restart
```

- Go to: Router Management → Create new
  - Check if the new column is visible

Insert a new boolean database column and add it to the web form

Required	Shell access, sudo privileges, MySQL access, GRSAdmin
Minimum privilege level	9

- Log in into linux shell
- Enter MySQL command line and add the new column to the table *config*.

```
mysql -u<user> -p config

( Enter password )

ALTER TABLE config ADD COLUMN `booleancolumn` boolean COLLATE utf8_unicode_ci DEFAULT NULL COMMENT 'store boolean value';
```

  Use *column name* and *comment* as you need, type is *boolean* and default value can be *NULL* (false), *0* (false) or *1* (true).
- Go to directory *<TC-WEBAPPS>/grsadmin/WEB-INF/*
  - Edit file *web.xml*, modify the parameter *modbooleans* and add the column name in the comma-separated list
- Go to directory *<TC-WEBAPPS>/grsadmin/WEB-INF/classes*
  - Edit files *forms.properties* and *forms_de.properties*
  - Add two lines for the new column name, one containing the field name for GRSAdmin, the other containing a more descriptive text.

```
newcolumn=New column
newcolumnInfo=More information for the new column
```

- Restart Tomcat

```
service tomcat restart
```

- Go to: Router Management → Create new
  - Check if the new column is visible

Change the label or description of a form field

If the predefined names and descriptions on the create/modify router page are not perfectly matching, it is quite easy to change them.

Required	Shell access, sudo privileges, GRSAdmin
Minimum privilege level	1

- Log in into linux shell
- Go to directory *<TC-WEBAPPS>/grsadmin/WEB-INF/classes*
  - Edit files *forms.properties* and *forms_de.properties*
  - Change the lines suitable to the column name (the field name and/or the info, a more descriptive text) In the example the column "ip" will be modified.

```
ip=IP address
ipInfo=The public IP address
```

- Restart Tomcat

```
service tomcat restart
```

- Go to: Router Management → Create new
  - Check if the modified names are applied.

Modify router status page

By default a defined set of columns of the table *grscache* is shown on router status page. The columns to be displayed and their order are configurable, as well as the time period when the value in *Last access time* is shown as error (in red).

Required	Shell access, sudo privileges, GRSAdmin
Minimum privilege level	1

- Log in into linux shell
- Edit file *<TC-WEBAPPS>/grsadmin/WEB-INF/web.xml* (needs *sudo* access rights)
- Look for the context parameter *statusheaders* and modify the list to your needs.
  The columns *Routername*, *Template/Config* and *Last access time* are mandatory, optional columns to be shown are:
  serial, mac, ip, lastmodified, lastconfighash, lastimsi, configstatus, version, servlet, servername, count.
- Look for the context parameter *max_config_age* and set the value (in seconds) to the same value as you have defined as reconfigure interval in your templates (default is 3600s).
- Restart Tomcat

```
service tomcat* restart
```

- Go to: Information → Router Status and check if the new settings are applied.

Add/remove columns from router table

By default all form fields which can be edited on the router form are also shown on the router table (Router Management → Database). In many cases the number of columns won't fit good on a web page, so you might want to reduce the displayed colums to some essential ones.

Required	Shell access, sudo privileges, GRSAdmin
Minimum privilege level	1

- Log in into linux shell
- Edit file *<TC-WEBAPPS>/grsadmin/WEB-INF/web.xml* (needs *sudo* access rights)
- Look for the context parameter *tableheaders* and modify the list to your needs.
- Restart Tomcat

```
service tomcat restart
```

- Go to: Router Management → Database and check if the new settings are applied

Remove a field from the web form

It is recommended to remove unnecessary fields from the forms to have clearly forms.

Required	Shell access, sudo privileges, GRSAdmin
Minimum privilege level	9

- Log in into linux shell
- Edit file *<TC-WEBAPPS>/grsadmin/WEB-INF/web.xml* (needs *sudo* access rights)
- Look for the context parameter *modfields* and modify the list to your needs, e.g. remove needless fields.
- Restart Tomcat

```
service tomcat restart
```

- Go to: Router Management → Create router and check if the new settings are applied

Optional: Remove the field from the database

Although not necessary, it is als recommended to remove redundant columns from the database, as it may otherwise cause problems especially when using CSV import/export if the ballast is kept.

Required	Shell access, MySQL knowledge
Minimum privilege level	-

- Log in into linux shell
- Log in into MySQL

```
mysql -u <user> -p config
```

- Delete waste column from database

```
ALTER TABLE config DROP COLUMN <columnname>
```

Manage multiple configuration servers

When using more than one configuration server (e.g. for redundancy or load balancing), you can configure GRSAdmin of one of the servers to get information from the other server(s).

The most important thing is, that all connected servers must share the same data in the database. When setting up the servers, you have two options:

- All configuration frontents connect to the same database instance.
- Each configuration server has it's own DBMS, but databases are configured for live replication of data.

In both cases, almost all data (e.g. routers, templates, files, router status) is shared among the servers, as they connect to the same data. Of course, the *Server Logfile* (where the Tomcat application server writes access log data) is individual for each server. With the following configuration you can get access to the log files of other server within one admin frontend. There is no need to log in to each GRSAdmin individually.

Edit the following file: `/var/lib/tomcat*/webapps/grsadmin/WEB-INF/web.xml`

Add the URLs of the other server like in the following example:

Change from:

```
<param-name>configserver_url</param-name>
<param-value>http://localhost:8080/configserver/</param-value>
```

to

```
<param-name>configserver_url</param-name>
<param-value>
    http://localhost:8080/configserver/,
    http://10.10.11.12:8080/configserver/[,http://<ip_of_other_server(s)>:8080/configserver/]
</param-value>
```

Add an automatic insert when creating a router

It might be helpful to define fields which are filled automatically when creating/modifying a router. Example:

- Create a random password for user login
- Create a unique name for a tunnel configuration
- Create a random password for a tunnel configuration

Required	Shell access, sudo privileges, GRSAdmin
Minimum privilege level	9

- Log in into linux shell
- Edit the file *<TC-WEBAPPS>/grsadmin/custom/routermod_add_sql.jsp*
  - Check the comments and examples in that file
  - Modify the example tag to meet your requirements. The following parameters are available: param.name, param.secret, param.device_type and any parameter contained in the list 'modfields'. Example:

```
<g:modifyRouter action="modify" routerName="${ param.name }" varToSet="VPNUser" valueToSet="VPN_${ param.name }"/>
<c:set var="randomval" value="<%= org.apache.commons.lang3.RandomStringUtils.randomAlphanumeric(13) %>"/>
<g:modifyRouter action="modify" routerName="${ param.name }" varToSet="VPNPass" valueToSet="${ randomval }"/>
```

  - (Deprecated since r939) *Add an SQL statement which meets your requirements. The following parameters are available:*
    param.name, param.secret, param.device_type and any parameter contained in the list 'modfields'. Example:<sql:update var="routerNew" sql="UPDATE `config` SET `VPNUser` = ? WHERE name = '${ param.name }'"> <sql:param value="VPN_${ param.name }" /> </sql:update> <sql:update var="routerNew" sql="UPDATE `config` SET `VPNPass` = ? WHERE name = '${ param.name }'"> <sql:param value="<%= org.apache.commons.lang3.RandomStringUtils.randomAlphanumeric(13) %>" /> </sql:update>
- Go to: Router Management → Create New
  - Fill the form with valid values
  - Click on create
  - If the auto-created values are shown on the form itself check them directly, otherwise look into the database.

Connect the admin GUI to a radius server

If you have an existing user management system based on Radius, it is possible to connect GRSAdmin to your radius server for login of users.

Required	Shell access, sudo privileges
Minimum privilege level	-

- Log in into linux shell
- Edit the file *<TC-WEBAPPS>/grsadmin/WEB-INF/web.xml* (with sudo privileges)
  - Uncomment the section with the init-parameters *radius-server*, *radius-secret*, *radius-port* and *radius-method*
  - Change the parameter values to your settings
- Restart tomcat.
- Try to log in with any radius user
- In case of errors, check the log file */var/log/tomcat/grsadmin.log* for details.

By default, after a failed radius login the system tries to login the user against the local database. If this is not desired, change the context-paramter *adminlogin_db* from *true* to *false*. In this case, the local login is disabled.

Howto: Daily tasks

While the other pages in this manual give a linear description of all features of the Garderos Configuration Server, this page will give you an idea how to solve a special task.

Preliminary notes:

- Whenever GRSAdmin is required, it is assumed that the user is already logged in. The privilege level of the current user is shown at the top right corner of the web page, near the username.
- Some administration tasks need access to the server's operating system. It is assumed, that the administrator has knowledge about Linux operating systems and the necessary access rights (sudo).
- If any task requires access to the MySQL database - **be careful!** It is assumed, that the administrator has knowledge about database administration. In doubt, ask the Garderos Support for help. Backup your data first!

Create a new router in the database

This is the standard procedure to create a new router in the database.

Required	GRSAdmin
Minimum privilege level	9

- Go to: Router Management → Create router
- Fill all required fields:
  - Mandatory fields are labeled in **bold**
  - Some fields may limit the possible input - if the criterias are not met, they get a red border
- Click on create
- Check for a resulting success or error message

Create many routers in the database (CSV import)

To create many routers at once it is necessary to use external tools like a spreadsheet tool or a text editor. When using this feature for the first time, it is suggested first to export the current database to a .csv file which then can be loaded as template and edited.

Required	GRSAdmin, spreadsheet tool (e.g. Excel, LibreOffice) or text editor
Minimum privilege level	14

- Open your spreadsheet tool and open any previously exported .csv file. The number of columns and the column names must match to your database.
- Edit the table as you want. You can use all functions of your tool to create data, e.g. autofill, drag and drop, string concatenation using formulars etc.
- Export data into a .csv (comma separated values) file. Use the following settings:
  - Use ',' (comma) as field separator
  - Use "" (double quotes) as text separator
  - Choose "Surround text with quotes"

The resulting file should look like this:

```
"name","secret","device_type", ....
"grs001","garderos","r7728", ...
```

```
"grs002","garderos","r7728", ...
```

- Open GRSAdmin, go to Router Management → CSV import
- Select the previously created file for upload
- Use the used delimiter (comma) and text separator (doubletick)
- Choose an appropriate datahandling:
  - Insert: Insert new records to database. If a routername already exists, the import fails.
  - Drop and replace: First all existing entries are removed from the database, then the new ones are inserted.
  - Update: Only existing router names will be updated, no new routers are created.
- Choose an appropriate error handling:
  - Stop on errors: If any error occures, the import is stopped, no changes are made to the database
  - Cleanup: If an error occurs, this line is skipped and the next line will be processed.
- Click on Import

Delete a router from the database

Required	GRSAdmin
Minimum privilege level	9

- Go to: Router Management → Database
- Choose the router to delete, click the red button in the first column.

Delete many routers from the database

If it is necessary to delete many routers (let's say more than 50) it may be cumbersome to delete them one by one using the standard delete button. Here two alternative methods are presented, which can me more convenient for this task, but might also be more risky in execution.

Method 1: Using MySQL command line

This method is only useful if all the routers to delete have any common property where an SQL expression can match, e.g. if all are using the same template.

Required	Linux console, MySQL command line access
Minimum privilege level	-

- Go to the shell of the database server.
- Enter mysql command line.
- Execute mysql command to delete entries from the database.

Example:

```
DELETE FROM config WHERE device_type LIKE '%r77test%';
```

Direct modifications in the database are on your own risk.

Method 2: Using CSV Export / Import in GRSAdmin

Required	GRSAdmin, text editor or spreadsheet tool (e.g. Excel, LibreOffice)
Minimum privilege level	14

- Go to: Router Management → CSV export
- Enter a file name, leave other settings and click on Export
- Open the downloaded file in your favourite text editor or spreadsheet tool
  - Remove all lines containing routers you want to delete
  - Store file with new name. Be careful to keep the format, with comma as separator and text surrounded by quotes.
- Go to: Router Management → CSV import
- Select the created file for upload
- Change datahandling to *Drop and replace*
- Click on Import

With datahandling *Drop and replace* first all entries are removed from the table *config*, then the data contained in the uploaded file is inserted. As a result, all routers deleted in the file are deleted from the database.

Due to the internal data handling in the database, with this method all entries of the router status page get lost, and will be filled as the remaining routers fetch the configuration the next time.

Find routers which might be currently offline

Background: Every running router tries to download the configuration in a periodic interval, the so called *reload-interval* (see *configuration.reload-interval* in the GRS configuration reference). The default interval is 3600 seconds (1 hour), but can be configured from 30 seconds to 86400 seconds (1 day). If the router did not request a configuration file within the reload interval, this might indicate an error.

As the configuration server stores the timestamp of the latest request of each router it is quite easy to find out if a special router behaves as expected or not.

Required	GRSAdmin
Minimum privilege level	1

- Go to: Information → Routerstatus
- Inspect the values in the column *Last access time*, check for irregularities:
  - Timestamps older than one hour (or your defined *max-config-age*) are marked in red
  - Turn the switches ***SHOW*** *active routers* and ***SHOW*** *inactive routers* to **HIDE**, then only problematic routers are shown
  - You can sort the entries by clicking on the column name, e.g. *Last access time*

Notes:

- If you defined another reload-interval for your routers than the default 3600 seconds, you'll have to adapt the parameter *max-config-age* of GRSAdmin:
  - Edit the file *<TC-WEBAPPS>/grsadmin/WEB-INF/web.xml*
  - Search for the context-parm named **max-config-age** and change the param-value to your needs

  - Save the file and restart Tomcat
- It is only possible to define one value for all routers, so if you have different values for *reload-interval* in your templates, the colouring of timestamps and filtering for overdue routers can not fit to all routers.

Find routers which do not get a configuration

Required	GRSAdmin
Minimum privilege level	9

- Go to: Information → Server Logfile
- From the *Filter* dropdown choose "No [device/template] found"
  or
  set the *Minimum loglevel* to "WARN"
- If any result is shown, this will give you a hint which routers can not get a configuration and why.

Use filters on router status and routers in database page

Required	Shell access, sudo privileges
Minimum privilege level	-

The following applies to the pages

- Information → Router Status
- Router Management → Database

There are two different filter types implemented on each of the pages:

The filters *Filter by name* and *Filter by type* within the table header initiate a new request to the server when clicking on *Filter*. In this case the database query is adapted accordingly and only matching routers are returned.
Advantage: Especially when having thousands of routers in the database, this reduces the amount of data to be transferred and increases page load speed.

The field *Search* on top right corner of the table causes a JavaScript based inline search in the table.
Advantage:

- Once the table is loaded it is a very fast way to reduce the output.
- All table columns are included in the filter, not only the routername and template.
- No requests are sent to the server.

Modify an existing router

Required	GRSAdmin
Minimum privilege level	9

- Go to: Router Management → Database
- Find the router you want to modify, then click on the corresponding routername in the table.
- The *Modify Router* form appears. Make your changes.
- Click on save

Clone an existing router ('copy as new')

Required	GRSAdmin
Minimum privilege level	9

- Go to: Router Management → Database
- Find the router you want to clone, then click on the corresponding routername in the table.
- The *Modify Router* form appears. Change the routername (important!) and modify all other settings which vary.
- Click on copy & new

Create a new template

Required	GRSAdmin
Minimum privilege level	9

- Go to: File Management → Templates
- Scroll down to the form *Create template*
- Enter name, device type (and optional version) in the corresponding input fields
  - Click to arrow in the input field or press *arrow down* when the field is selected. This will give a list with possible values, e.g. device types which are already referenced in the database.
- Click on *Create*.

For more information on how to edit template files see the section *Modify a template*.

Define time scheduled content in templates

Required	GRSAdmin
Minimum privilege level	9

This feature was introduced in May 2018, for version > 685.
Since February 2019 *time-of-day* parameters are possible, version must be > 765.

The aim is to enable time-based template control to activate or deactivate parts of the configuration at a certain time.

How to use:

- First a marker with a certain name, a start time and an end time has to be defined in the template file.
  Format: `#define <marker> <from-time> <to-time>`
  All parts are separated by single space. Any string is allowed for <marker>. Timestamps are in format "yyyy-MM-ddTHH:mm:ss" or "HH:mm".
  Example 1: `#define T1 1970-01-01T00:00:00 2018-04-22T03:00:00`
  Example 2: `#define day 07:00 19:00`
- Later in template these defined markers can be used for one or more individual configuration lines:
  `#if <marker> <config_line>`
  Example: `#if T1 system.version=003_004_022`

How it works:

- If the current time is within the range defined for a marker, the corresponding comments and markers are removed for any matching 'if'-line
- If the current time is out of the range for a marker, the comments are kept in file
- Multiple markers with different ranges can be defined
- The "#if" statement has to be placed in front of each individual line which should be handled.
- To handle a block of configuration lines, use an "@include" statement which is handled by a marker
- Include-files can also contain definitions and markers.
- Markers are only valid within the scope of the file.

Example of a template file:

```
#define T1 1970-01-01T00:00:00 2018-05-01T03:00:00
#define T2 2018-05-01T03:00:01 9999-01-01T00:00:00
#define day 07:00 19:00
#define night 19:00 07:00

#if T1 system.version=003_004_020
#if T1 system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_020.img
#if T1 @include acl.txt

#if T2 system.version=003_004_022
#if T2 system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_022.img
#if T2 @include acl_neu.txt

#if day system.description=Normal (daylight) settings.
#if night system.description=Night settings
#if night no interface.wlan



configuration.remote.server.0.url=https://213.61.82.123:8443/configserver/config
hardware.wwan.0.dial-in-number=*99#
hardware.wwan.0.name=wwan0
interface.eth.0.ip-assignment=none
interface.eth.0.name=eth0
...
...
```

If the current time (when GRS requests a configuration) is *before* 2018-05-01 3:00:

```
#define T1 1970-01-01T00:00:00 2018-05-01T03:00:00
#define T2 2018-05-01T03:00:01 9999-01-01T00:00:00


system.version=003_004_020
system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_020.img
# ------ content of acl.txt -------------
acl.ipv4.input.0.action=ACCEPT
....
....
# ------ end of acl.txt -----------------
```

```
#if T2 system.version=003_004_022
#if T2 system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_022.img
#if T2 @include acl_neu.txt



configuration.remote.server.0.url=https://213.61.82.123:8443/configserver/config
hardware.wwan.0.dial-in-number=*99#
hardware.wwan.0.name=wwan0
interface.eth.0.ip-assignment=none
interface.eth.0.name=eth0
...
...
```

If the current time (when GRS requests a configuration) is *after* 2018-05-01 3:00:

```
#define T1 1970-01-01T00:00:00 2018-05-01T03:00:00
#define T2 2018-05-01T03:00:01 9999-01-01T00:00:00

#if T1 system.version=003_004_020
#if T1 system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_020.img
#if T1 @include acl.txt

system.version=003_004_022
system.update-url=https://config.garderos.com/configserver/config?file=grs_gwuz_armel_003_004_022.img
# ------ content of acl_neu.txt -------------
acl.ipv4.input.0.action=DROP
....
....
# ------ end of acl_neu.txt -----------------

configuration.remote.server.0.url=https://213.61.82.123:8443/configserver/config
hardware.wwan.0.dial-in-number=*99#
hardware.wwan.0.name=wwan0
interface.eth.0.ip-assignment=none
interface.eth.0.name=eth0
...
...
```

If the current time (when GRS requests a configuration) is at 23:00 (night)

```
#define day 07:00 19:00
#define night 19:00 07:00

#if day system.description=Normal (daylight) settings.
system.description=Night settings
no interface.wlan
```

Modify a template

Required	GRSAdmin
Minimum privilege level	9

- Go to: File Management → Templates
- Click on the filename of the template to be modified.
- On the *Edit file* page the content of the textarea can be freely edited:
  - To avoid syntax errors it can be helpful to copy lines from the GRS CLI (e.g. *show configuration running*) to the text area.
  - Unlike the CLI lines can be commented by adding a '#' as first character in the line. These lines will be ignored by the GRS.
  - Unlike on the GRS the order of configuration lines does not have to be alphabetically. Config parameters can be grouped as desired, empty lines are allowed.
  - Values which have to be replaced by content of the router database have to be written in the form **${** *<column_name>* **}**
  - By choosing a value from the dropdown *Show possible variable names* on the top right corner of the text area the variable is displayed and can be copied to the text area
  - Other template files can be inluded by the statement **@include** *<filename>*
- Click on *save* when finishing the modification.

Search for templates containing a text pattern

Sometimes, especially when having numerous template files, it may be required to find templates files which contains a special value.

Required	GRSAdmin
Minimum privilege level	9

- Go to: File Management → Templates
- Enter text to search in the input field *Find text in template* in the table header.
- Click on *Search*
- Now only template files which contains the searched text are shown in the list.
- To show all templates again just click on the *Reset* button in the table head.

Note:
The *Search* field on the top right corner of the table does not search within the file content but in the table content, i.e. the filenames.

Upload a file

Required	GRSAdmin
Minimum privilege level	9

Upload a template file

- Go to: File Management → Templates
- Scroll down to the form *Template upload*
- Click on *Choose file* and select the template file to upload from your file system
- Click on *Upload*
- The page will be reloaded, the file is listed in the table.

Upload a download file (GRS image, scripts etc.)

- Go to: File Management → Files
- Scroll down to the form *File upload*
- Click on *Choose file* and select the file to upload from your file system
- Allowed file types for upload (other file types will be rejected):
  - GRS image files
  - Shell scripts
  - Certificates
  - Key files (authorized_keys)
- Click on *Upload*
- The page will be reloaded, the file is listed in the table.

Modify a text file (not a template)

Editing files in the browser is limited to script files.

Required	GRSAdmin
Minimum privilege level	9

- Go to: File Management → Files
- Click on the filename of the file to be modified, works only for script files.
- On the *Edit file* page the content of the textarea can be freely edited.
- Click on *Save*

Change user's password

Each user can modify his own password by clicking on the username on top right corner of the GUI, then choose change password

If a user has forgotten his password, administrators with level 15 can (re)set the password for that user.

Required	GRSAdmin
Minimum privilege level	15

- Go to: User Management → Modify User
- Select the user to reset from the dropdown, the click on Search

- Enter a new (standard) password to the form twice, then click on save
- Now the user can log in with the new password, then set an individual password again.

Perform an upgrade of GRS routers via the Configuration Server

Beside the delivery of configuration files, it is a very common task to perform GRS image updates by the configuration server.

Required	GRSAdmin
Minimum privilege level	9

- Upload the image file to the configserver's database. Once the file is uploaded, it is accessible by the URL:
  http(s)://<your_configuration_server>/configserver/config?file=<filename>
- Modify the template of the routers which should be updated. At least the following lines are necessary:

```
system.version=aaa_bbb_ccc
system.update-url=http(s)://<your_configuration_server>/configserver/config?file=<filename>
```

- Wait for the next reconfigure interval of the routers.
- For more advanced update options please see the GRS documentation.

Example:

```
system.version=003_003_028
system.update-url=https://config1.garderos.com:8443/configserver/config?file=grs_gwuz_armel_003_003_028.img
```

Security

The basic installation of the Garderos Configuration Server does not cover any security issues. Configuration files and login credentials are transferred in clear text through the Internet.

In productive systems it is highly recommended to make additional actions!

HINT **The following measures are recommended:**

1. **Essential: Enable secure communication via HTTPS in Tomcat (see certificate management below)**
2. **Essential: Secure server(s) by firewalls**
3. Recommended: Install database on a separate server
4. Optional: Configure Tomcat to accept only connections from trusted clients
5. Optional: Configure GRS to establish only connections to a trusted server
6. Optional: Enable OCSP certificate management

Certificates for the impatient

To enable HTTPS for your server, execute the following commands:

```
cd /etc/tomcat*/
keytool -genkey -alias <configserver_name_or_ip> -keysize 4096
-keyalg RSA -keypass garderos -validity 365 -keystore config.keystore
-storepass garderos
```

Replace *<configserver_name_or_ip>* by the IP address or the domain name of your server.

Edit the ***server.xml*** file (/etc/tomcat*/server.xml):

```
vim /etc/tomcat*/server.xml
```

Up to Tomcat 8 use:

```
#add the connector for port 8443
<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true" sslProtocol="TLS"
           sslEnabledProtocols="TLSv1.2"
           keystoreFile="/etc/tomcat*/config.keystore"
           keystorePass="garderos"
           keystoreType="JKS" />
```

For Tomcat 9 and later use:

```
<Connector  port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"
               maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
               sslProtocol="TLS"
               sslEnabledProtocols="TLSv1.2+TLSv1.3" useServerCipherSuitesOrder="true"
               ciphers="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
                       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
                       TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,
                       TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,
```

```
                    TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
                    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
                    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
                    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
                    TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,
                    TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,
                    TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
                    TLS_AES_256_GCM_SHA384,
                    TLS_AES_128_GCM_SHA256"
            URIEncoding="UTF-8"
            keystoreFile="/etc/tomcat9/config.keystore"
            keystorePass="garderos"
            keystoreType="JKS"
        />
```

```
/etc/init.d/tomcat restart
```

For more details see below.

Certificate management

Creating the certificates

Self-signed certificate using Java *keytool*

The easiest way to generate a self signed certificate for Tomcat is using the Java *keytool* command. The following command creates a a new keystore file including a private key and certificate, which can instantly be used by Tomcat:

```
# sudo keytool -genkey -alias <server_name or URL> -keysize 4096
-keyalg RSA -keypass <keypass> -validity <days> -keystore
<keystore-filename> -storepass <storepass>

Fill the fields in the form as required

What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
correct?
  [no]:  yes
```

Example:

```
keytool -genkey -alias config1.garderos.com -keysize 4096 -keyalg RSA
-keypass garderos -validity 365 -keystore config1.keystore -storepass
garderos
```

Signed certificate using Java *keytool*

Example for using Java keystore file:

```
# Generate a new keystore with key and certificate like above
sudo keytool -genkey -alias <server_name> -keysize 4096 -keyalg
RSA -keypass <keypass> -validity 3650 -keystore
<keystore-filename> -storepass <storepass>

# Generate CSR from this keystore
sudo keytool -certreq -keyalg RSA -alias <server_name> -keystore
<keystore-filename> -file <servername>.csr -keypass
<keypass> -storepass <storepass>

# Send the .csr file to the CA for signing. Usually the signed
certificate is returned as .crt file.

# (optional) Import first the CA certificate to your keystore
sudo keytool -import -keystore <keystore-filename> -alias root
-import -trustcacerts -file /opt/certificates/CA/ca.cer -keypass
<keypass> -storepass <storepass>

# Import the signed server certificate
sudo keytool -import -keystore <keystore-filename> -alias
<server_name> -file ./<server>.crt -keypass <keypass>
 -storepass <storepass>
```

Self-signed certificate using openssl tools

A self-signed certificate for the Tomcat server can created by the following commands:

- Create a server key

```
openssl genrsa -out ./server.key 4096
```

- Create a certificate signing request (CSR)

```
openssl req -new -key ./server.key -out ./server.csr
```

Fill the requested fields as shown in the following example. You may leave some fields blank. Don't use a challenge password.

```
Country Name (2 letter code) [XX]:DE
State or Province Name (full name) []:BY
Locality Name (eg, city) [Default City]:Munich
Organization Name (eg, company) [Default Company Ltd]:Garderos
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:config.garderos.com
```

```
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

- Create a self-signed x.509 certificate, valid for 1 year (365 days):

```
openssl x509 -req -days 365 -in ./server.csr -signkey ./server.key -out ./server.crt
```

Signed certificate using openssl tools

A officially trusted certificate for the Tomcat server can created using the following commands:

- Create a server key

```
openssl genrsa -out ./server.key 4096
```

- Create a certificate signing request (CSR)

```
openssl req -new -key ./server.key -out ./server.csr
```

  Fill the requested fields as shown in the following example. You may leave some fields blank. Don't use a challenge password.

```
Country Name (2 letter code) [XX]:DE
State or Province Name (full name) []:BY
Locality Name (eg, city) [Default City]:Munich
Organization Name (eg, company) [Default Company Ltd]:Garderos
Organizational Unit Name (eg, section) []:IT
Common Name (eg, your name or your server's hostname) []:config.garderos.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

- Send the CSR file to a trusted certification authority, like VeriSign or Thawte and buy a trusted certificate. Dependent on the issuer you will receive the certificate, commonly it is sent as .crt file by email.

Configuration of Tomcat for HTTPS

Basic rules for Tomcat configuration:

1. There must only be the HTTPS connector on port 8443. Other connectors (like the AjpConnector) must be removed or commented.
2. HTTPS configuration:
   - Limit the SSL to support only protocol TLS1.2 and newer, as older versions can not be regarded as secure anymore.
   - It is also advised to limit the usable TLS ciphers to ones which are regarded as secure, e.g. by the BSI ([1])
3. The *webapps* should only contain the necessary applications, e.g. *configserver* and *grsadmin*, optional *logging*.
4. Usually it is not possible to configure Tomcat to use port 443, as it is not possible to bind to ports lower than 1024 for applications not started as root. To use port 443 it is recommended to use port forwarding.

SSL parameters can be found in the HTTP-connector in the file /etc/tomcat*/server.xml:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"

           sslImplementationName="org.apache.tomcat.util.net.jsse.JSSEImplementation"

           maxThreads="150" SSLEnabled="true" scheme="https" secure="true" sslProtocol="TLS"

           sslEnabledProtocols="TLSv1.2+TLSv1.3" useServerCipherSuitesOrder="true"

           ciphers="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,

                    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,

                    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,

                    TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,

                    TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,

                    TLS_DHE_RSA_WITH_AES_256_CBC_SHA,

                    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,

                    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,

                    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,

                    TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,

                    TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,

                    TLS_DHE_RSA_WITH_AES_128_CBC_SHA,

                    TLS_AES_256_GCM_SHA384,

                    TLS_AES_128_GCM_SHA256"

            URIEncoding="UTF-8"


            clientAuth="true"                       // if GRS clients have to show a valid certificate: true|false|want

            truststoreFile="/path/to/truststore"    // contains CA which signed the GRS certificates

            truststorePass="<truststore_password>"  // password for the truststore

            truststoreType="JKS"


            keystoreFile="/path/to/keystore"        // contains server key and signed certificate which is presented to the GRS device

            keystorePass="<keystore_password"       // password for the keystore

            keystoreType="<keystore_type>"          // optional - contains "PKCS12" if keystoreFile is in PKCS12-format

/>
```

Basic Tomcat configuration

If you want to use the secure HTTPS protocol, the server needs a private key and a certificate. This enables the server to encrypt the data before sending, so that the communication can not be intercepted. The private key and certificate of the server can be given in Java **keystore** or in **PKCS#12** format.

Using Java keystore

If the keystore-files are created as written before, Tomcat can easily be configured to use this keystore for https connections:

1. Configure the following in the *Connector*-Part of the server.xml (assumed the file is generated in /etc/tomcat*/tomcat.keystore)

```
keystoreFile="/etc/tomcat*/tomcat.keystore"
keystorePass="<keystore_pass>"
```

Using PKCS12 keystore file

1. Convert the certificate and key file to a PKCS12 keystore

```
openssl pkcs12 -export -in <server-crt> -inkey <server-key> -out <pkcs12-file> -name <alias> -CAfile <CA-certificate> -caname root -chain
```

1. Configure the following in the *Connector*-Part of the server.xml (assumed the file is generated in /etc/tomcat*/tomcat.keystore)

```
keystoreFile="/etc/tomcat*/server.p12"
keystorePass="<keystore_pass>"
keystoreType="PKCS12"
```

- *server-crt* is the signed certificate of the server.
- *server-key* is the private key of the server.
- *pkcs12-file* is the file in PKCS#12-format containing private key, certificate and CA certificate, e.g. **server.p12**
- *alias* is a symbolic name for the entry. The value is arbitrary, but must be unique, e.g. *config1*
- *CA-certificate* is the certificate file of the signing CA.
- *root* is a symbolic name for the CA. The value is arbitrary, but must be unique.

HINT The *openssl* command requires a password for export in PKCS#12 format. The password must be set in the Tomcat configuration file *server.xml* in the attribute *keystorePass*.

Optional webapp configuration

If both connector ports (8080 and 8443) are working, Tomcat should be configured to allow only secure communication and to redirect requests on port 8080 to the secure port 8443.

Edit *configserver/WEB-INF/web.xml* and/or *grsadmin/WEB-INF/web.xml*, and add the following lines inside the container element:

```
<!-- Require HTTPS for everything (exceptions may follow below...) -->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HTTPSOnly</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
```

```
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>

<!-- optional: exclude /img (favicon) and /css. from the above rule-->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HTTPSOrHTTP</web-resource-name>
        <url-pattern>*.ico</url-pattern>
        <url-pattern>/img/*</url-pattern>
        <url-pattern>/css/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

This configuration file declares that the entire webapp is meant to be HTTPS only, that the container should intercept HTTP requests and redirect them to the equivalent *https://* URL. The exception are requests having URL patterns that match the *HTTPSOrHTTP* web resource collection, in which case the requests will be served through the protocol the request came in, HTTP or HTTPS.

Restart Tomcat. It now redirects HTTP requests to HTTPS, and it serves the webapp by HTTPS only.

The complete Tomcat-SSL documentation can be found at:

https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html

HINT Check the open ports on the server with the *netstat*-command. After modifying the Tomcat configuration the ports 8009 and 8080 should not be visible anymore, but port 8443 should still be there.

```
netstat -ant
```

Don't forget to change the firewall rules (if present).

```
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           state NEW
tcp dpt:8443
```

Note:
If working with self-signed certificates, the *grsadmin* can not connect to the *configserver*, because Java rejects to accept the certificate. To avoid such problems do the following:

- Import the CA certificate to the truststore of your Java installation (which is used by Tomcat):

```
sudo keytool -importcert -alias "yourca" -file ca.cer -keystore
/etc/pki/java/cacerts
or
sudo keytool -importcert -alias "yourca" -file ca.cer -keystore
/etc/ssl/certs/java/cacerts
```

- Use the name or IP of the certificate in the configuration of the *grsadmin*-webapp:

```
vim grsadmin/WEB-INF/web.xml
...
    <context-param>
       <param-name>configserver_url</param-name>
       <param-value>https://'''config1.garderos.com''':8443/configserver/</param-value>
   </context-param>
...
```

- Restart Tomcat

Enabling HSTS in Apache Tomcat

The HTTP HSTS is a mechanism that allows websites to declare that they can be only accessed via secure connection (HTTPS).

To activate HSTS in Tomcat, a special *filter* has to be configured in */etc/tomcat*/web.xml*:

```
<filter>
  <filter-name>httpHeaderSecurity</filter-name>
  <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
  <async-supported>true</async-supported>
  <init-param>
    <param-name>hstsEnabled</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>hstsMaxAgeSeconds</param-name>
    <param-value>31536000</param-value>
  </init-param>
  <init-param>
    <param-name>hstsIncludeSubDomains</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

<!-- The mapping for the HTTP header security Filter -->
<filter-mapping>
  <filter-name>httpHeaderSecurity</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Restrict access to admin pages

In many scenarios it is necessary to allow access to the configuration webapp from *everywhere*, as GRS devices are typically connecting through the Internet to the Configserver. On the other hand, the access to the administrator pages can often be restricted to a limited number of users, usually coming from certain IP addresses (e.g. company intranet).

IP filter for GRSAdmin pages

- Go to the directory **/var/lib/tomcat*/webapps/grsadmin/WEB-XML**
- Edit the file **web.xml**
  - Add the address filter below the existing filter/filter mapping. Example:

```
<filter>
  <filter-name>Remote Address Filter</filter-name>
  <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
  <init-param>
    <param-name>allow</param-name>
    <param-value>127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1</param-value>
  </init-param>
  <init-param>
    <param-name>denyStatus</param-name>
    <param-value>404</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>Remote Address Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Restart tomcat

Notes to the filter parameters:

- The parameter *allow* gives access only to the given IP addresses and blocks the rest. The parameter *deny* allows everything but the given addresses.
- IPv4 and IPv6 addresses have to be defined separately. Multiple addresses can be separated by '|'. Wildcards have to be defined as Java regular expressions, see example.
- The *denyStatus* is 403 by default ('Access denied').

For more information see http:/ / tomcat. apache. org/ tomcat-7. 0-doc/ config/ filter. html#Remote_Address_Filter

Extended configuration

Server only accepts connections from a trusted GRS client

To increase the security it is possible to configure the configuration server in a way, that only connections from devices with valid certificates are allowed.

Step 1: Tomcat only allows connections from trusted clients

- Import the certificate of the CA signing the router's certificates to a local keystore:

```
keytool -import -keystore ca_garderos.jks -file CA_production.pem -alias garderosca -trustcacerts
```

When importing the CA certificate, you will be aksekd for a keystore password - remember this password for the next step.

- Configuration in **/etc/tomcat*/server.xml**:

```
  clientAuth="true"
  truststoreFile="/etc/tomcat*/ca_garderos.jks"
  truststorePass="<keystore_pass>"
  truststoreType="JKS"
```

- Restart Tomcat

After a server restart you won't be able to access the GRSAdmin page via https with your browser, as your browser can not present the required certificate.
Additionally any script which sends data to this server using curl (e.g. any logging script) would fail now. See below for the solution.

Step 2: Configure the GRS to send the created certificates on each request

```
system.certificate.my-certificate=local:router.crt
system.certificate.my-private-key=local:router.key
```

Both the key and certificate files have to be stored on the GRS device before. Garderos devices are usually delivered with certificates signed by the Garderos CA, named *router.crt* and *router.key*.

Step 3 (optional): The config servlet checks the CN of the presented certificate if it can be found in the database (whitelist) :

- Add the following parameter to the servlet configuration in *configserver/WEB-INF/web.xml*:

```
    <init-param>
        <param-name>certCN-check-dbkey</param-name>
        <param-value>serial</param-value>
    </init-param>
```

How it works:

- With the configuration of *Step 1* the servlet code is only reached, if the client (GRS device) presented a valid certificate
- The servlet extracts the CN (common name) from the certificate
- The value of the extracted CN is compared with the corresponding value in the column *serial* of the configserver's database
- If the CN matches with the value in the database, the configuration is delivered. Otherwise an error message is sent to the client.

To make scripts on the routers which sends data to the logging webapp work with certificates, use the following curl options:

- *--cacert*: Path to the CA certificate which signed the server's certificate. Can be ommitted by using the *-k* attribute.
- *--cert*: Path to the router certificate.
- *--key*: Path to the private key file.

A complete request might look like this:

```
curl --cacert /mnt/hd/files/ca.cer  --cert /mnt/hd/files/router.crt --key /mnt/hd/files/router.key  --data-urlencode "...." $logserver
```

GRS shall only connect to trusted servers

It is possible to increase the security by configuring the GRS devices in a way, that the only accept a configuration from a trusted server. Following conditions have to be met:

- The certificate of the CA which signed the server certificate has to be known to the GRS. This can be done with the following configuration option:

```
system.certificate.ca-certificate=local:server_ca.cer
```

- All certificates must be valid, i.e. not outdated.

Connecting an OCSP Responder

The validity of a client certificate of a GRS router can be checked by the configuration server against an OCSP responder. The communication between configuration server and OCSP responder is done either by using HTTP or HTTPS. HTTP should be preferred as it is more performant. All configuration parameters of the connection to the OCSP responder are done in the web.xml configuration file.

Possible parameters:

- **ocspResponderUrl**: (Mandatory) URL of the OCSP responder.
- **ocspPolicy**: (Optional) How strict shall the responses be interpreted.
  - **strict** - Only accept a certificate if:
    - OCSP-Server answered "good" and
    - the OCSP response is correctly signed and
    - if using HTTPS: The certificate of the OCSP server is valid.
  - **moderate** - Accept a certificate if:
    - The OCSP response is correctly signed and
    - OCSP-Server answered "good" and
    - if using HTTPS: The certificate of the OCSP server is valid.
    - OCSP-Server is unreachable.
  - **tolerant** - If no value is set, the default *tolerant* is used: Only reject a certificate if:
    - OCSP-Server answers "revoked".
- **ocspHttpUser**: (Optional) Username for HTTP basic authentication against the OSCP responder.
- **ocspHttpPassword**: (Optional) User password for HTTP basic authentication against the OSCP responder.
- **ocspClientCert**: (Optional) Client certificate to present to the HTTPS OCSP responder if certificate authentication is used.
  HTTPS only. The certificate must be in PKCS#12 format.
- **ocspClientCertPasswd**: (Optional) Password for the client certificate file.

```
...
  <context-param>
    <description>OCSP Responder URL</description>
    <param-name>ocspResponderUrl</param-name>
    <param-value>http://127.0.0.1:3456</param-value>
  </context-param>

  <context-param>
    <description>OCSP Policy: How strict shall the responses be interpreted</description>
    <param-name>ocspPolicy</param-name>
    <param-value>STRICT</param-value>
  </context-param>

  <!-- Optional parameters for HTTPS -->
  <!--
  <context-param>
    <description>
      Optional: HTTPS only! Client certificate to present to the HTTPS OCSP
                responder if certificate authentication is used. The certificate
                must be in PKCS#12 format.
    </description>
    <param-name>ocspClientCert</param-name>
    <param-value>/usr/share/tomcat6/conf/client.p12</param-value>
  </context-param>

  <context-param>
    <description>
      Optional: HTTPS only, password for the client certificate file (ocspClientCert)
    </description>
    <param-name>ocspClientCertPassword</param-name>
    <param-value>garderos</param-value>
  </context-param>

  <context-param>
    <description>
      Optional: Username for HTTP basic authentication against OSCP responder.
    </description>
    <param-name>ocspHttpUser/param-name>
    <param-value>ocspuser</param-value>
  </context-param>

  <context-param>
    <description>
      Optional: User password for HTTP authentication against OSCP responder.
    </description>
    <param-name>ocspHttpPassword</param-name>
    <param-value>topsecret</param-value>
```

```
    </context-param>
    -->
  ...
```

Note:

1. It is not necessary to define a separate CA certificate for the OCSP communication, as the CA certificate is already known due to the SSL authentication.
2. Tomcat 7 has no native OCSP support, therfore OCSP is implemented in the servlet. This means, that the HTTPS communication between GRS and configuration server is always successful, but a request with an invalid certificate is answered with the HTTP error code 400.

Using standard ports 80 and 443 with Tomcat

Usually Tomcat is using ports 8080 and 8443, as the ports below 1024 are only allowed for applications having root permissions.

To avoid running Tomcat as *root* use iptables port forwarding to redirect the standard ports to the Tomcat ports.

Example firewall rules:

```
## forward port 80 to 8080
iptables -t nat -A OUTPUT     --destination localhost   -p tcp --dport 80 -j REDIRECT --to-ports 8080
iptables -t nat -A OUTPUT     --destination <public_ip> -p tcp --dport 80 -j REDIRECT --to-ports 8080
iptables -t nat -A PREROUTING --destination <public_ip> -p tcp --dport 80 -j REDIRECT --to-ports 8080

# forward port 443 to 8443
iptables -t nat -A OUTPUT     --destination localhost   -p tcp --dport 443 -j REDIRECT --to-ports 8443
iptables -t nat -A OUTPUT     --destination <public_ip> -p tcp --dport 443 -j REDIRECT --to-ports 8443
iptables -t nat -A PREROUTING --destination <public_ip> -p tcp --dport 443 -j REDIRECT --to-ports 8443
```

iptables on CentOS

It is important to have the necessary firewall rules for port forwarding (besides the rules that protect your server) permanently available. On RedHat / CentOS the firewall settings are stored in the file ***/etc/sysconfig/iptables***.

HINT With RHEL 7/CentOS 7, firewalld was introduced to manage iptables. As firewalld is probably more suited for workstations than for server environments, we suggest to use the classic iptables setup.

```
yum install iptables-services
systemctl enable iptables
#when using IPv6
systemctl enable ip6tables
systemctl start iptables
systemctl start ip6tables
```

An example file including the rules from above might look like this (IPv4 only):

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
```

```
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT   -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT   -p icmp -j ACCEPT
-A INPUT   -i lo -j ACCEPT
-A INPUT   -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT   -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
-A INPUT   -m state --state NEW -m tcp -p tcp --dport 8080 -j ACCEPT
-A INPUT   -m state --state NEW -m tcp -p tcp --dport 8443 -j ACCEPT
-A INPUT   -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT

# add forwarding of https (443) to tomcat-https (8443) and 80 to 8080
*nat
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -d 212.16.16.16/32 -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A OUTPUT     -d 127.0.0.1/32    -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A OUTPUT     -d 127.0.0.1/32    -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A OUTPUT     -d 212.16.16.16/32 -p tcp -m tcp --dport 443 -j REDIRECT --to-ports 8443
-A PREROUTING -d 212.16.16.16/32 -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
-A OUTPUT     -d 127.0.0.1/32    -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
-A OUTPUT     -d 127.0.0.1/32    -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
-A OUTPUT     -d 212.16.16.16/32 -p tcp -m tcp --dport 80  -j REDIRECT --to-ports 8080
COMMIT
```

If you have a working firewall configuration built by inserting the necessary rules manually, you can convert it to a config file with the command **iptables-save** with an I/O redirect. The file can be /etc/sysconfig/iptables or any other file.

```
/sbin/iptables-save > /etc/sysconfig/iptables
```

To activate the rules call

```
systemctl restart iptables
```

To load the rule set from the configuration file (e.g. when playing around with iptables) into the running iptables service use **iptables-restore** with an I/O redirect. When iptables-restore is run the existing rules in the memory are flushed and the new set is loaded.

```
$/sbin/iptables-restore < /etc/sysconfig/iptables
```

References

[1] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf

Backup and Redundancy

This article describes what files have to be included in backups and how to set up a redundant DBMS system with one master and one slave using MySQL replication. As the new Garderos Configuration server released in 2017 stores all data including templates and image files in the database, no file system based synchronisation is necessary.

Backup

Backup of Configuration Server

If there is already an existent backup tool (like a tape backup) for complete servers, it would be a good idea to integrate the configuration server(s) to the backup strategy.

If there is no such system, it is usually not necessary to create a full backup of the servers, as a fresh installation of the server can be done within a few hours. There are only a few customized files which should be saved to get a new server with the same configuration.

Here is a list what should be saved - the exact directory names may vary depending on your distribution:

- Configserver webapps (/var/lib/tomcat*/webapps/*)
- Tomcat configuration file (/etc/tomcat*/server.xml)
- Server certificate(s), often also located in /etc/tomcat*/... - in doubt check the server.xml to find the path to the certificate.

A simple but effective way to do these backups is by creating a script which collects all these files in a single archive file. This script should be executed regularly by installing a entry in the crontab file.

Backup of database

Even with redundant databases as described below it is necessary to create regularly backups of the whole database, as redundant systems may protect from hardware failures but not from accidentally deleted records.

A MySQL database backup can usually be performed within one command:

```
mysqldump --user='db_user' --password='db_password' config > configdb_dump.sql
```

If you are using the logging-feature of the configuration server for collecting statistic data, it might be convenient to store these data additionally, but it is not necessary.

```
mysqldump --user='db_user' --password='db_password' logdata > logdata_dump.sql
```

More information can be found in the MySQL documentation [1]

To create these database dumps regularly it is recommended to create a crontab entry, which executes the backup at a certain time or interval. Example:

```
crontab -e
# perform dump at 3:15 AM every day
15 3 * * * /usr/bin/mysqldump --user='db_user' --password='db_password' config > /tmp/configdb_dump.sql
```

For more advanced options the backup commands can be wrapped to a script where e.g. an uniqe filename (e.g. containing the execution date/time) is created so that more backup copies are created.

DBMS redundancy

MySQL master-master replication using GTID

Combined from Mysql master-master replication [2] and Mysql replication with GTID [3].

Prerequisites:

- MySQL > 5.7.6 or
- MariaDB > 10.0

MySQL: Modify mysql server configuration on both servers

First modify the MySQL configuration file on both servers to enable synchronisation (on Ubuntu 20.04: */etc/mysql/mysql.conf.d/mysqld.cnf*), add e.g. at the end of the file (but within the section '[mysqld]':

```
# added to enable replication
log-bin=mysql-bin
innodb_flush_log_at_trx_commit = 1
sync_binlog = 1
binlog-format = ROW
binlog_do_db = config #only the given database will be synced

gtid-mode=ON
enforce-gtid-consistency = true
log-slave-updates = true

 server-id = 1
```

- Server-ID must be different on each server, e.g. use the last digits of the IP address (or just 1 and 2).
- Find details of the config options in the links above or the MySQL documentation
- Make sure to allow remote access to the databases, comment the following line (if present)

```
# bind-address            = 127.0.0.1
```

Restart both servers:

```
sudo systemctl restart mysql
```

MariaDB: Modify mysql server configuration on both servers

- Example in CentOS: Add file 'replication.cnf' in */etc/my.cnf.d/*
- Example in Debian 10: Add file 'replication.cnf' in */etc/mysql/mariadb.conf.d*

```
[mariadb]
log-basename=master1
log-bin

binlog-format=row
binlog_do_db=config

server_id = 1
```

- Server-ID must be different on each server, e.g. use the last digits of the IP address (or just 1 and 2).
- log-basename must be different on each server, e.g. use master1 and master2

- Make sure to allow remote access to the databases, comment the following line (if present)

```
# bind-address            = 127.0.0.1
```

Create replication user

On both servers execute:

```
CREATE USER 'repli'@'%' identified by 'garderosR';
GRANT REPLICATION SLAVE ON *.* to 'repli'@'%';
```

Options:

- Username (*repli*) can be free defined
- Any password can be set
- For security reasons the hostname can be definede, e.g.

```
GRANT REPLICATION SLAVE ON *.* TO repli@'172.16.26.%' IDENTIFIED BY 'garderosR';
```

Transfer data from existing master

Create database dump:

```
mysqldump -u root -p --all-databases --single-transaction --triggers --routines --events > repl_all.sql
```

Transfer dump to master 2:

```
scp repl_all.sql <master2>
```

Insert database content to master 2:

```
mysql -u root -p < repl_all.sql
```

Start replication on both machines

For MySQL:

On master 2:

```
stop slave;
CHANGE MASTER TO MASTER_HOST = 'master_1', MASTER_USER = 'replicator',
MASTER_PASSWORD = 'password', MASTER_AUTO_POSITION=1,
get_master_public_key=1;
start slave;
```

On master 1:

```
stop slave;
CHANGE MASTER TO MASTER_HOST = 'master_2', MASTER_USER = 'replicator',
MASTER_PASSWORD = 'password', MASTER_AUTO_POSITION=1,
get_master_public_key=1;
start slave;
```

Since MySQL version 8 the parameter *get_master_public_key=1* is necessary to allow the replica to connect to the master. See MySQL8 documentation [4] for details.

For MariaDB:

On master 2:

```
stop slave;
CHANGE MASTER TO MASTER_HOST = '<master_1-ip>', MASTER_USER = 'repli', MASTER_PASSWORD = 'garderosR', master_use_gtid=slave_pos;
start slave;
```

On master 1:

```
stop slave;
CHANGE MASTER TO MASTER_HOST = '<master_2-ip>', MASTER_USER = 'repli', MASTER_PASSWORD = 'garderosR', master_use_gtid=slave_pos;
start slave;
```

Testing the replication

- Create entry on master_1, check if present on master_2
- Create entry on master_2, check if present on master_1

Commands

```
show master status \G
show slave status \G
```

Repair a broken replication

See chapter Troubleshooting.

MySQL database replication using global transaction identifiers

Basically any existent MySQL replication mechanism should be sufficient for beeing used with the Garderos Configuration Server. Currently we suggest to use replication based on global transaction identifiers (GTIDs) as described here [5].

From the MySQL documentation:

> *The newer method based on global transaction identifiers (GTIDs) is transactional and therefore does not require working with log files or positions within these files, which greatly simplifies many common replication tasks. Replication using GTIDs guarantees consistency between master and slave as long as all transactions committed on the master have also been applied on the slave.*

The restrictions for using GTIDs do not apply for the Configuration Server.

Here are the main steps taken from the MySQL documentation [5]:

Create user for replication on the master

Each slave connects to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect.

```
CREATE USER 'repl'@'%' IDENTIFIED BY 'garderosR';
GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%';
```

- Username ('repl') and password ('garderosR') can be replaced
- For security reasons the host range (here '%' for 'any' host) can be limited

Synchronize the servers

Make the servers read-only. To do this, enable the read_only system variable by executing the following statement on both servers:

```
mysql> SET @@global.read_only = ON;
```

Then, allow the slave to catch up with the master. It is extremely important that you make sure the slave has processed all updates before continuing.

On master:

```
mysqldump -u root -p config --allow-keywords --single-transaction --flush-logs --master-data=2 -r /tmp/master_configdb.sql


scp /tmp/master_configdb.sql <slave>
```

On slave:

```
mysql -u root -p config < /tmp/master_configdb.sql
```

Stop both servers

```
sudo /etc/init.d/mysql stop
```

Set unique server id and define replication options (on master and slave)

Edit my.cnf file (on Ubuntu 16.04: */etc/mysql/mysql.conf.d/mysqld.cnf*), add e.g. at the end of the file (but within the section '[mysqld]':

Master

```
[mysqld]
...
...
# added to enable replication - this is the master.
log-bin=mysql-bin
server-id=1
innodb_flush_log_at_trx_commit=1
sync_binlog=1

gtid-mode=ON
enforce-gtid-consistency
```

Slave

```
[mysqld]
...
...
# added to enable replication - this is a slave
server-id=2
```

```
gtid-mode=ON
enforce-gtid-consistency
```

Restart the servers

```
sudo /etc/init.d/mysql start
```

Direct the slave to use the master

Execute a CHANGE MASTER TO statement on the slave, using the MASTER_AUTO_POSITION option to tell the slave that transactions will be identified by GTIDs.

```
mysql> CHANGE MASTER TO MASTER_HOST = '<hostname>',
MASTER_PORT = <port>, MASTER_USER = '<username>',
MASTER_PASSWORD = '<password>', MASTER_AUTO_POSITION = 1;

Example:
mysql> CHANGE MASTER TO MASTER_HOST = '172.16.26.100', MASTER_PORT =
 3306, MASTER_USER = 'repl', MASTER_PASSWORD = 'garderosR',
MASTER_AUTO_POSITION = 1;
```

```
mysql> START SLAVE;
```

Change <hostname>, <port>, <username> and <password> to your settings.

Disable read-only mode

```
SET @@global.read_only = OFF;
```

Check replication

Use the following commands to check the replication status:

On master:

```
SHOW MASTER STATUS;
```

On slave:

```
SHOW SLAVE STATUS;
```

Additionally change any entry in the master database either manually or via the connected configserver. Check if the change is applied to the slave.

MySQL database replication using binary log file (deprecated)

As the current implementation of the Configserver is using InnoDB as storage engine, a more advanced replication mechanism can be used (see above).

The following set of commands activates database replication for MySQL on RedHat EL (CentOS should be the same).

For a detailed description please see the MySQL documentation:

https://dev.mysql.com/doc/refman/5.7/en/replication.html

On CentOS / RedHat the DB content is stored in /var/lib/mysql
The configuration file is /etc/my.cnf

HINT Edit/**etc**/**hosts** on both servers and add representative names to identify the server (e.g. *masterdb* and *slavedb*). It makes reconfiguration easier, if later a machine has to be replaced with a different IP address.

On the master

Execute:

```
sh> mysql -u root -p
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%' IDENTIFIED BY 'garderosR';
```

In this case the replication user is named *repl*, and it's password is *garderosR*.

Now lock the database to define a stable state:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Log in to the master server from a different console and transfer the complete database with the current state to the slave server.

```
mysqldump -u root -p --all-databases --allow-keywords --single-transaction --flush-logs --master-data=2 -r /tmp/masterdb.sql
scp /tmp/masterdb.sql <slave>
```

After duplicating the database execute:

```
mysql> SHOW MASTER STATUS;
```

If the result is empty, binary logging was not activated. In this case you will need the following values later:

- "" (empty string) for "MASTER_LOG_FILE" and
- "4" for "MASTER_LOG_POS".

If the result is not empty, note the shown values instead. These values must be used later.

Now the master database can be unlocked.

```
mysql> UNLOCK TABLES;
```

Modify the *my.cnf* (e.g. */etc/mysql/my.cnf*) file on the master and add the following lines if not already there:

```
sh> vim my.cnf
    [mysqld]
       log-bin=mysql-bin
       server-id=1
```

Now restart the master database server

```
sh> /etc/init.d/mysqld restart
```

On the slave

These steps must be executed on the **slave**:

Install the content of the master database:

```
mysql -u root -p < /tmp/masterdb.sql
```

Edit *my.cnf* file. Add the following to the [mysql] section, then restart the database server:

```
vim /etc/mysql/my.cnf
   [mysqld]
   slave-id=2

/etc/init.d/mysqld restart
```

Log in to the mysql console and execute the following commands. Use your system's parameters:

```
sh> mysql -u root -p
mysql> CHANGE MASTER TO MASTER_HOST='10.10.11.62', MASTER_USER='repl', \
                      MASTER_PASSWORD='garderosR',MASTER_LOG_FILE=' ',MASTER_LOG_POS=4;
mysql> START SLAVE;
```

(Deprecated) File backup and replication

As with Configuration Server released and delivered after September 2016 all data including templates and files are stored within the database, the file replication of these files can be omitted. For all other files like configuration files, logfiles etc. the *backup*-parts of the following can be kept.

On Linux systems the tool **rsync** is often used for backups and file replication between two machines.

Usually the commands are put into a shell script, which is executed periodically e.g. by a cronjob.

rsync can be used to:

- Backup files or directories to the same machine.
- Backup files or directories to a remote machine.
- Keep directories on two different machines synchronized.

Basic command for remote file backup

This command copies all files from the source to the destination directory. SSH is used for secure data transfer. Files at destination that do not yet exist at the source are ignored.

```
rsync -avz -e ssh <source-dir> <user>@<remote-ip>:<dest-dir>
```

Full synchronization of directories

The following command forces a full synchronization of two directories. Any file that does not yet exist or is deleted at the source will be deleted at the destination.

```
rsync -avz --delete -e ssh <source-dir <user>@<remote-ip>:<dest-dir>
```

SSH auto-login

It is preferred to transfer the data through a secure connection, usually by SSH. To enable automatic login from one computer to another without entering the passwords it is necessary to create an SSH key pair on the source computer and transfer the public key to the destination.

SSH key exchange

On master: create SSH key pair

```
cd ~/.ssh
ssh-keygen -t rsa -b 2048 ( confirm filename, no passphrase )
```

Transfer the public key to the slave:

```
scp ~/.ssh/id_rsa.pub <user>@<slave>
```

On slave: add the public key to the user's authorized keys file:

```
cat id_rsa.pub >> ~/.ssh/authorized_keys
```

Advanced file replication using incron

Instead syncronizing files on a time based schedule as done via standard *crontab*, the *incron* is able to watch the filesystem for changes and execute a script every time something changed in the configured directories.

Installation:

```
sudo apt-get install incron
```

Configuration:

```
vim /etc/incron.allow
Add user "root"
```

```
incrontab -e
```

```
/var/lib/tomcat*/webapps/configserver/WEB-INF/files IN_MODIFY,IN_CREATE,IN_DELETE,IN_NO_LOOP /opt/configserver/bin/mirrorfiles.sh $@

/var/lib/tomcat*/webapps/configserver/WEB-INF/templates IN_MODIFY,IN_CREATE,IN_DELETE,IN_NO_LOOP /opt/configserver/bin/mirrorfiles.sh $@
```

The tailing parameter **$@** sends the corresponding directory to the called script as parameter. hen incron]

Example for */opt/configserver/bin/mirrorfiles.sh*:

```
#!/bin/bash
# mirrors files and templates to second configuration server using rsync

if [ -z "$1" ]; then
    echo "You must specifiy a directory to sync"
    exit 1
fi
dirParam=$1
logfile=$( basename $dirParam )

if [ -z "$logfile" ]; then
    echo "Obviously not a valid directory"
```

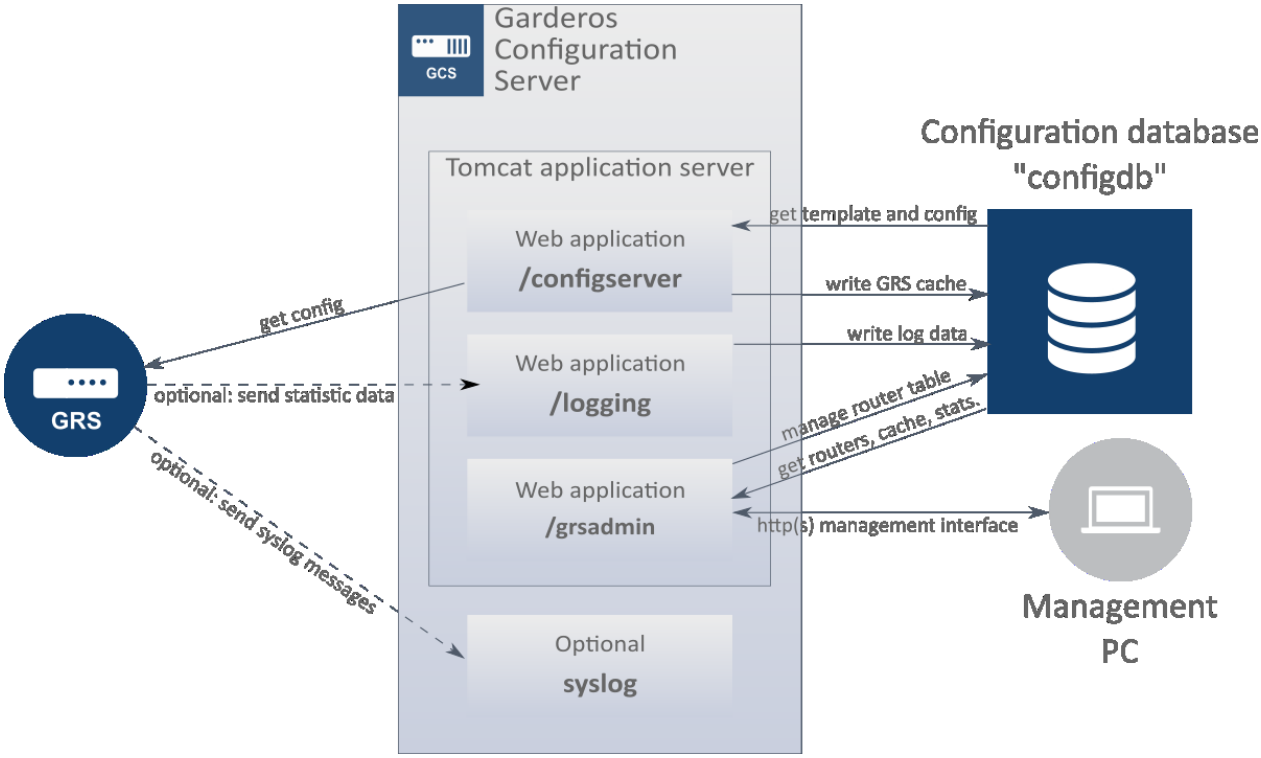
```
    exit 2
fi

workingdir=$(dirname $0)
workingfile=$(basename $0)
cd "$workingdir/.."
basedir=$(pwd)


log="$basedir/logs/lastsync_$logfile.txt"
date=$(date +%Y-%m-%d-%H-%M)
dirs=$(cat $basedir/etc/sync_dirs)

user="root"
rsyncopts="-avz --delete"
destip="172.16.26.200"

echo -e "Starting sync at $date with directory $1 \n" >> $log
for dir in $dirs
do
  if [[ "$dir" == "$dirParam" ]]; then
    echo -e "Syncing directory $dir to $destip" >> $log
    /usr/bin/rsync $rsyncopts -e 'ssh -p 22' $dir/ $user@$destip:$dir 2>&1 >> $log
    echo -e "Sync done." >> $log
  fi
done
```

HINT **Attention:** The event *IN_ALL_EVENTS* to more than one directory could cause a sync-loop and therefore high network and CPU load.

References

[1] https://dev.mysql.com/doc/refman/5.7/en/backup-methods.html
[2] https://www.digitalocean.com/community/tutorials/how-to-set-up-mysql-master-master-replication
[3] http://www.admin-magazin.de/Das-Heft/2017/02/MySQL-Replikation-mit-GTIDs/(offset)/2
[4] https://dev.mysql.com/doc/refman/8.0/en/caching-sha2-pluggable-authentication.html
[5] https://dev.mysql.com/doc/refman/5.7/en/replication-gtids.html

Addon: Logging web application



HTTP(S) logger

The standard logging method for GRS routers is, to use a central syslog server, where all devices send their log messages to. Usually the syslog only contains the messages from processes running on the router, but it can easily be extended to send and collect statistic data like traffic counters, system uptime, temperature and many more. Using syslog to collect statistical records has some disadvantages though, especially if there is a large number of GRS routers:

- Having many routers can cause a huge number of log records, which all must be processed. As syslog does not care if a message is received or not, sometimes messages might get lost without notice.
- It is a computationally intensive task to extract the relevant messages from all log messages.

To avoid some of these disadvantages Garderos provides an alternative way to collect statistic data from the managed routers:

1. A script running on each device collects the desired data.
2. The script sends the data in an HTTP(S) request to the logging web application.
3. A servlet in the logging web application receives these requests and stores the data in a database.
4. *GRS Admin* provides a page to show the collected data in your browser:
   Information -> Router Statistics.

The GRS part

This is an example, how to collect some statistic data on a GRS router and send them to the web logger:

```
#!/bin/sh

logserver="https://config1.garderos.com/logging/logger"


#collect statistic data
```

```
hostname=$( hostname )
temp=$( cli sh sys temp | grep "CPU" | cut -d ":" -f 2 )
uptime=$( cli sh sys info | grep Uptime | cut -d ":" -f 2-10 )
rxbytes=$( ifconfig ppp0 | grep bytes | sed "s#\(.*RX bytes:\)\([0-9]*\) \(.*\)#\2#" )
txbytes=$( ifconfig ppp0 | grep bytes | sed "s#\(.*TX bytes:\)\([0-9]*\) \(.*\)#\2#" )
down=$( echo $modem | sed "s#\(.*Down Rate: \)\(.*\)\( Kbps Up.*\)#\2#" )
up=$( echo $modem | sed "s#\(.*Up Rate: \)\(.*\)\( Kbps SNR.*\)#\2#" )
curl --data-urlencode "grsname=$hostname" --data-urlencode "cputemp=$temp" --data-urlencode "uptime=$uptime" \
     --data-urlencode "dslspeedup=$up" --data-urlencode "dslspeeddown=$down" \
     --data-urlencode "rxbytes=$rxbytes" --data-urlencode "txbytes=$txbytes" $logserver
```

Database logging

Storing the received data to the config database needs some configuration, as for every incoming *parameter_name* a corresponding database column in the table *logdata* is necessary.

Some helpful mysql statements:

```
# log in to the mysql database
mysql -u <db_user> -p logdata


# show existing tables in 'logdata'
mysql> SHOW COLUMNS IN logdata;
+--------------+-------------+------+-----+---------------------+-------+
| Field        | Type        | Null | Key | Default             | Extra |
+--------------+-------------+------+-----+---------------------+-------+
| timestamp    | timestamp   | NO   |     | 0000-00-00 00:00:00 |       |
| sourceip     | varchar(16) | YES  |     | NULL                |       |
| grsname      | varchar(32) | YES  |     | NULL                |       |
| cputemp      | varchar(32) | YES  |     | NULL                |       |
| uptime       | varchar(32) | YES  |     | NULL                |       |
| dslspeedup   | varchar(32) | YES  |     | NULL                |       |
| dslspeeddown | varchar(32) | YES  |     | NULL                |       |
| rxbytes      | varchar(32) | YES  |     | NULL                |       |
| txbytes      | varchar(32) | YES  |     | NULL                |       |
| wwan0signal  | varchar(32) | YES  |     | NULL                |       |
| wwan1signal  | varchar(32) | YES  |     | NULL                |       |
+--------------+-------------+------+-----+---------------------+-------+


# add a column
mysql> ALTER TABLE logdata ADD COLUMN test varchar(100);


# delete a column
mysql> ALTER TABLE logdata DROP test;


# show values in logdata
mysql> SELECT * FROM logdata WHERE timestamp > '2020-04-22';
+---------------------+-------------+---------+---------+---------+------------+--------------+---------+---------+-------------+-------------+
| timestamp           | sourceip    | grsname | cputemp | uptime  | dslspeedup | dslspeeddown | rxbytes | txbytes | wwan0signal | wwan1signal |
```

```
+---------------------+------------+---------+---------+---------+-----------+-------------+---------+---------+-----------+------------+
...
...
| 2020-04-22 14:04:59 | 10.10.12.44 | grs001  | 54.0    | 23m:38s | 1040      | 17112       | 55266   | 79673   | NULL      | NULL       |
...
```

Display logged data in GRS Admin

The page *Router statistics* within the web application *GRS Admin* is designed for displaying statistic data stored in the database by the logging application. This page can be enabled/disabled in the site configuration, also the values to be shown can be modified.

Configuration within the GRS Admin webapp

- Log in to the GRS Admin web application, then go to *Settings* (bottom right corner of the page)
- Turn the switch *Show Router Statistics* off or on and click the *Save* button.
- If showing the statistics is turned on, there are new settings:
  - *Statistic columns*: Which colums from database *logdata* shall be shown on the statistics page.
  - *Show Graphs*: For which colums of the database a graph shall be drawn.

Configuration using *web.xml* (the 'old' way)

The GRS Admin web application has a feature to display the data collected in the database. To enable this feature please edit the file */var/lib/tomcat*/webapps/grsadmin/WEB-INF/web.xml*. Enable (or uncomment) the following section:

```
    <context-param>
        <param-name>logdb</param-name>
        <param-value>true</param-value>
    </context-param>
```

Optional configuration parameters for this page:

- Which colums from database *logdata* shall be shown on the statistics page:

```
    <!-- these columns are shown on showlog_db page which shows collected values from database-->
    <context-param>
        <param-name>dbshowcolumns</param-name>
        <param-value>uptime,cputemp,wwansignal,wwannetwork,wwanband,phone,filehandles,revision</param-value>
    </context-param>
```

- For which colums of the database a graph shall be drawn:

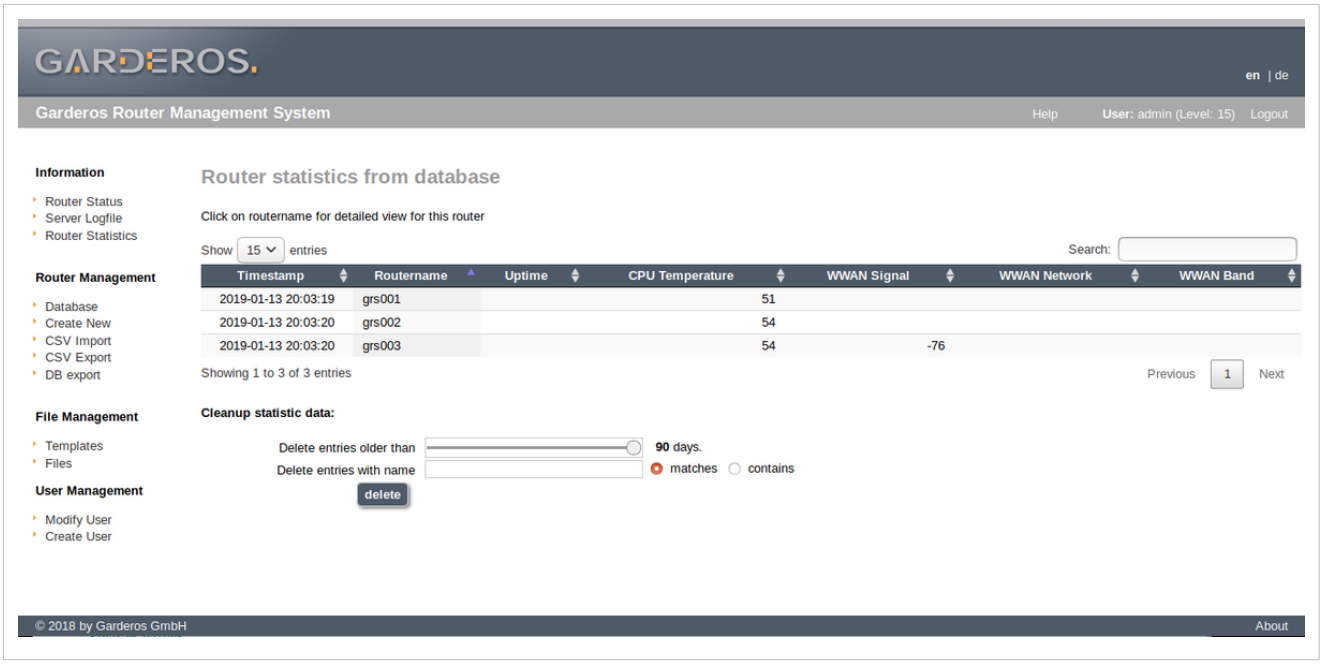
```
    <!-- for these columns a graph will be drawn on showlog_db page, this is only possible, if the column
        contains text which can be parsed to numbers. any non-digit charachter (like 'C', 'dBm') will be removed. -->
    <context-param>
        <param-name>dbshowgraphs</param-name>
        <param-value>cputemp,wwansignal</param-value>
    </context-param>
```

After the config change, the Tomcat application server has to be restarted:

```
service tomcat* restart
```

Viewing statistic data

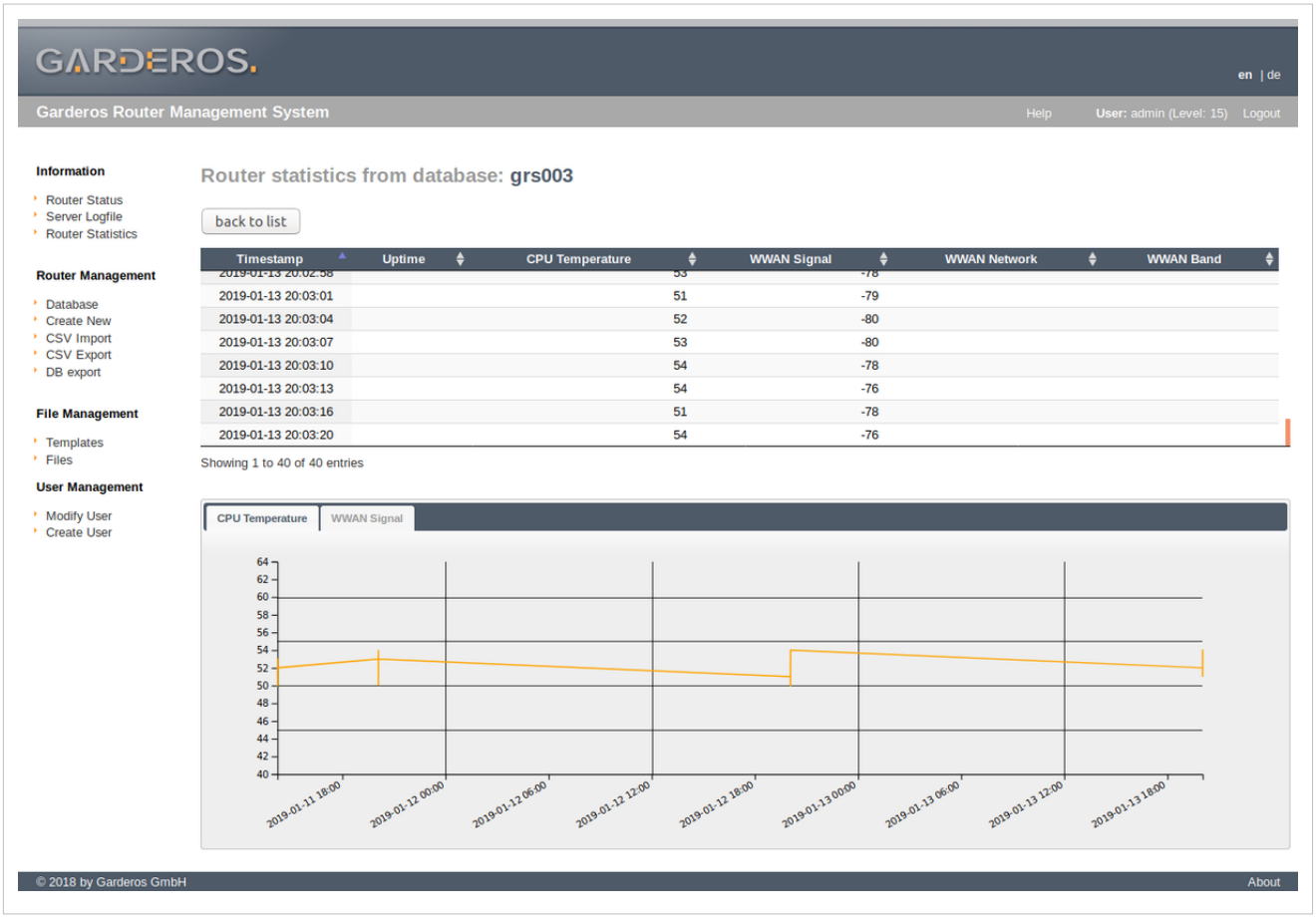
If statistics are enabled, the GRS Admin should contain the entry "Router statistics" in the menu:



The page shows the most recent entry of each router which is sending data to the database.

A click on one of the router names forwards to a page showing details for this router:

- The most recent 100 entries in the database
- A graph for some numeric values, e.g. the CPU temperature (see configuration above):



Database cleanup

Dependent on the configuration (interval sending statistic data) and number of routers, the database 'logdata' can grow rapidly. If the database contains millions of lines, the web page performance will be very poor. Removing old or unnecessary data can directly be done on the statistics page see the section **Cleanup statistic data**:

- Slider *Delete entries older than x days*, where 0<x<90
  Only removes entries older than the number of given days.
- Text input *Delete entries with name*:
  - *matches*: Deletes only routers where the name matches exactly the given argument, e.g. *grs001*
  - *contains*: Deletes all entries where the routername contains the specified string, e.g. *grs*. An empty text input machtes **all** routers.

HINT Please be aware, that this basic logging and data collection feature is limited to collect data from a few routers only and shall not replace any SNMP based monitoring solution like OpenNMS.

File logging

Data received within this HTTP request are appended to the mentioned file "as is", every single data is stored in the format "parameter_name = parameter_value".

The file location is defined in **logging/WEB-INF/web.xml** as parameter **logpath**. The default is */opt/configserver/log/grslog.txt*.

```
2016-04-22 14:04:59; IP=10.10.12.44; grsname = grs001; dslspeedup =
1040; uptime =  23m:38s; dslspeeddown = 17112; rxbytes = 55266; txbytes
 = 79673; cputemp =   54.0 C
```