Reference Manual

ufdbGuard REST API version 1.0.16

a fast URL classifier

Table of Contents

1 Introduction

The ufdbGuard REST API, or just "REST API", is a URL classifier. The REST API is implemented as a daemon process and answers to queries.

A URL classifier is a tool that given any URL is able to produce a list of URL categories that it belongs to. The ufdbGuard REST API is designed to be used together with the URL database of URLfilterDB. In addition to our database URL tables, one can load and use user-defined URL tables. Each URL table holds one URL category.

The ufdbGuard REST API is made with performance as the first priority. Therefore it is installed on a server in the client's data center to reduce latency, the database is loaded in memory, does not use compression and decompression of data, and since the data does not contain sensitive information there is no overhead of encryption.

The performance depends on the CPU. On a system with an Intel CPU with 4 cores and 8 threads the classifier can do 150,000 classifications per second.

Service providers and system integrators may register as a trial user at www.urlfilterdb.com to receive a 60-day trial license for the REST API and a URL database.

1.1 Copyright

The ufdbGuard REST API software suite is entirely developed and owned by URLfilterDB B.V. with all rights reserved. URLfilterDB B.V. holds the copyrights on the ufdbGuard REST API software suite.

The URL database is a commercial product and has a copyright by URLfilterDB. A license is required to use the URL database which is defined in The Terms of Contract document that can be downloaded at the website: www.urlfilterdb.com.

1.2 Latest Major Enhancements

Version 1.0.9 introduces support for database format 3.1 which has separate data structures for IPv4 and IPv6 addresses and supports IP subnets in a table.

Version 1.0.8 introduces support for database format 3.0 which makes the internal URL lookup 4 times faster. The overall API query time, however, will have a performance increase that it less than 4 because of the REST protocol overhead. Version 1.0.8 also introduces support for hugepages.

The file `CHANGELOG` in the directory `/usr/share/doc/ufdbrestd-`*`VERSION`* contains a list of all changes of all versions of the ufdbGuard REST API.

1.3 Support

The support conditions are negotiated and offered in a quote of URLfilterDB.

The free minimal support conditions for each client of the ufdbGuard REST API is:

1. email support for questions related to urgent matters of the ufdbGuard REST API and the URL database. Answers are on the same business day or morning of the next business day.
2. virtual meetings on business days with support staff. Virtual meetings must be requested at least 2 business days in advance.
3. 24x7 access to the partner portal where emergency changes to the URL database can be made.

The partner portal of URLfilterDB has URL https://partner.urlfilterdb.com and a username/password will be made available on the first purchase of a license. On this portal a client can modify the classification of URLs. This feature is only to be used in emergencies and should not be abused. URLfilterDB reserves the right to undo modifications of a client when it finds that a modification is not according its own classification guidelines. In this case a client may consider using a user-defined blacklist or whitelist.

1.4 Feedback

We welcome feedback from those who test our software. Feel free to send your questions and feedback to the support desk: support@urlfilterdb.com.

2 Prerequisites

The ufdbGuard REST API needs 10 GiB disk space and 4 GiB memory. In case that the default log file size is changed, the required disk space may increase. During database downloads at least 1 GiB free file system space is required in `TMPDIR` (default `/tmp`).

The ufdbGuard REST API is distributed as an RPM package for RHEL 7/8 and CentOS 7/8 and a Debian package for Ubuntu 18.04. In addition, the `wget`, `curl`, `tar` and `gzip` commands are required which are all included in most UNIX distributions.

The REST daemon answers queries on a single configurable TCP port on all available network interfaces. The required capacity of the NICs depends on the peak query traffic. The NIC that has the default route to the internet has no high performance requirement and may be as low as 1 gbps.

The URL database is regularly downloaded via the default route to the internet. The files with uncategorised URLs and statistics are uploaded via the default route to the internet. Both processes connect to `updates.urlfilterdb.com` using port 443 and optionally port 80. `updates.urlfilterdb.com` currently has 2 IPv4 and 2 IPv6 addresses.

In case that no direct internet access is available on the system where the REST daemon runs, an HTTPS proxy must be available in order to download the URL database and upload uncategorised URLs.

3 REST API Processes

3.1 ufdbrestd

The REST API daemon is multithreaded and uses the pthreads library. The name of the daemon is `ufdbrestd`. It usually starts automatically when the system is booted and under normal circumstances is always running.

The `ufdbrestd` daemon accepts the following command line options:

`-c` *`file`*	specify the configuration file.
`-d`	enable debug mode. Multiple `-d` options increase the debug level.
`-v`	show version number and exit.
`-h`	show help.

When the ufdbrestd process is started it runs until terminated by the administrator.

The ufdbrestd daemon rereads the configuration file and reloads the URL database when it receives a HUP signal.

The ufdbrestd daemon records every 60 minutes the unclassified URLs to the servers of URLfilterDB so that they can be classified and included in a future version of the URL database. Depending on a configuration option, the uncategorised URLs are stored in a file (to be uploaded by a client process using `curl`) or are uploaded directly to the website `updates.urlfilterdb.com`. This upload does not disturb the queries of REST clients.

3.2 ufdbRESTupdate

`ufdbRESTupdate` is a script that downloads a fresh copy of the URL database and sends a HUP signal to ufdbrestd to inform it to reload the URL database.

The `ufdbRESTupdate` script accepts the following command line option:
`-v` be verbose.

It is recommended to have a crontab job to download a fresh URL database 1-4 times per day. Note that the daemon refuses to load the URL database if it is older than 28 days.

Starting with version 1.0.9, `ufdbRESTupdate` calls the binary executable `ufdbRESTdl` which does the actual download. `ufdbRESTdl` replaces `wget` and has support for an encrypted password.

See section 6.5 for more information.

3.3 ufdbRESTupload

The REST daemon maintains a list of uncategorised URLs which, after being uploaded to updates.urlfilterdb.com, are used to extend the URL database. The REST daemon can upload these URLs directly or produce *upload files*. The daemon produces upload files if the parameter upload-with-files is on.

The upload files can be uploaded with the script `ufdbRESTupload`.

If upload files are produced, it is recommended to have a crontab job that runs 15 minutes after the crontab jobs for `ufdbRESTupdate`.

4 REST Queries

REST clients can query the REST server by sending appropriately formatted HTTP commands to the server.

4.1 HTTP/1.1

The daemon expect that clients use HTTP/1.1 for communication. A request shall have a HTTP request line followed by a small set of HTTP headers optionally followed by a request body. The daemon only supports the HTTP request methods OPTIONS, GET and POST.

The daemon expects at least one of the following headers: `Content-Length` with a zero or non-zero value, or `Transfer-Encoding` with `chunked` encoding. All other headers are optional.

The optional headers are `Connection`, `User-Agent`, and a user-defined *send-back* header (see parameter `send-back-header` in section 7.1). All other headers are ignored.

The HTTP request line and headers must be terminated with CRLF. The HTTP headers must be followed by one empty line that only consists of CRLF.

The daemon supports HTTP pipelining.

4.2 REST Server Configuration

4.2.1 OPTIONS

The REST client can retrieve server characteristics by sending an `OPTIONS` request:

Example (substitute *hostname*):

```
OPTIONS http://hostname:13931/options.json HTTP/1.1
```

The REST server usually replies with a `200 OK`

Example:

```
HTTP/1.1 200 OK
{ "restserverapiversion": "1.0",
  "reloading": "no",
  "databasetimestamp": "20201230.0915",
  "secondssincelastload": "64800",
  "maxconnections": "12",
  "categories": "ads,adult,altdns,...,webtv",
  "state": "normal",
  "numfatalerrors": "0",
  "numerrors": "0",
  "timeout": "20" }
```

In the above example the REST server has replied that it uses API version 1.0, stated that it is not reloading the database, has given the database timestamp and seconds since the most recent database load, stated that it supports a maximum of 12 connections (and hence maximum performance can only be achieved with 12 parallel streams), show which URL categories it supports, has given its internal state which should be "normal", has zero fatal errors and zero non-fatal errors.

The possible values for "reloading" are "no", "soon" (in max 10 seconds) and "yes". Note that before version 1.0.8 a database reload took 3-5 seconds but since 1.0.8 the reload time is almost instantaneous.

Values for "state" that are different from "normal" are always a sign of serious error conditions and indicate that the REST server is not or not fully capable of performing its task.

If "numfatalerrors" is not zero, an operator should immediately check the logs. If "numerrors" is not zero, an operator should also check the logs but with less urgency.

The "databasetimestamp" should be verified since the daemon refuses to use database tables that are over 28 days old.

The server may also reply with a `4xx` or `5xx` code.

Example:

```
HTTP/1.1 503 REST SERVER IS EXITING
```

The OPTIONS request can be made from any application that supports the HTTP protocol and can also be made with the `curl` command.

Example:

```
$ curl --request OPTIONS http://localhost:13931/options.json
```

4.2.2 API Version

The REST client can retrieve the API version by sending the *API version request.*

Example (substitute *hostname*):

```
GET http://hostname:13931/api/version HTTP/1.1
```

The REST server responds with:

```
{ "daemon-version": "1.x.y", "api-version": "1.0" }
```

4.3 URL Classification

4.3.1 Retrieve Classification for a Single URL

To request the categories of a URL the REST client sends the following REST request:

```
GET http://<FQDN>[:<port>]/useparams/yes/categories.json?url=<ESCAPED-URL> HTTP/1.1
```

Note that the URL parameter must be escaped as per RFC 3986.

Example:

To get the URL categories of the URL "http://www.example.com/foo.php?bar=xyz" :

```
GET http://hostname:13931/useparams/yes/categories.json?url=http%3A%2F
%2Fwww.example.com%2Ffoo.php%3Fbar%3Dxyz HTTP/1.1
```

Alternatively, the REST client may send an escaped URL using the following:

```
GET http://<FQDN>[:<port>]/useparams/no/url/<ESCAPED-URL> HTTP/1.1
```

Example:

```
GET http://hostname:13931/useparams/no/url/http%3A%2F%2Fwww.example.com
%2Ffoo.php%3Fbar%3Dxyz
```

The REST server responds with a 200 OK:

Example:

```
HTTP/1.1 200 OK
{ "url": "www.example.com", "categories": [ "news", "entertainment" ],
  "reloading": "no" }
```

The possible values for "reloading" are "no", "soon" (in max 10 seconds) and "yes". Note that before version 1.0.8 a database reload could take 3-5 seconds while starting with version 1.0.8 the reload is almost instantaneous. In the very unlikely event that the REST daemon replies with "reloading":"yes" it is recommended to resend the query.

In case that the URL did not belong to any category, the REST server responds with an empty list.

Example:

```
HTTP/1.1 200 OK
{ "url": "www.example.net", "categories": [ ], "reloading": "no" }
```

The HTTP GET request can be made from any application that supports the HTTP protocol and can also be made with the `curl` command.

Example:

```
curl http://localhost:13931/useparams/yes/categories.json?url=http%3A%2F%2Fwww.cnn.com
```

4.3.2 Retrieve Classification for a Batch of URLs

The REST client can also request categorisation for a list of URLs (max 1024 URLs). The maximum size of the REST request is 128 KB.

Example:

```
POST http://<FQDN>[:port]/useparams/no/geturllist.json HTTP/1.1
```

```
X-HTTP-Method: GET
```
with body content:
```
{ "urls": [ "<HTTP-ESCAPED-URL>", "<HTTP-ESCAPED-URL>", ... ] }
```

Example client request (substitute *hostname*):
```
POST http://hostname:13931/useparams/no/geturllist.json HTTP/1.1
```
body content:
```
{ "urls": [ "http%3A%2F%2Fwww.foo.bar%2F", "http%3A%2F%2Fwww.cnn.com" ] }
```

The REST server replies with
```
   { "reloading": "status",
     "urlcategories": [
      { "url": "<JSON-ESCAPED-URL>", "categories": [ <LIST-OF-CATEGORIES> ] },
      { "url": "<JSON-ESCAPED-URL>", "categories": [ <LIST-OF-CATEGORIES> ] },
      ...
   ] }
```

Example server reply:
```
   { "reloading": "no",
     "urlcategories": [
      { "url": "www.foo.bar/", "categories": [ ] },
      { "url": "www.cnn.com", "categories": [ "news" ] },
      ...
   ] }
```

5 Load Balancing Support

Each REST daemon is an independent process which has no knowledge of other REST daemons. A load balancer can be used in front of a set of REST daemons without any modification to the REST daemon configuration.

With version 1.0.8 and above the daemon will only very rarely respond with "reloading":"yes" and an empty set of URL categories. When this happens, it is recommended that the client re-sends the query.

6 Software Installation

6.1 Upgrading from a Previous Version

A previous version can simply be overwritten by a new version.
When an upgrade is performed, it is recommended to perform all installation steps including the last step, retrieval of database update in section 6.5.

NOTE: on systems that support configuration files in `/etc/sysconfig`, a new configuration file `/etc/sysconfig/ufdbrest` is used. During the installation a message reminds the administrator of the new file.

6.2 Install Software

The package can now be installed. Note that you must be `root`.

```
# yum install ufdbrestd-1.0.16-CentOS7.x86_64.rpm
```

or

```
# dpkg -i ufdbrestd-1.0.16.amd64.deb
```

The package installation may create user `ufdbrest` and group `ufdbrest` if they do not exist, and various files in directories specified in the next section.

NOTE: the superuser should set a password for the user `ufdbrest`.

6.3 File Locations

The ufdbGuard REST API files and its location for RHEL 7 and CentOS 7 are in the following table.

ufdbrestd	/usr/bin
ufdbRESTupdate	/usr/bin
ufdbRESTGenTable	/usr/bin
ufdbrestd.conf	/etc
ufdbrest	/etc/rc.d/init.d
ufdbrest	RHEL/CentOS: /etc/sysconfig Ubuntu/Debian: /etc/default [1]
ufdbrest	/etc/cron.d
URL database files	/var/ufdbrestd/blacklists
license status file	/var/ufdbrestd/blacklists/license-status
log files	/var/ufdbrestd/log
upload files	/var/ufdbrestd/upload
pid file	/var/ufdbrestd/run

log files: log files are rotated and the maximum size of an individual log file is set with the option `max-logfile-size` (default is 1 GB). The maximum number of log files (including perpetual rotation) that the daemon creates is 9.

upload files: the daemon generates files with statistics and uncategorised URLs if the option `upload-with-files` is `on`. The operator is responsible for scheduling with little delay the upload of these files to `updates.urlfilterdb.com`.

[1] This manual assumes installation on an RHEL/CentOS system. If you install the package on an Ubuntu/Debian system, substitute /etc/sysconfig by /etc/default where appropriate.

6.4 User Account

The ufdbrestd daemon can run as a regular unprivileged user if it is configure to use an unprivileged port, i.e. a port number higher or equal to 1024. If the port number is lower than 1024, ufdbrestd must be run as root.

6.5 Get Daily Updates

The script `ufdbRESTupdate` takes care of downloading a new version of the URL database. It is the responsibility of the system integrator to integrate `ufdbRESTupdate` in its processes. After `ufdbRESTupdate` has downloaded a fresh URL database the REST daemon receives a HUP signal to load the new URL database.

Depending on the configuration option `upload-with-files`, the daemon either uploads counters and uncategorised URLs to https://updates.urlfilterdb.com, or writes a file which must be uploaded with `curl`.

The `ufdbRESTupdate` script needs the username and password that you received when the (trial) license was received which can be defined in a system configuration file which resides in a system-dependent directory (`/etc/sysconfig` or `/etc/default`).

```
$ vi /etc/sysconfig/ufdbrest
...
DOWNLOAD_USER=lic99999
DOWNLOAD_PASSWORD=aa22bbpp
```

Users that evaluate the URL database may use the demoXX username and password.

Test the `ufdbRESTupdate` script with the verbose option:

```
$ ufdbRESTupdate -v
```

The output should be similar to:

```
reading parameters from /etc/sysconfig/ufdbrest ...
ufdbRESTdl: going to download
https://updates.urlfilterdb.com/licensed/databases/ufdbguard/3.1/blacklists-
latest.tar.gz
to /tmp/urlfilterdb-latest.tar.gz
ufdbRESTdl: downloaded URL database is stored in /tmp/urlfilterdb-latest.tar.gz
(155306200 bytes)
ufdbRESTdl: downloaded license status is stored in /var/ufdbrest/blacklists/license-
status (30 bytes)
-rw-rw-r-- 1 ufdbrest ufdbrest 155306200 Apr 10 16:36 /tmp/urlfilterdb-latest.tar.gz
/tmp/urlfilterdb-latest.tar.gz successfully unpacked in /var/ufdbrest/blacklists
license status: OK: expiry date is 01-01-2028
Going to send HUP signal to ufdbrestd process (pid 1212) to signal it to reload the URL
database
```

6.5.1 ufdbRESTupdate

`ufdbRESTupdate` downloads the URL database and needs access to the servers of URLfilterDB. Firewall rules may need to be modified to provide access to `updates.urlfilterdb.com`.

A proxy can be used to download the URL database: modify `/etc/sysconfig/ufdbrest` and assign appropriate values to the variables `http_proxy` and `https_proxy`. If the proxy requires authentication, also assign appropriate values to `PROXY_USER` and `PROXY_PASSWORD`.

The URL database resides in a compressed tar file which is downloaded to the directory `$TMPDIR` (default is `/tmp`) and then untarred in the directory `$BLACKLIST_DIR`. Both variables can be set in `/etc/sysconfig/ufdbrest`.

6.5.2 Exit Codes of ufdbRESTupdate

The `ufdbRESTupdate` script has exit code 0 when the download is successful and a non-zero code in case of an issue. In case of a non-zero exit code, the script prints the exit code and explanation on stdout. In case of an error, the script also logs a message to the system log. The script uses the exit codes in the following table.

exit code	*explanation*
0	all is well
1	ufdbRESTdl option error
2	ufdbRESTdl cannot retrieve password from <file>
3	ufdbRESTdl cannot create database file <file>
4	ufdbRESTdl TLS/SSL or download error
5	ufdbRESTdl could not sync database to disk
6	downloaded database is too small
7	ufdbRESTdl cannot create license status file <file>
8	ufdbRESTdl cannot download license status
9	downloaded license status file is too small
11	The configuration file `/etc/sysconfig/ufdbrest` cannot be read
12	The variable `DOWNLOAD_USER` is not set.
13	The download failed with unknown reason
14	The temporary directory (`TMPDIR`) does not exist or is not writable
15	The URL database top directory (`BLACKLIST_DIR`) does not exist or is not writable
16	Cannot remove temporary download file – check permissions of directory `$TMPDIR`
22	license expiration warning: less than 2 months to renew license
23	license expired: a license renewal is required immediately
61-80	exit code of `tar` + 60. `tar` unpacks the downloaded URL database. There may be an issue with file system space.
127	Could not find ufdbRESTdl in PATH

In case of an error, it is advised to run `ufdbRESTupdate -v` from the command line to have more feedback about what is going wrong.

6.5.3 License Status

The content of `license-status` file (see 6.3 for the location) is one of:

OK: expiry date is <date>
WARNING: license expires in <n> days
EXPIRED: license expired <n> day(s) ago
EXPIRED: license terminated since <date>

When the license is two months before its termination date, the `ufdbRESTupdate` script also warns with a different exit code (see previous section).

6.6 Upload Files

As for the `ufdbRESTupdate` script, a proxy can be used to download the URL database: modify `/etc/sysconfig/ufdbrest` and assign appropriate values to the variables `http_proxy` and `https_proxy`. If the proxy requires authentication, also assign appropriate values to `PROXY_USER` and `PROXY_PASSWORD`.

7 Configuration

The REST server reads its configuration from the configuration file that is specified with the `-c` command line option. The whole configuration as how ufdbrestd understands it, is logged in the logfile.

The configuration file has 2 sections: parameters and categories.

7.1 Parameters

The following parameters can be defined.

Define where the logfile `ufdbrestd.log` will be created:

```
logdir "directory"
```

Define the maximum log file size. Must be in the range 2 MB – 2 GB. Default is 1 GB. When the log file reaches the maximum log file size, the log file will be rotated (rename ufdbrestd.log to ufdbrestd.log.1) where a maximum of 8 files are kept available.

```
max-logfile-size NNN [mb]
```

Define the directory where the URL database resides:

```
dbhome "directory"
```

Define the port where ufdbrestd listens to, the default is 13931:

```
port NNN
```

Specify if all requests must be logged:

```
option log-requests on|off
```

Specify if the parameters of a URL and a long path of a URL must be logged:

```
option ufdb-show-url-details short|long|verylong
```

Specify if URLs that are not part of the current URL database must be uploaded. This option should always be `on`.

```
option analyse-uncategorised-urls  on|off
```

Specify the time that a query response may be cached in seconds (default 6 hours):

```
option cache-max-age <max-age-in-seconds>
```

This option is used to generate a reply header `Cache-Control` with `max-age` set to the specified value.

Specify how ufdbrestd responds to URL classification requests while a new configuration is being loaded. With stall-when-reloading set, classification requests will stall until the new configuration is finished reloading. This depends on hardware and is usually less than 10 seconds. Default: on.

```
option stall-when-reloading on|off
```

Specify how a configuration refresh is executed. When fast-refresh is on, ufdbrestd loads a new configuration while the current configuration is still being used and then swaps the current for the new in a split second. So when `fast-refresh` and `stall-when-reloading` are both set, ufdbrestd will always reply to a URL classification request, even in the middle of a reconfiguration. The trade off is that with the fast-refresh the memory usage is doubled. Default: on.

```
option fast-refresh on|off
```

Specify the field name of a request header that the daemon must send back in its reply. The daemon scans for this field in the headers of all requests and copies it to the reply that it sends back to the client. The maximum length for the header ID is 60 bytes and the maximum length for its value is also 60 bytes.

```
option send-back-header "<field-name>"
```

Example:

```
option send-back-header "X-Query-ID"
```

Specify if Linux may be hinted to use transparent hugepages using `madvise(2)`. Default: on.

```
option madvise-hugepages on|off
```

Specify the number of worker threads. Default: 4. This must be a number between 2 and 96:

```
num-workers 8
```

Note that the performance depends much on a correct configuration of the number of threads. When there is no pipelining of requests, the threads in a REST server are usually not 100% busy since they wait for requests and are also idle when a reply is being sent. Therefore the number of threads should be 1.2 to 1.5 times the number of available hardware threads. So on a CPU with 4 cores and 8 hardware threads, the maximum number of worker threads is between 9 and 12. If pipelining is used, the maximum number of worker threads should be close to the number of hardware threads.

Specify the maximum number of connections. This must be a number between 2 and 512.

```
max-connections 16
```

Specify whether the daemon sends counters and uncategorised URLs directly to `updates.urlfilterdb.com` (default) or writes the data to a file. In case that the data is uploaded directly, any firewall between the server and `updates.urlfilterdb.com` must allow traffic. In case that a file is written, it is the responsibility of the client to make sure that the file is uploaded.

```
option upload-with-files on|off
```

Specify the directory where upload files are stored (only used if `upload-with-files` is `on`):

```
upload-file-directory "<directory>"
```

Specify whether messages are logged to the system log. Messages are always logged in the log files and optionally sent to the system log (syslog). Default is `off`. If set to `on`, all messages are also sent to the system log. If set to `syslog-fatal`, only fatal errors are sent to the system log.

```
option use-syslog off|on|syslog-fatal
```

Specify the debug level. This must be a number between 0 and 8:

```
option debug-level 0
```

Specify if requests must be timed and logged:

```
option debug-timings on|off
```

Specify if I/O must be debugged:

```
option debug-basic-io on|off
```

7.2 Categories

In the second section all categories of the URL database are defined. The default configuration file already includes the standard categories.

Each category has a definition where the URL table is defined. The path of the domainlist points to a database table with suffix `.ufdb`. The path is either absolute or relative to the dbhome parameter.

Example:

```
category "safe" {
   domainlist      "safe/domains"
}
```

7.3 Tasks

On a client system, the following tasks must be implemented.

1. start/stop the ufdbGuard REST daemon. By default this is done by `/etc/init.d/ufdbrest` script.
2. run `ufdbRESTupdate` 1-4 times per 24 hours. This script download a new version of the URL database and sends a HUP signal to the REST daemon which trigger the daemon to load the new database.
3. if option `upload-with-files` is `on`, the generated files with uncategorised URLs must be uploaded to the servers of URLfilterDB in a timely manner. See section 7.4 how to upload files.

7.4 Upload files with uncategorised URLs

If option `upload-with-files` is `on`, the REST daemon generates files with statistics and uncategorised URLs. The files are stored in the directory defined by the `upload-file-directory` parameter.

The command to upload a single file is as follows:

```
curl -s -S --user-agent "UFDB-<Vendor>-<Version>" --request PUT  \
   --upload-file "<filename>"  \
   https://updates.urlfilterdb.com/cgi-bin/uncatmeta.pl
```

Replace <Vendor>, <Version> and <filename> with appropriate values.

The webserver should respond to the above curl command with the text "OK". The upload failed if no "OK" is given. The server may respond with one of the following lines:

```
error: REQUEST_METHOD is <method>
error: parsing failed
```

NOTE: in case that an HTTP proxy with user authentication is used, the above curl command also needs the option `--proxy-user` *`<username:password>`* where the username and password should be the same as PROXY_USER and PROXY_PASSWORD in `/etc/sysconfig/ufdbrest`.

8 Monitoring

The REST daemon can be monitored to be informed if the daemon is running and to retrieve a health status indicator.

The REST daemon writes informational messages, errors (prefixed by "ERROR:") and fatal errors (prefixed by "FATAL ERROR:") to the log files. Log files are automatically rotated (maximum 8 files with additional suffix `.1` ... `.8`) when the log file reaches the maximum configured log file size (see `maximum log file size` parameter).

The health status of the REST daemon can be queried by sending an OPTIONS request to the REST daemon (see section 4.2.1). It is recommended to inspect the values of the attributes databasetimestamp, secondssincelastload, state, numfatalerrors and numerrors of the reply message.

A monitoring agent can monitor the REST daemon log files, monitor the system log and regularly query the health status

9 User-defined URL tables

In cases where additions or exceptions to the categories of URLfilterDB are desired, an administrator can define user-defined URL categories.

9.1 Create a URL Table

In this section a URL table called *alwaysallow* is created which contains a small set of URLs.

Edit the file that contains the URLs of the new category:

```
$ cd /usr/local/ufdbrestd
$ vi blacklists/alwaysallow/domains
```

Add the appropriate URLs and <u>always remove a leading</u> `www.`:

```
yourcompany.com
news.google.com
google.com/news
```

In case that many URLs have a leading `www.`, one may use the `-W` option to remove the `www.` prefix automatically.

The ufdbGuard REST API only uses proprietary database files, so generate a `.ufdb` database file from the ASCII files with `ufdbRESTGenTable`:

```
$ cd /usr/local/ufdbrestd
$ bin/ufdbRESTGenTable -W -n -t alwaysallow -d blacklists/alwaysallow/domains
```

The above command generates the file `blacklists/alwaysallow/domains.ufdb` and should be invoked each time that the domains file is changed. The `-W` option removes the initial `www.` from all URLs which is highly recommended since ufdbrestd removes `www.` from URLs before it does a database query. The `-n` option specifies that the URL table does not have to be encrypted.

`ufdbRESTGenTable` does a sanity check for the validity of the URLs.

Then configure the category by editing the `ufdbrestd.conf` file and uncomment the category definition for *alwaysallow*. The configuration file should have the following lines:

```
category alwaysallow
{
   domainlist alwaysallow/domains
}
```

Finally send a HUP signal the daemon to reload the URL database.

```
# kill -HUP pid
```

User-generated database files are accepted by the REST daemon even if they are older than 28 days. The maximum age requirement of 28 days only holds for database tables generated by URLfilterDB.

9.2 How URLs are Matched against the URL Database

The URL database lookup uses an algorithm to match a URL against the entries in the tables of the URL database. The algorithm uses the following logic.

1. Port numbers and embedded usernames and passwords are ignored. So a URL like john:secret@example.com:8080/foo is simplified to example.com/foo.
2. If a URL table contains an entry with a domainname example.com it matches all URLs that contain example.com including subdomains, and matches URLs like example.com/foo.html, www.example.com and secure.example.com.
3. If a URL table contains an entry with a domainname with a "pipe tag", e.g. |.example.com, it matches all URLs that contain the domain example.com but not subdomains (*). This entry matches URLs like example.com/foo and www.example.com.
4. If a URL table contains an entry with a domainname and a path, e.g. "example.com/foobar" it matches all URLs that have the domain example.com (but not subdomains) and have a URL path that *starts* with the given URL path, so its matches www.example.com/foobar.html and does *not* match sub.example.com/foobar.
5. If a URL table contains an entry with a domainname, a path and a pipe tag, e.g. example.com/foobar|, it matches all URLs that have the domain example.com (but not subdomains) and have a URL path *equal* to the given path, so it matches www.example.com/foobar and does *not* match www.example.com/foobar.html.
6. If a URL table has an entry with parameters, the URL is matched if it contains all parameters of the table entry *in any order*. For example, if a table contains example.com/watch?p1=foo, the

URLs www.example.com/watch?p1=foo and www.example.com/watch?p0=x&p1=foo&p2=bar are matched.

(*) "www" and "www0"..."www99" are not considered subdomains.

10 Performance Tuning

10.1 Linux 2.6+ Performance

There are various sources for Linux system tuning and this section is not a replacement. This section contains recommendation for tuning Linux systems that runs ufdbrestd only.

10.1.1 Optimize System Memory Usage

Linux has various settings that control allocation of memory and how the system behaves when real memory gets scarce. Below are given the setting that gave satisfactory system behavior on our test systems.

Add to `/etc/sysctl.conf` the following lines:

```
# swappiness can have a value of between 0 and 100
# swappiness=0 tells the kernel to avoid swapping processes out of physical
# memory for as long as possible
# swappiness=100 tells the kernel to aggressively swap processes out of
# physical memory and move them to swap cache
# default: 60
# For an application server we use a less aggressive setting of 15.
vm.swappiness=15

# VFS cache pressure:
# default: 100
# With values lower than 100, the data cache is reduced more and the
# inode cache preserved
vm.vfs_cache_pressure=50
```

Run the following command to make the new configuration active:

```
# sysctl -p /etc/sysctl.conf
```

10.1.2 Bind Threads

Total performance is increased by a few percent if threads are each bound to a different core/thread

10.1.3 Use Hugepages

Total performance is increased by a few percent if the application can use hugepages.

Version 1.0.8 introduced support for database format 3.0. When this database format is used, the daemon may hint Linux to use transparent hugepages using the OS call `madvise(2)` for each table that occupies more than 1.6 MB memory. The default configuration of Linux is to transparently try to use hugepages when advised by `madvise(2)`.

11 URL Categories

The URL database of URLfilterDB uses the following URL categories. Some categories have subcategories. URLs in a subcategory are also in the parent category.

Ads

Websites with advertisements, traffic trackers, user behaviour analysis and web page counters.

Parkeddomain

Websites that are parked. Usually parked domains are expired domains or domains for sale and managed by domain brokers. Many parked sites have ads and some domain brokers use ad brokers that redirect users to adult, gambling or scam sites.

P2P

P2P stands for point-to-point file sharing. The P2P category contains websites that can be used directly or indirectly to upload, download and share files. Most P2P sites have copies of movies, adult content, malware, warez and entertainment, and much of this content violates copyright.

Proxies

Sites that can be used to download content of other sites, URL rewriting sites and VPNs. Proxies are commonly used in an attempt to circumvent a URL filter and it is recommended to always block proxies.

Adult

Websites suitable for adults only (not only sexual content).

Malware

Websites that contain or redirect to viruses or malware.
NOTE: this URL category is *not* a replacement for a antivirus tool.

Warez

Websites with illegal software, illegal software codes, hacker’s sites, warez and cracks.

Toolbars

Websites for toolbars of browsers. A toolbar is an extension to a web browser that may violate your privacy or make private files public.

Illegal

Websites explaining how to perform Illegal activities.

Arms

Sites with firearms and toys that look like firearms.

Violence

Websites about violent behavior.

Gambling

Websites offering gambling opportunities.

Drugs
Websites about hard drugs.

Webmail
Email accessible with a web browser. Webmail of business sites is not included while webmail of ISPs is included in this category.

Dating
Websites about love, dating, romantic poetry, and friendship.

Chat
Websites to use IRC and chat. Subcategories exist for AIM, Ebuddy, Facebook Chat, Google Talk, MSN Messenger, Oovoo, Skype and Yahoo Chat.

Forum
Websites where people exchange non-business information in a forum.

Private
Blogs and sites of private persons.

Webtv
sites with a audiovisual streams or television-like streams.

Webradio
sites with a music streams or radio-like streams.

Dailymotion
videos of dailymotion

Vimeo
videos of Vimeo

Youtube
videos of Youtube

Audio-Video
Audio and video streams.

Sports
Websites related to sports including sports sections of news sites, fans of sports, sites about actively doing a sport.

Finance
Websites of banks, insurance companies, stock markets and stock brokers.

Jobs
Websites about and for job applications.

Games
Websites to play games and information about gaming.

Entertainment
Entertainment, lifestyle, hobby, arts, museums, fashion, electronic cards, magazines, horoscopes, desktop wallpapers, clip art, photos, portals, events, fan sites, baby-related, child sites, other sites for interest of private persons that are not related to business.

Food
Websites of restaurants and sites with recipes. Fast food chains, however, are part of the category *shops*.

Religion
Websites related to any religion.

Shops
Websites with shops, price comparisons, and auctions aimed at consumers (b2b is excluded).

Travel
Websites about travel agencies, airliners, tourism sites, hotels, holiday resorts.

News
Websites providing news and opinions.

External Applications
Free web-based document editors, spreadsheet applications, desktops, groupware, etc. where "internal" documents can be stored on external servers.

Social Networks
Sites that focuses on building and reflecting of social networks or social relations among people. Subcategories exist for Badoo, Facebook and Twitter.

DNSoverHTTPS
IP addresses and domainnames of services for DNS lookups over HTTPS. This category has also a text file iplist with all IP addresses which can be used to configure a firewall.

Alternate DNS
There is a collection of alternative DNS systems with alternative TLDs like .coin, .libre, .bazar and .geek. See https://www.opennic.org for more information.
The DNS servers use ports 53, 443 and alternate ports like 8443, 5335 and 5353. This category has also a text file iplist with all IP addresses which can be used to configure a firewall.

Dynaddress
Websites with a dynamic IP address.

Extappl
Websites that deal with off-line documents and data.

Education
Websites of schools, universities and educational institutes.

Health
Websites of doctors, clinics, diseases and other health-related sites.

Qmovies
Websites which contain or link to movies with probable copyright infringement.

Searchengine
URLs used by search engines.

Checked
URLs that are verified by URLfilterDB not to be part of any other category. This category contains business sites, governmental sites and useful sites for the general public. This URL category is also used by the ufdbGuard API to track uncategorized URLs and should always be loaded.

The classification rules for URL database are based on *user intent* and classification is from the point of view of a business. So, a website that has a business use, is by default part of the category "checked". For example, access to a website to sell equipment for building constructors is in category "checked" and hence is not part of the category "shops". Also governmental sites, and all sites for basic human needs like electricity and water are in the URL category "checked".

The nature of the content is more important than the strict definition, so an advertisement with a nude person is classified as adult rather than advertisement (although may be included in both categories), and a forum about games is classified as games.

The general impression is also taken into account when a site is categorized. For example, most buyers at `ebay.com` are consumers rather than business users and therefore `ebay.com` is considered a shop for consumers and part of the *shops* category.

URLs may be part of one or more categories, e.g. `www.usatoday.com` is *news* while `www.usatoday.com/sport` is both *news* and *sports*.

12 Error Messages

The error messages in the following sections may contain variable texts where `%s` represents character string, `%d` represents an integer and `<file>` represents a string with a filename.

Many errors listed in the next sections will never occur or will occur very rarely. To better understand error messages, it is always recommended to try to correlate errors of the REST daemon with errors and messages from the direct environment of the REST daemon. The direct environment includes but is not limited to file storage, virtual machine management, network management and applications that connect to the REST daemon.

12.1 HTTP Errors

The REST daemon usually responds with an HTTP 200 code indicating that all is ok and that the message body contains a reply to the request. In case that something is wrong, the REST daemon may reply with a 4xx or 5xx HTTP status code. Below are the errors the the daemon may send.

`500 REST SERVER HAS A FATAL ERROR.  See UFDBRESTD server log for details.`
A fatal condition appeared and a person has to inspect the log file of the daemon.

`503 REST SERVER IS EXITING.`
This daemon instance is exiting and will be dead soon. Most likely because the daemon was sent a TERM signal.

`411 CONTENT-LENGTH IS REQUIRED WHEN CONTENT IS OVER %d KB`
The daemon cannot accept chunk-encoded content that is larger than 128 KiB.

`408 READ TIMEOUT IS 20 SECONDS`
The client sent a partial request and the daemon timed out after 20 seconds trying to read the rest of the message.

`405 METHOD NOT ALLOWED`
The daemon only supports GET, POST and OPTIONS methods.

`400 PROTOCOL ERROR: %s`
This is a parser error where `%s` includes a description of what went wrong. The descriptions are:

```
the REST request has no path
the REST request has an illegal value or has no terminating slash for 'useparams'.
received 'xxx'
the REST request must have '/useparams/no/ with geturllist.json'.  received '%s'
the REST request must have '/useparams/yes/' with 'categories.json'.  received '%s'
the REST request has no '?' after 'categories.json'.  received '%s'
the REST request has no '?url=' after 'categories.json'.  received '%s'
the REST request has trailing characters after 'options.json'.  received '%s'
the REST request path is not understood.  received '%s'
the REST request is incomplete.  received '%s'
internal error: parseRESTrequest: line is NULL
incomplete REST request: no URL
cannot parse command line.  received '%s'
no path found.  received '%s'
malformed or missing path.  received '%s'
connection error: Content-Length was set to %d but could only read %d bytes of the body
expected a body chunk size but found empty line or EOF
cannot parse body chunk: unrecognised chunk size '%s'
already read %d bytes and next chunk of %d bytes is too large
could not read entire chunk: chunk size is %d but read only %d bytes
there was no '\r\n' after the last chunk
expected a non-zero Content-Length or Transfer-Encoding:chunked
error reading request line
error parsing header: %s
ufdbrestd does not support Upgrade header
connection error
```

12.2 Logfile Messages

The log file rotates automatically when it reaches the maximum size defined by the `max-logfile-size` parameter. When the logfile rotates, it is renamed to `ufdbrestd.log.1` ... `ufdbrestd.log.8` and a new file `ufdbrestd.log` is created.

The logfile contains informational messages, errors and fatal errors.

With the exception of a few informational messages all lines in the logfile are prefixed with a date-time stamp followed by the message. An error message starts with the text "ERROR:" and a fatal error messages starts with the text "FATAL ERROR:". The message may contain multiple lines. In case of a multi-lines message, the second and following lines have an indentation.

Examples:

```
ufdbrestd version 1.0.16
Copyright (C) 2014-2024 by URLfilterDB B.V. with all rights reserved.
2021-10-28 12:45:08.3022 FATAL ERROR: line 37: category has empty ID
2021-10-28 12:45:08.3028 ERROR: URL has no domain: /index.html
2021-10-28 12:45:08.3741 maximum number of connections for REST client is 25
2021-10-28 12:45:08.3741 number of REST server worker threads is 4
2021-10-28 12:45:08.3741 ufdbrestd started -- pid 14116, 4 threads, 25 connections
2021-10-28 15:45:09.8113 HUPfastRefresh: Could not obtain a write-lock quickly to refresh the URL database
2021-10-28 15:45:09.8113    sleeping 1 second before attempting to acquire the write-lock again
```

The following sections contain the complete list of errors and fatal errors. The list of informational messages is quite large since when the debug parameter is set, many different informational messages are logged. It is out of scope of this document to list informational messages.

12.3 Logfile Errors

The following messages are non-fatal errors without the date-time prefix. The first 3 messages are multi-line messages. Depending on the `use-syslog` parameter, these messages may also appear in the system log.

```
connection queue is full.
   REST client(s) are not obeying the Max-Connections option !!  *****

readBody: fd %d: there was no '\r\n' after the last chunk (protocol error)  Client %s
User-Agent: %s

RestAnswerGetURLlist: error parsing body of REST request
   Client %s  User-Agent: %s"
   body:
   %s

readSocketLine: fd %d: \n seen where \r\n was expected. buffer: %s
readSocketLine: fd %d: did not find a line terminator '\r\n' !!
parseRESTpath: the REST request has no path.  received '%s'  Client %s  User-Agent: %s
parseRESTpath: the REST request has an illegal value or has no terminating slash for
'useparams'.  received '%s'
parseRESTpath: the REST request has no URL after 'url/'.  received '%s'  Client %s
User-Agent: %s
parseRESTpath: the REST request has a URL without a domain.  received '%s'  Client %s
User-Agent: %s
parseRESTpath: the REST request must have '/useparams/no/ with " 'geturllist.json'.
received '%s'
parseRESTpath: the REST request must have '/useparams/yes/' with " 'categories.json'.
received '%s'
parseRESTpath: the REST request has no '?' after 'categories.json'.  received '%s'
Client %s  User-Agent: %s
parseRESTpath: the REST request has no '?url=' after 'categories.json'.  received '%s'
Client %s  User-Agent: %s
parseRESTpath: the REST request has trailing characters after 'options.json'.  received
'%s'  Client %s  User-Agent: %s
parseRESTpath: the REST request path is not understood.  received '%s'  " Client %s
User-Agent: %s
parseRESTpath: the REST request is incomplete.  received '%s'  Client %s  User-Agent:
%s"
```

```
parseRESTrequest: line is NULL  Client %s  User-Agent: %s
parseRESTrequest: no GET/POST/OPTIONS method found in '%s'  Client %s  User-Agent: %s
parseRESTrequest: cannot parse command line.  received '%s'  Client %s  User-Agent: %s
parseRESTrequest: cannot parse command line.  received '%s'  Client %s  User-Agent: %s
parseRESTrequest: no path found.  received '%s'  Client %s  User-Agent: %s
parseRESTrequest: parsing of path failed.  received '%s'  Client %s  User-Agent: %s
parseRESTheader: content is NULL  Client %s  User-Agent: %s
parseChunkSize: chunk_spec = NULL  Client %s  User-Agent: %s
parseChunkSize: chunk_spec is void  Client %s  User-Agent: %s
parseChunkSize: expected a chunk size but got '%s'  Client %s  User-Agent: %s
parseChunkSize: cannot convert chunk size '%s'" Client %s  User-Agent: %s
readBody: fd %d: Content-Length exceeds maximum of %d KB  Client %s  User-Agent: %s
readBody: Content-Length was set to %d but could only read %d bytes of the body.
Client %s  User-Agent: %s
readBody: fd %d: expected a body chunk size but found empty line or EOF   Client %s
User-Agent: %s
readBody: fd %d: cannot parse body chunk: unrecognised chunk size '%s'  Client %s
User-Agent: %s
readBody: fd %d: already read %d bytes and next chunk of %d bytes is too large.  Client
%s  User-Agent: %s
readBody: fd %d: could not read entire chunk: chunk size is %d but read only %d bytes.
Client %s  User-Agent: %s
readBody: fd %d: there was no '\r\n' after the last chunk (protocol error)  Client %s
User-Agent: %s
readBody: fd %d: cannot parse body chunk: unrecognised chunk size '%s'  Client %s
User-Agent: %s
readBody: expected a non-zero Content-Length or Transfer-Encoding:chunked.  Client %s
User-Agent: %s
restdRequest: fd %d: error reading request line
restdRequest: fd %d: error parsing request line "%s": %s
restdRequest: fd %d: error parsing header: %s
restdRequest: fd %d: REST header "Host" was not sent
restdRequest: fd %d: could not read the body content.  Client %s  User-Agent: %s
writeSocketBuffer: fd %d: eof=%d  write error: %s
fd %d: RestAnswerError since daemon has fatal error
fd %d: connection error (%d) detected in \"%s\": %s  Client %s  User-Agent: %s
RestAnswerOptions: pthread_rwlock_rdlock failed with code %d
RestAnswerGet: pthread_rwlock_rdlock failed with code %d
RestAnswerGet: fd %d: error releasing read-lock: %d
RestAnswerGetURLlist: pthread_rwlock_rdlock failed with code %d
RestAnswerGetURLlist: the list has more than %d members  Client %s  User-Agent: %s
RestAnswerGetURLlist: double quote to terminate URL expected  Client %s  User-Agent: %s
RestAnswerGetURLlist: fd %d: error releasing read-lock: %d
RestAnswerGetURLlist: fd %d: error releasing read-lock: %d
waitForInput: select failed for fd %d: %s
HUPfastRefresh: pthread_rwlock_timedwrlock failed with code %d
```

```
HUPfastRefresh: Could not acquire write-lock. The pthread implementation is poor for
acquiring write-locks *****

HUPfastRefresh: pthread_rwlock_wrlock failed with code %d

HUPslowRefresh: pthread_rwlock_timedwrlock failed with code %d

HUPslowRefresh: Could not acquire rw-lock. The pthread implementation is poor for
acquiring write-locks *****

HUPslowRefresh: pthread_rwlock_wrlock failed with code %d

sigterm: failed to kill the main thread

SH: sigwait() failed with %s

signal %d received unexpectedly

No new URL database was loaded in the past %d hours.  It is highly recommended to
refresh the URL database.  *****

readMoreDynamic: fd %d: run out of options *****

readMoreDynamic: REST request exceeds %d KB

request_main: poll returns EBADF !!!

closing new connection of REST client (fd %d) since internal fd queue is full

accept(2) on socket failed: %s

accept(2) failed: %s.  A new client failed to connect to this daemon. retrying...

the daemon socket is now re-opened after accept(2) failure.  (fatal error is now
resolved)

readSocketLine: fd %d: no input within time limit, eof=%d error=%d

readSocketLine: fd %d: no sufficient input within time limit  eof %d  error %d  *****

readSocketLine: fd %d: not enough input for continued line within time limit

readSocketLine: fd %d: \n seen where \r\n was expected. buffer: %s

readSocketLine: fd %d: did not find a line terminator '\r\n' !!

ufdbReadChunk: fd %d: illegal length of %d

ufdbReadChunk: fd %d: length too large: %d

ufdbReadChunk: fd %d: already have %d bytes and more %d bytes exceeds the buffer size

ufdbReadChunk: fd %d: eof=%d  could only read %d bytes of a chunk of %d bytes ***

ufdbReadChunk: fd %d: eof=%d  could not read the entire chunk of %d bytes: only
received %d bytes in %d seconds

ufdbReadContent: fd %d: illegal length of %d

ufdbReadContent: fd %d: length too large: %d

ufdbReadContent: fd %d: already have %d bytes and more %d bytes exceeds the buffer size

ufdbReadContent: fd %d: eof=%d  could only read %d bytes of a content of %d bytes ***

file \"%s\": error in regular expression \"%s\"

parseIPv6URL: illegal IPv6 address in URL: %s

URL '%s' has two or more '?'  (truncating second)

URL has no domain: %s

URL exceeds maximum length: %s

ufdbDecompressBZIP2Table: madvise( %p %d hugepage ) returns error %d: %s

ufdbDecompressZLIBTable: madvise( %p %d hugepage ) returns error %d: %s

URL category %s has a file cksum of %d but a different memory cksum of %d

table %s uncompression failed with error %d

UFDBloadDatabase: madvise( %p %d hugepage ) returns error %d: %s
```

```
UFDBupdateURLwithNormalisedDomain: URL does not start with '[': %s
UFDBupdateURLwithNormalisedDomain: URL does not have a ']': %s
category "%s" is empty
standard database table '%s' is old.  run ufdbRESTupdate -v
cannot load standard database table '%s' (API error code is %d)  *****
ufdbLoadDatabases: category list is empty
database table %s is old.  run ufdbRESTupdate -v
cannot load database table %s (API error code is %d)  *****
cannot load expressionlist %s (API error code is %d)  *****
cannot connect to %s/%s  port %d: %s
ufdbRegisterUnknownURL: cannot store very long domainname "%s"
ufdbrestd: can't write to logfile %s
restdCatchBadSignal: calling exit() after waiting for 10 seconds for main to terminate
ufdbrestd
Cannot create directory %s: %s
cannot change directory permissions (chmod) of %s: %s
cannot remove PID file %s
cannot write to PID file %s
main: bad signal %d during startup.  going to exit...
```

12.4 Fatal Errors in the Logfile

The following messages are fatal errors without the date-time prefix. The first 3 messages are multi-line messages. Depending on the `use-syslog` parameter, these messages may also appear in the system log.

```
UFDB file for table "%s" has an invalid key
   contact support@urlfilterdb.com

table %s is dated %s and has EXPIRED.  *******************
   Check the clock, licenses and cron job for the update command.
   Run the update command with -v option to verify the URL database download.

Could not load the URL database.  *****
   verify the value for "dbhome" (%s)  *****
   You may need to run ufdbRESTupdate [-v]

saveUncatURLsToFile: cannot create directory to store upload file: %s: %s
saveUncatURLsToFile: cannot stat upload-file-directory %s: %s
saveUncatURLsToFile: cannot create upload file %s: %s
daemon cannot create listening socket.  socket(AF_INET,SOCK_STREAM,0) failed: %s
cannot bind IPv4 socket (any address; port %d).  bind(2) failed: %s
tried %d times to bind the server socket.  Giving up and terminating...
IPv4 socket successfully bound after %d %s  (error is resolved)  *)
cannot listen on daemon socket.  listen(sockfd,%d) failed: %s
num-workers is %d.  Change num-workers or increase listen(2) backlog capacity or
consult an infrastructure expert
A FATAL ERROR OCCURRED: ALL REQUESTS WILL BE ANSWERED WITH "OK"
internal error: signal %d received.  ufdbrestd will terminate...
request_main: poll returns error: %s
accept(2) failed with EPERM.  Probably because of a firewall issue.
accept(2) and retries failed.  Giving up.
```

```
ufdbMalloc: failed to allocate %d bytes
ufdbMallocAligned: cannot allocate %d bytes %d-aligned memory
ufdbCalloc: failed to allocate %d bytes
ufdbRealloc: failed to allocate %d bytes
ufdbDecompressBZIP2Table: cannot allocate %d bytes memory for BZIP2 table decompression
ufdbDecompressBZIP2Table: BZIP2 decompression failed with code %d
ufdbDecompressZLIBTable: cannot allocate %d bytes memory for ZLIB table decompression
ufdbDecompressZLIBTable: ZLIB decompression initialisation failed: error %d  %s
ufdbDecompressZLIBTable: ZLIB decompression failed: error %d  %s
URL table %s has an invalid UFDB header
contact support@urlfilterdb.com
URL table %s has an erroneous UFDB header: indexsize < 0
UFDB file for table "%s" has data format version "%s" while this program does not
support versions higher than "%s"
Download/install a new version of this program.
URL table %s has an invalid UFDB2+ header
URL table %s has an invalid date (date=%s)
table %s has an invalid date (%s) or the clock is not correct
the difference between current system time and database timestamp is %d seconds (%d
days)
cannot parse 2.1 table: tag type is %d
cannot parse 2.2 table: tag type is %d
cannot parse database table with DB format %s
UFDBlookup admin=NULL
line %d: category has empty ID
line %d: illegal category identifier '%s'.  Only use alphanumeric and _-+/ in category
identifiers.
error unloading database table (error=%d)
error unloading expressions (error=%d)
error unloading standard database table (error=%d)
ufdbrestd caught signal %d
ufdbrestd: can't open configuration file %s
expected a [quoted] identifier in configuration file %s line %d
syntax error in category definition in configuration file %s line %d
```

*) This message indicates that the the previous fatal error "cannot bind IPv4 socket..." is now resolved.

12.5 Fatal Errors in the Download Logfile

The `ufdbRESTupdate` script calls the `ufdbRESTdl` program to download a URL database. `ufdbRESTdl` produces the logfile `ufdbRESTdl.log` which may contain the following fatal error messages. The following messages are fatal errors without the date-time prefix.

```
password has less than 6 characters
cannot open password file <file>: <error>
<file> has no password
fwrite(%d,%d) returned %d - error writing output file: errno %d %s
cannot write to database file <file>: %s
the database file could not (entirely) be saved on disk due to %d write error(s)
timeout downloading the database: %s
libcurl could not download the database: %d %s
URL database download failed with HTTP status code 401 Unauthorized. Doublecheck the
username and password settings.
URL database download failed with HTTP status code %d
failed to flush output file buffers - errno %d %s
URL database file <file> has only %d bytes and is assumed to be corrupt
fsync for URL database file <file> (%d bytes) failed: %s
```

```
cannot write to license status file <file>
cannot download the license status file: %s
license status download failed with HTTP status code %d
license status file <file> has only %d byte(s) and is assumed to be corrupt
```

13 Privacy Policy

The privacy policy of URLfilterDB is stated on the website: www.urlfilterdb.com/privacystatement.html.

14 More Information

The support desk can answer all questions. Send an email to support@urlfilterdb.com to ask a question or send your feedback.