Loway

QueueMetrics

call center suite

The QueueMetrics Uniloader User Manual

Loway SA

Version 26.10, 2026/10/12: covers Uniloader 26.10 for QueueMetrics 26.10

Table of Contents

What is Uniloader? . . . 1
- What happens if.... . . . 1

Installation . . . 3
- Standard installation (all systems) . . . 3
- Migrating from the yum (RPM) installation . . . 5
- Switching from the MySQL connection to the webqloader user . . . 7
- Manual installation . . . 8
- Running in production . . . 9
- Example: Uploading data to a local QueueMetrics system . . . 10
- Example: Uploading data to a QueueMetrics Live system . . . 10

Concepts . . . 12
- Back-ends . . . 12

Usage . . . 14
- Uploading data . . . 14
- Feedback actions: proxying AMI . . . 15
- Forced upload of existing queue_log files . . . 15
- Rewriting queues and agents . . . 16
- Splitting a single queue_log file into multiple back-ends . . . 17

Scenarios . . . 20
- One Asterisk instance, one local QueueMetrics instance . . . 20
- One Asterisk instance, one hosted QueueMetrics Live instance . . . 20
- One Asterisk instance, multiple QueueMetrics instances . . . 20
- Multiple Asterisk instances, one hosted QueueMetrics-Live instance . . . 21
- Multiple Asterisk instances, one QueueMetrics instance . . . 22
- One Asterisk instance, multiple QueueMetrics-Live instances . . . 22
- Multiple Asterisk instances, multiple QueueMetrics instances . . . 24

Event tracking . . . 25
- Installing event tracking through RPM . . . 26
- Tracking Music-on-Hold . . . 27
- Tracking Parking lots as queues . . . 27
- Automatically Tracking Outbound Calls . . . 29
- Automatically tracking hotdesking events . . . 32
- Debugging missed events and incorrect logging . . . 32

FreeSwitch / FusionPBX support . . . 34
- General overview . . . 34
- Setting up . . . 40

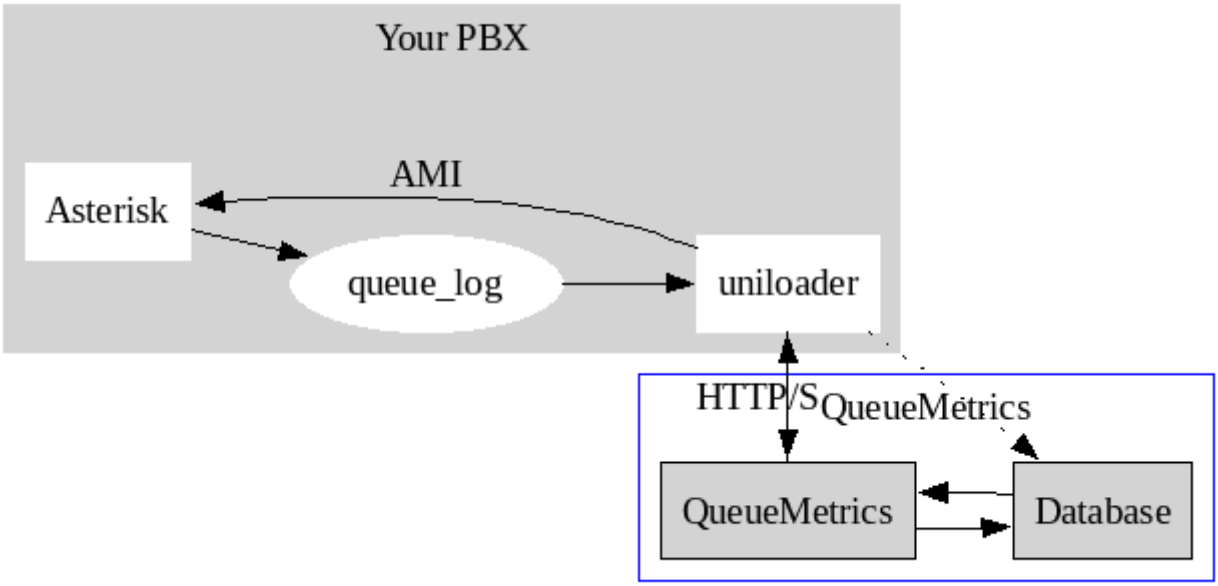
Diagnostics and tools . . . 44
- Diagnostics: AMI connection test . . . 44

Diagnostics: Test upload link . . . 46
Diagnostics: Test connection to Mysql/MariaDB database . . . 47
Diagnostics: Test connection to Postgres database . . . 48
Diagnostics: Open a SQLite database . . . 49
Diagnostics: Test Freeswitch's ESL port . . . 49
Diagnostics: Test data download from Enswitch . . . 51
Diagnostics: Test an AudioVault server . . . 52
Diagnostics: Direct QueueMetrics database access . . . 53
Direct DB access: Database contents . . . 53
Direct DB access: Exporting a partition . . . 54
Direct DB access: Deduplicating data . . . 54
Direct DB access: Creating private clones of reports . . . 56
Diagnostics: Regexp tester . . . 57
Diagnostics: Transcriptions . . . 58
Diagnostics: WebSocket . . . 59
Diagnostics: 3CX . . . 60
PBX Information . . . 65
FreePBX . . . 65
Issabel . . . 66
FusionPBX . . . 66
Uploading configuration to a QueueMetrics instance . . . 67
Integrics Enswitch . . . 68
3CX . . . 69
Asterisk tools . . . 71
Queue-Add: add an extension to some or all queues . . . 71
Queue-Kick: log-off all (or some) users . . . 72
Queue-Replicate: replicate presence events . . . 74
User Information . . . 76
Editing users . . . 76
Searching for users . . . 77
Editing classes . . . 77
Searching for classes . . . 78
Expiring users . . . 78
Password reset . . . 78
AudioVault . . . 80
The FileSearch driver . . . 81
Serve: runs the AudioVault service . . . 81
Securing AudioVault with HTTPS . . . 84
find: Testing a configuration . . . 86
Configuration files . . . 87
QueueMetrics configuration.properties / *arch_properties* table . . . 87

Transcribe . . . 91
Overview . . . 91
Transcription Servers and Capacity . . . 92
Drivers . . . 92
Task Tags . . . 93
Running the Command . . . 93
How It Works . . . 96
Stereo Agent Mode . . . 97
HTTP Trace Log . . . 98
Sampling (`--skip-prc`) . . . 98

What is Uniloader?

Uniloader is a program that is installed on your Asterisk PBX. It uploads data to a local or remote QueueMetrics instance and receives actions to be performed on the local PBX.



Uniloader is deployed as a single binary file that has to be installed on the PBX itself. It is designed as a very lightweight application so it can work unobtrusively even on low-end hardware; and it is meant to be very safe, so data will not be lost even in cases where the remote QueueMetrics server becomes unavailable.

When it runs, it uploads data using either HTTP/S or the MySQL protocol (depending on the back-end you specify). HTTP/S is meant to run with remote QueueMetrics instances, especially QueueMetrics Live (see http://queuemetrics-live.com for more information) , while MySQL is meant for local systems; either case works if QueueMetrics is hosted on the same machine.

When running over HTTP, if the QueueMetrics server has no direct connection to the PBX, Uniloader is able to act as a proxy and will receive actions to be performed on the Asterisk server via AMI (Asterisk Manager Interface). This way you can run QueueMetrics remotely and still take advantage of the ability to log agents on and off, pause them, listen to calls via ChanSpy, etc.

Uniloader is also used to perform other administrative/complementary tasks that perform useful functions on an Asterisk system connected to QueueMetrics; for example, it can generate music-on-hold events on queues, and can help diagnosing issues.

What happens if....

- *The queue_log file is rotated*: Uniloader will detect the rotation and will pick up the new file
- *The remote system becomes unavailable*: Uniloader will keep on trying until the system is back on line. When it is, it will detect how much data was uploaded and will reload the missing parts. If using multiple remote systems one of them becomes unavailable, only data for that specific system will be blocked, while other systems will keep on working in real-time.

- *Asterisk becomes unavailable*: any commands meant to be sent to Asterisk are queued and will be performed when Asterisk comes back on-line
- *The PBX reboots*: you need to make sure Uniloader starts on boot. It has no device dependencies so it can be started at any time.

Installation

Uniloader is installed from a `.tar.gz` package, using the `install.sh` script it contains. This works on any system running Systemd, including CentOS-based and RPM-based PBX systems (FreePBX, PBXact, etc.), Debian and Ubuntu. If your system does not run Systemd, refer to the manual install below.

The Loway yum repository is no longer available, so `yum install uniloader` and `yum update` cannot be used anymore. If you installed Uniloader with yum in the past, see Migrating from the yum (RPM) installation.

Standard installation (all systems)

Installing Uniloader is very easy. First find the package you want to download under https://downloads.loway.ch/software/uniloader/

```
mkdir -p /opt/loway/uniloader
cd /opt/loway/uniloader
wget https://downloads.loway.ch/software/uniloader/uniloader-26.10.1.tar.gz
tar zxvf uniloader-26.10.1.tar.gz
cd uniloader-26.10.1

./install.sh
```

The automated install script will:

- install Uniloader system-wide (so you can call it from any shell)
- create configuration and init scripts for all services below
- start the service `uniloader` (though it won't work until you edit its credentials and restart it) and make it startable on reboots
- install a script called `./check.sh` that you can use to check the settings for each of the services, by using the configuration you specified for each unit. This makes it very easy to make sure it is correct.

For each service, we keep a separate configuration file under `/etc`, that is the one you are generally supposed to edit. If you make a change to the configuration files, make sure you issue a `systemctl restart uniloader` so that the new configuration can be picked up.

If you choose to enable any other service, it's up to you to make sure that it is correctly restarted on boot.

To update, just download the new version of Uniloader as you did, and run the install script again. When updating, configuration files are not changed, while unit files are replaced with the current version.

A quick recap of Systemd commands might help:

- `systemctl restart uniloader` will restart the service, so a new configuration can be picked up
- `systemctl status uniloader` will show the status of the service (if it's running or not), and the last lines of its logs.
- `journalctl -u uniloader` will show a complete log, that you can then filter appropriately. You can pass multiple `-u` services that you want to see, and you'll see the logs in the correct order.
- `systemctl enable uniloader` will make sure that this service will start on boot
- `systemctl daemon-reload` is needed if you make any change to the unit scripts. Usually not needed.
- `journalctl --disk-usage` shows the size of the logs

Uniloader

- Service name: `uniloader`
- Configuration file: `/etc/uniloader`
- Testing: `./check.sh upload` that will use the current configuration and make sure the destination QM server is reachable.

We strongly suggest always using the HTTP driver to connect to QM - this makes it easy to send data to a local instance, a remote instance you control, or a QueueMetrics Live instance. Remember to enable the user `webqloader` on QM, and set a password for it.

```
URI=http://127.0.0.1:8080/queuemetrics
LOGIN=webqloader
UPASSWD=Secret1234
TOKEN=
```

`PASS` or `UPASSWD`? The `uniloader` program always receives the password through the environment variable `UPASSWD` (or through the `--pass` option). What changes between installations is the name used **in the configuration file**, and it depends on how the service unit reads it:

- If the unit file contains the line `Environment=UPASSWD=${PASS}`, the password must be set as `PASS=` in the configuration file. This is the case for most older and manually-created units, and for the files of old RPM installations (`/etc/sysconfig/uniloader`).
- If the unit file does not contain that line, set `UPASSWD=` in the configuration file, as in the example above.

To see which one applies to you, run `systemctl cat uniloader` and look for `UPASSWD`. Use only one of the two names, and do not rename it when you edit the file: just change its value. If the service starts but the log shows an authentication error even if the password is right, the wrong variable name is the most likely cause.

Make sure you restart the service after making any changes.

You can also turn-on the splitter for Uniloader by uncommenting the right line, and the editing the example splitter file and `/etc/uniloader_splitter.json`

Unitracker

- Service name: `unitracker`
- Configuration file: `/etc/unitracker` - you must set up the correct AMI credentials
- Testing: `./check.sh unitracker` will make sure that the AMI credentials used by Unitracker are working
- This service must be enabled manually if needed.

By default, only MOH tracking is enabled. You may want to enable parkings or, more commonly, outbound tracking. In this case, do not forget to set up the "hidden channels" mask to decide which calls you don't want to track.

AudioVault

- Service name: `audiovault`
- Configuration file: `/etc/audiovault`
- Testing: `./check.sh audiovault` that will try requesting a (non-existent) file as if it were an external client. You are expected to see no errors there.
- This service must be enabled manually if needed
- In order to run this effectively in production, it's strongly advisable to put an HTTPS proxy in front, as per its documentation.

Freeswitch

- Service name: `uniloader-freeswitch`
- Configuration file: `/etc/uniloader-freeswitch`
- Testing: `./check.sh freeswitch` that will check ESL credentials
- This service must be enabled manually if needed
- When enabiling this service, you must make sure that the queue_log file that this service generates and the one being imported by Uniloader match - so do not forget to check `/etc/uniloader` as well.
- This service is meant to run all of the time and never being restarted, as it will lose events if it is restarted during its operation.

Migrating from the yum (RPM) installation

If you installed Uniloader with `yum install uniloader`, you can no longer update it with `yum`. To upgrade, remove the RPM package and install the `.tar.gz` package as described in Standard installation (all systems). From then on, every update is done by running `install.sh` again. You

need root access and a system running Systemd; plan a short maintenance window, as data is not uploaded while Uniloader is stopped.

The two installations place their files in different locations:

	yum (RPM) installation	**tar.gz with `install.sh`**
Updates	`yum update`	download the new package and run `install.sh` again
Program files	`/usr/local/uniloader`	installed system-wide by `install.sh`
Uniloader configuration	`/etc/sysconfig/uniloader`	`/etc/uniloader`
Unitracker configuration	`/etc/sysconfig/unitracker`	`/etc/unitracker`
Service control	`service uniloader restart`	`systemctl restart uniloader`

Follow these steps:

1. **Back up your configuration files.** Save every file you use, and write down the installed version with `uniloader --version`:

   ```
   tar czf /root/uniloader-config-backup-$(date +%F).tar.gz \
     $(ls -d /etc/sysconfig/uniloader* /etc/sysconfig/unitracker* \
             /etc/uniloader* /etc/unitracker* /etc/audiovault* \
             /etc/defaults/uniloader* 2>/dev/null)
   ```

   If you start Uniloader in other ways (for example from `/etc/rc.local` or a cron job), save those files too.

2. **Remove the RPM package.**

   ```
   service uniloader stop
   yum remove uniloader
   rpm -qa | grep -i uniloader
   ```

   The last command should print nothing. If you do not use the Loway repository for any other package, you can also delete `/etc/yum.repos.d/loway.repo`.

3. **Install the new package** with `install.sh`, as described in Standard installation (all systems).

4. **Restore your settings** in the new configuration files under `/etc`, using your backup as a reference. Copy the values one by one (`URI`, `LOGIN`, the password, `TOKEN`) instead of overwriting the new file with the old one: the new files contain comments that explain each setting. Pay attention to the name of the password variable: see the `PASS` or `UPASSWD` tip above. Settings that only served the old init script, such as the log, lock and pid file paths, are not needed anymore; logs are now available with `journalctl -u uniloader`.

5. **Start and check.** Restart the service and test the connection:

```
systemctl restart uniloader
./check.sh upload
```

Then place a test call and make sure that it appears in QueueMetrics.

Only the `uniloader` service is started by `install.sh`. If you used other services, such as the tracking service, set up the configuration file under `/etc`, test it with `./check.sh <service>` and enable it with `systemctl enable --now <service>`.

If you were using a direct MySQL connection (a `URI` starting with `mysql:`), you can keep it, copying `URI`, `LOGIN`, the password and `TOKEN` (the partition) exactly as they were. We suggest instead to switch to the HTTP driver with the `webqloader` user: see Switching from the MySQL connection to the webqloader user.

Switching from the MySQL connection to the webqloader user

Uniloader can upload data to QueueMetrics in two ways:

- **MySQL (the old way):** Uniloader writes directly to the QueueMetrics database (`URI` starting with `mysql:`). This is the method used by older installations, and it is still supported.
- **HTTP/S (the new way):** Uniloader sends data to QueueMetrics over HTTP/S, using the `webqloader` user of QueueMetrics (`URI` starting with `http:` or `https:`). This is now the recommended method.

The HTTP/S connection is better because:

- the PBX does not need to reach the MySQL server, and does not need to know the database credentials;
- it works in the same way with a local instance, a remote instance or QueueMetrics Live;
- when running over HTTP, Uniloader can act as a proxy for the actions QueueMetrics sends to the PBX (log agents on and off, pause them, listen to calls, etc.);
- data deduplication is available (it requires QueueMetrics 19 or later and the HTTP interface).

To switch:

1. **Enable the `webqloader` user in QueueMetrics.** On QueueMetrics Live it is already enabled; on an on-premise installation, go to **Users**, find `webqloader`, tick **Enabled**, set a strong password and save. The user must hold the keys `WQLOADER`, `USER` and `ROBOT`.
2. **Edit `/etc/uniloader`.** Comment out the MySQL settings and set the HTTP ones:

```
#URI="mysql:tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1"
#LOGIN=queuemetrics
URI=http://127.0.0.1:8080/queuemetrics
LOGIN=webqloader
```

```
UPASSWD=Secret1234
TOKEN=
```

The `URI` is the base address of QueueMetrics, as you would use it in a browser, with `https://` if QueueMetrics is behind SSL. For the name of the password variable, see the `PASS` or `UPASSWD` tip above.

3. **Mind the token.** In MySQL mode the token holds the **partition** (such as `P001`); in HTTP/S mode it is usually left blank. In a cluster, it holds the name of the cluster member instead. If you were using a partition, check your cluster configuration before switching.
4. **Restart and check.**

```
systemctl restart uniloader
./check.sh upload
```

Then place a test call and make sure it appears in QueueMetrics. If the login fails, check that `webqloader` is enabled and that the password variable has the right name.

Once everything works, the MySQL account that Uniloader used is no longer needed: if it was dedicated to the PBX, you can remove it and close access to the MySQL port from the PBX.

Manual installation

The Uniloader can be downloaded from https://downloads.loway.ch/software/uniloader/

The package contains:

- Uniloader binaries for all supported architectures (i386, amd64, arm7, arm64),
- A sample extensions_queuemetrics file,
- A sample splitter file.

Just copy the file "uniloader_xxx" for your architecture (Intel 32 / 64 bit, ARM 7, ARM64) into your computer and make it executable:

```
cp ./bin/uniloader_arm7 ./uniloader
chmod a+x ./uniloader
```

To test it, run:

```
./uniloader -?
```

It should output a result like:

```
NAME:
```

```
   uniloader - the data upload companion for QueueMetrics and QueueMetrics Live.

USAGE:
   uniloader [global options] command [command options] [arguments...]

VERSION:
   22.11.3 - build: 1234-20230511.0849/8300761 - OS: darwin/arm64 - RT: go1.17.5

COMMANDS:
   help, h  Shows a list of commands or help for one command

   Diagnostics:
     pbxinfo  Loads PBX information
     test     Runs environment tests

   Services:
     upload, u  Uploads a source file to a QueueMetrics or QueueMetrics Live instance
     track, t   Tracks Asterisk events and creates relevant queue_log entries.
     fsw        Parses FreeSwitch mod_callcenter events
     av         Runs AudioVault - a media storage server for QM (experimental)

   Tools:
     cfgfile   Reads and writes a configuration file
     user      Creates or alters users
     qmdb      Performs direct operations on QueueMetrics' database
     asterisk  Performs direct operations on Asterisk queues

GLOBAL OPTIONS:
   --src value, -s value  The source queue_log file to be uploaded (default:
"/var/log/asterisk/queue_log")
   --cacert value         An optional CA Cert file in .pem format, or 'insecure' to
skip any certificate validation
   --verbose-back-end     Enables verbose back-end logging. Default: false.
   --read-pipe            The source file is a Unix pipe
   --help, -h             show help
   --version, -v          print the version
```

If it does, it is working.

Running in production

Uniloader produces a verbose log on STDOUT that should be redirected to a file and periodically rotated.

You should also make sure that Uniloader is started when the PBX boots and that in case it should crash it is automatically restarted.

We advise running Uniloader using *nice* so that it has reduced access to scarce CPU resources in case of high load / contention with the PBX - while the PBX voice quality quickly degrades on a resource-starved system, Uniloader does not really care about small delays in data uploading.

We also advise adding Uniloader to the command path, so that its many debugging functions can be accessed easily.

Example: Uploading data to a local QueueMetrics system

This is the most common scenario when using a locally installed QueueMetrics system.

```
nohup nice \
./uniloader -s /var/log/asterisk/queue_log \
    upload --uri "mysql:tcp(1.2.3.4:3306)/queuemetrics?allowOldPasswords=1" \
            --login queuemetrics --pass javadude --token P001 \
            >> /var/log/uniloader.log &
```

Will upload the queue_log file located at `/var/log/asterisk/queue_log` to the remote "queuemetrics" database on server 1.2.3.4 with login "queuemetrics" and password "javadude", using the default partition "P001".

After you start it, check the file `/var/log/uniloader.log` to make sure there are no errors. The most common error is that you did not create the correct grants for your MySQL user to upload data remotely.

If everything seems to work, log in into QueueMetrics, select "Edit system parameters" and make sure that the default partition is P001.

```
# This is the default queue log file.
default.queue_log_file=sql:P001
```

At this point, log off; log on again and click on "Mysql storage information"; select partition P001 and select "Autoconfigure queues". Now the default queue "00 All" will include all of your queues and you can see your historical and real-time status.

See also One Asterisk instance, one local QueueMetrics instance.

Example: Uploading data to a QueueMetrics Live system

You received your QueueMetrics Live access information; and in the received email you find that your instance is called "ABCD" and the password is "1234".

```
nohup nice \
./uniloader --src=/var/log/asterisk/queue_log \
     upload --uri https://my.queuemetrics-live.com/ABCD \
            --login webqloader --pass 1234 \
```

```
            >> /var/log/uniloader.log &
```

At this point Uniloader will start feeding the remote database. You can login at any time by visting the address https://my.queuemetrics-live.com/ABCD and logging in as "demoadmin" and the password you were given.

The first time you log in, click on "System diagnostic tools" and then "Live DB inspector" to see data being uploaded. When your database is complete (this may take a few minutes, depending on how much data is on your PBX), go back to the home page and click on "Mysql storage information"; select partition P001 and select "Autoconfigure queues". Now the default queue "00 All" will include all of your queues and you can see your historical and real-time status.

See also One Asterisk instance, one hosted QueueMetrics Live instance. If you run a Yeastar system, see the chapter dedicated to Yeastar MyPBX.

Concepts

Back-ends

Uniloader supports three different back-ends: HTTP/HTTPS, MYSQL and FILE.

Each back end is functionally similar and can be thought of as a black box; it can be selected simply by entering a proper URI for the server.

HTTP/HTTPS back-end

If your URI looks like:

```
http://myserver/queuemetrics
```

Then you are using HTTP. In this case, the value of the "token" parameter is either a server in a cluster, or you can leave it blank to denote the default server, and user/password are for a valid QueueMetrics HTTP user.

The HTTP back-end also supports HTTPS URLs and will, by default, retrieve actions to be performed on the PBX.

Please note that some appliances do not support HTTPS, so running HTTP might be mandatory.

HTTPS CA certificate issues

On some systems (especially appliances) it is possible that when running HTTPS requests, they all fail with an error like:

```
x509: failed to load system roots and no roots provided
```

In this case, you have to manually tell the Uniloader where to find the correct CA .pem files for your system, by using the "--cacert" parameter.

E.g.

```
./uniloader --cacert=/etc/certs/default.pem upload ....
```

Will force Uniloader to use the supplied root certificates. In case they are totally missing, we suggest copying a recent certificate file from a working Linux distribution and point to that.

As a temporary workaround, you can completely disable certificate validation by issuing `--cacert insecure`. When running in this mode, a log line is on command start-up to remind you that you are not verifying certificates.

MySQL back-end

If your URI looks like:

```
mysql:127.0.0.1/queuemetrics
```

the loader will connect to a MySQL database called "queuemetrics" on "127.0.0.1", using the supplied login and password; the token in this case is the partition that we want to upload data to.

If your MySQL is running on a remote system, it might be advisable to use a MySQL URI of the format:

```
mysql:tcp(1.2.3.4:3306)/uniloader?allowOldPasswords=1
```

This will connect to a database called "uniloader" on 1.2.3.4 and will set the parameter "allowOldPasswords" to 1, as it is sometimes needed to use old versions of MySQL.

A complete reference of all allowed DSN (Data Source Name) formats and connection parameters is available at https://github.com/go-sql-driver/mysql

If upload is stuck at some point and cannot progress because of an error that looks like `Data too long for column 'agent' at row 1`, this means that some fields on your queue_log file are larger than the allowed space on the server's database. In our experience, this only happens if you have a corrupted file or you manually logged a value that is exceedingly long. At this point, you can either:

- Edit the queue_log file to remove the extraneous value and restart Uniloader, or
- Restart Uniloader with a connection string like `mysql:tcp(1.2.3.4:3306)/uniloader?allowOldPasswords=1&sql_mode=''` that will prevent the database from complaining by silently trimming any long strings. Please note that this may cause synchronization issues when trying to find the current high water mark and in any case causes data loss, so should be used only sparingly to override a temporary stumbling block.

File back-end

If your URI looks like:

```
file:/my/file/path
```

The loader will try and append to a local file. This module is meant for quick testing of splitting rules and does not currently check the state of the local file before writing to it.

It can also be used as a quick way to "throw away" a log file, by using *file:/dev/null* on Unix systems.

This back-end is only meant for testing and experimentation; the file is rebuilt on every run, so no guarantee about data integrity is implied.

Usage

Uploading data

To upload data, you need the upload command in Uniloader:

```
NAME:
   uniloader upload - Uploads a source file to a QueueMetrics or QueueMetrics Live
instance

USAGE:
   uniloader upload [command options] [arguments...]

OPTIONS:
   --uri value, -u value       The connection URI. Valid URIs start with file:,
mysql:, http:, https:
   --login value, -l value     The login for your connection (default: "webqloader")
   --pass value, -p value      The password for your connection (default: "qloader")
[$UPASSWD]
   --token value, -t value     In MySQL mode, the partition. In HTTP/S mode, usually
blank or server-id
   --splitter value, -x value  A JSON file describing how to split the source into
multiple QM instances
   --noActions                 Actions from QM will NOT be sent to the PBX via AMI.
Requires HTTP/S.
   --pid value                 The PID file to write. If present, won't start.
   --db-rewriter-json value    The JSON configuration file for database agent and
queue rewrites.
   --forced-upload             Will upload data without checking for HWM and will
terminate when file is over.
```

So you usually launch it like:

```
./uniloader --src /var/log/asterisk/queue_log upload \
            --uri mysql:/queuemetrics --login qm --pass 1234 --token P001
```

You can avoid passing parameters which value matches the defaults, so if your token is blank, or your user is "webqloader" (as it is the case with default QueueMetrics Live instances), you do not need to pass them explicitly.

Uniloader reads the source file specified in "src" and automatically detects if the file is rotated/rewritten.

When data is being uploaded, Uniloader makes sure that data is not uploaded twice and retries on errors. You can safely restart it at any time and it will automatically synchronize with the current state of the selected back-end.

You should NEVER have multiple Uniloader (or older Qloader) processes point to the same partition on the same instance at the same time. If you do, you will get hard-to-debug data corruption.

Feedback actions: proxying AMI

It is possible for Uniloader to act as a kind of proxy for a remote QueueMetrics instance. This happens by default if you use a HTTP back-end. If you do not want this feature, you need to start Uniloader with the "--noActions" option.

For example:

```
./uniloader --src /var/log/asterisk/queue_log upload \
            --uri http://my.queuemetrics-live.com/test1234 --pass 1234
```

All access information to the Asterisk PBX is to be configured on the QueueMetrics instance; for example, if the PBX server is accessible on the address 127.0.0.1 (so the same host Uniloader is running on) and you log-in as "admin" password "amp123", you should edit the configuration properties and make sure that it says:

```
callfile.dir=tcp:admin:amp123@127.0.0.1
default.webloaderpbx=true
platform.pbx=DIRECTAMI
```

If you use mode CLASSIC, make sure you include the default QueueMetrics dial-plan in extensions.conf: `#include extensions_queuemetrics.conf`, and have Asterisk reload the configuration.

The AMI feedback feature works transparently for both Asterisk AMI and FreeSwitch ESL (as long as the platform is set correctly).

You can see actions being performed in QueueMetrics from "System diagnostic tools" and then selecting "Remote commands".

Forced upload of existing queue_log files

It is possible to upload a queue_log file and then have Uniloader terminate as soon as the operation completes.

Just run it as:

```
uniloader -s /var/log/queue_log.12 upload \
          -u https://my.queuemetrics-live.com/test/ -p 9999 --forced-upload
```

In this mode:

- The file is uploaded from the beginning to the end, no matter the current HWM for the instance - you can upload older data, or multiple log files without caring about their sequence.
- You can run this in parallel to an existing Uniloader service that is uploading current data.
- You can upload the same file multiple times - if some or all data is already present on the database, it will be skipped.
- You can apply splitting and rewriting rules - in this case, the process terminates when the last uploader consumes all rows.

In order for data deduplication to be applied, you need a QueueMetrics version 19+ and you need to use the HTTP interface for uploading. On older systems or using direct MySQL access, data **will** be duplicated, so this should not be used. In the case you suspect duplicate data, see Deduplicating data.

To upload multiple files in a sequence, you could use something like:

```
for FILE in $( find /var/log/asterisk -type f -iname "queue_log*" -printf "%T+ %p\n" |
sort | awk '{print $2}' ) ; \
   do uniloader --src "${FILE}" upload --uri https://my.queuemetrics-live.com/INSTANCE
--token "" --pass "PASS" --forced-upload ; done
```

This gathers all files under `/var/log/asterisk` that look like partial queue_logs, sorts them and uploads them all.

Rewriting queues and agents

If you run Uniloader with the option `--db-rewriter-json` and pass a JSON file like the one below:

```
{
    "type": "mysql",
    "uri": "localhost/queuemetrics",
    "login": "queuemetrics",
    "password": "javadude",
    "shorten-domain": false,
    "sql-agent": "SELECT '' as TENANT, ? as ID ",
    "sql-queue": "SELECT '' as TENANT, ? as ID  "
}
```

Then every time an agent or queue id is found, a SQL query is run to resolve it to a tuple *(tenant, id)* that in turn is used to create its actual name.

This is useful because sometimes you use simple ids for queues and agents, but such IDs look bad and are not useful for splitting the log into multiple tenants. E.g. if your agent on the queue is called *SIP/10907686*, it would be better to use it as *SIP/customer7-123* if you know that id *10907686* is agent *123* for tenant *customer7*.

Rewriting happens after the log is read and before it is split, so the splitter already receives agent

and queue fields rewritten.

- The queries must return exactly one line, with two string fields that are the tenant and the agent id.
- The placeholders are replaced in the query.
- If you do not use multiple tenants, always return a blank string as the tenant.
- It is better to return a complete agent id, like *Agent/123*, rather than just *123*
- As the tenant name is often the virtual host that the tenant uses on your system, you can have it shortened to the first token, e.g. "customer3.mypbx.some" becomes "customer3".

To avoid excessive database load, queries are run just once. It is mandatory that the same query always returns the same result, or multiple runs might produce different queue_log files.

Splitting a single queue_log file into multiple back-ends

If you run a single Asterisk instance on which multiple clients are hosted, chances are that you configure your Asterisk system with a common naming convention, so that all extensions for your client Foo Company are named "foo-123", all queues are named "foo-q1" and so on.

If you do, it is actually possible to split the queue_log file that Asterisk generates into multiple virtual queue_log files. To do this, Uniloader looks for references of the client name in queues and agents, and can optionally rewrite them so that a reference for queue "foo-q1" is sent to a specific QueueMetrics Live instance set up just for Foo Company; and it is rewritten as simply "q1".

To split a single queue_log file you need to create a split file that details what you want done, and then you can launch:

```
./uniloader --src queue_log.txt upload --splitter splitter.json
```

Please note that you do not need to specify a "main" rule on the command line. If you do, a copy of the source file will be also uploaded to the main driver, without applying any transformation.

These are sample contents for a splitter.json file:

```
[
    {
        "uri": "http://my.queuemetrics-live.com/foocompany",
        "login": "webqloader",
        "pass": "verysecure",
        "token": "",
        "matcher": ["foo-"],
        "match": "any",
        "removematch": true,
```

```
        "disabled": false,
        "noactions": false,
        "clientname": "foo"
    },
    {
        "uri": "mysql:127.0.0.1/queuemetrics",
        "login": "queuemetrics",
        "pass": "itsasecret",
        "token": "P001",
        "matcher": ["bar-"],
        "match": "any",
        "removematch": false
    }
]
```

The following items must be specified for each instance.

- `uri`: the URI to upload data to. You can mix and match different backends as you see fit
- `login`, `pass` and `token`: the information required by your back-end
- `matcher`: an array of strings that will be searched in the agent and queue fields.
- `match`: it can be either "any" (if a string is found, it is considered a match), "prefix" or "suffix"
- `removematch`: if true, the matching string is removed from the queue and agent fields
- `disabled`: set to true to manually turn off a rule
- `noactions`: set to true to turn off AMI actions for this instance, as you would do for the main instance by using the "--noActions" flag.
- `clientname`: the name of the instance, that will be injected in the AMI responses using the dialplan variable `UNILOADER_CLIENT` before they are passed to Asterisk. It will also used to replace the sequence `!UNILOADER_CLIENT` in your Asterisk channels.

If you avoid setting some item, it is assumed to be a blank string or the "false" boolean value. Defaults you set with the command lines are ignored, so all relevant information must be specified in the JSON file.

Split data is sent only to instances matching the specific split rule; so the main instance you specify on the command line will be fed all data in any case. As you usually do not want this, you can simply avoid entering any "--uri" parameter on the command line.

Splitting FAQs

What happens if one back-end is or becomes unavailable?

Each back end runs in parallel; but if one should lag behind or should not be available, data for it is delayed until the system is fully operational; at that point it will catch up automatically.

You can also safely restart Uniloader even if not all data is currently uploaded to all instances; the

only thing you have to consider is that, in case your queue_log is rotated, then only data present in the current queue_log file is uploaded.

This works correctly only for the MySQL and HTTP drivers; in case you specify a file back end, it will be truncated and rebuilt on each invocation.

Can I use different back-ends?

Yes, of course. Mix and match them as you best see fit.

Can I use feedback actions?

Yes - provided that all back-ends are HTTP/S.

What happens to the default back-end?

The default back-end - the one that is specified on the command line - is sent the raw *queue_log* data. If you don't need this, you can use a file back-end and point it to */dev/null*, or you can simply omit it.

Do I have to have a splitting rule for all my virtual clients?

No. Only the rules you specify will be applied, so if you do not include a rule for a specific client, the relevant logs will simply be ignored. This means that you may host on the same Asterisk instance clients who use QueueMetrics and clients that don't.

How do I modify the configuration on a live system?

You can simply create a new JSON file and restart the Uniloader. In a few seconds it will sync again and start tailing the files. The file will be read in parallel by all the different back-ends, so it will not require a proportional amount of disk IOPS.

Why do I need the clientname field?

If you have a scenario where multiple QM instances are fed by the main QueueMetrics instance, it will be handy to have rewriting enabled, so that e.g. the queue called "foo-q1" appears at the QueueMetrics level as simply "q1".

This works fine when uploading data to QueueMetrics, but when actions are performed by that QueueMetrics instance, they will appear as happening on queue "q1" and not on the actual Asterisk queue "foo-q1".

By injecting the variable UNILOADER_CLIENT is therefore possible to edit the actions dialplan and rebuild the correct physical name to be used when performing actions at the Asterisk level.

Scenarios

One Asterisk instance, one local QueueMetrics instance

You want to use Uniloader for your local QueueMetrics instance.

In this case you should use the MySQL back-end. You would not usually use the AMI feedback as QueueMetrics is able to connect directly to the PBX.

You may also use the HTTP back-end, but there are currently no advantages in doing so.

See also: Example: Uploading data to a local QueueMetrics system.

One Asterisk instance, one hosted QueueMetrics Live instance

You want to use Uniloader for a QueueMetrics-Live instance.

In this case you should use the HTTP or HTTPS backend, and turn on AMI feedback, as the QueueMetrics Live instance has no way to connect directly to your PBX.

In order to make your life easier, QueueMetrics Live actions are pre-configured to send actions back via HTTP; you just need to make sure that the AMI credentials specified in the property *callfile.dir* in *configuration.properties* match the ones used on your PBX.

See also: Example: Uploading data to a QueueMetrics Live system.

One Asterisk instance, multiple QueueMetrics instances

You may want to send multiple copies of the same data set to different QueueMetrics systems - e.g. in a very busy contact center, you might have one server used for agent logins and interactive monitoring, and a separate one to run large reports and receive API requests. This way you would be able to share the load and avoid bringing down interactive users if a huge report causes an "out of memory" error.

Another scenario could be disaster recovery, where you send a second copy of all call data to a different location.

This can be achieved by using the splitter, entering an empty matcher so that all servers match all rows. For example, your *'splitter.json'* file could look like:

```
[
    {
        "uri": "http://reporting-server/queuemetrics",
```

```
        "login": "webqloader",
        "pass": "CHANGEME",
        "token": "",
        "matcher": [""],
        "match": "any",
        "removematch": false,
        "disabled": false,
        "noactions": true,
        "clientname": ""
    },
    {
        "uri": "http://agent-server/queuemetrics",
        "login": "webqloader",
        "pass": "CHANGEME",
        "token": "",
        "matcher": [""],
        "match": "any",
        "removematch": false,
        "disabled": false,
        "noactions": false,
        "clientname": ""
    }
]
```

Though not strictly necessary, you may want to enable actions only for the server that is used interactively and disable them for the server running batch reports (by setting `noactions` to true).

See also: Splitting a single queue_log file into multiple back-ends.

Multiple Asterisk instances, one hosted QueueMetrics-Live instance

You have multiple Asterisk boxes and want to consolidate all their activity into a single QueueMetrics-Live instance.

QueueMetrics-Live supports up to 5 Asterisk instances in a cluster for a total of 50 agents per instance. Each Uniloader instance must upload data to a different partition. Each cluster member should be defined in the QueueMetrics-Live instance.

For example, this is how you would configure three Asterisk instances on a QueueMetrics-Live system:

```
default.queue_log_file=cluster:*

cluster.servers=srva|srvb|srvc

cluster.srva.manager=tcp:dial:12345@127.0.0.1
cluster.srva.queuelog=sql:A
```

```
cluster.srvb.manager=tcp:dial:12345@127.0.0.1
cluster.srvb.queuelog=sql:B

cluster.srvc.manager=tcp:dial:12345@127.0.0.1
cluster.srvc.queuelog=sql:C
```

In this case, you would run Uniloader with the token "srva" on server A, and it would upload data to partition A:

```
./uniloader --src=/var/log/asterisk/queue_log
            upload --uri https://my.queuemetrics-live.com/MYINSTANCE
                   --login webqloader --pass CHANGEME --token=srva
```

The same goes for servers "srvb" and "srvc" that would upload to B and C respectively.

As you can see, each cluster member in QueueMetrics defines its own AMI credentials; so you can safely use the AMI feedback mode with no further configuration. Please note that it's common for all AMI instances to just point to "127.0.0.1" because Uniloader runs on the same box as Asterisk itself.

When configuring agents, make sure that you set the server for each agent, so when they log in to queues they already point to the right server.

Multiple Asterisk instances, one QueueMetrics instance

You have multiple Asterisk boxes and want to consolidate all their activity into a single QueueMetrics instance.

In this case, you need a cluster-enabled QueueMetrics instance, and each Uniloader instance should upload data to a different partition. Each cluster member should be defined in the QueueMetrics instance (if you use the MySQL back-end, you set the token to the name of the partition; if you use HTTP you should set it to the name of the cluster member).

The names for cluster members are the ones you use in the property *cluster.servers* of your QueueMetrics *configuration.properties* file.

Each cluster member in QueueMetrics defines its own AMI credentials; so you can safely use the AMI feedback mode with no further configuration.

One Asterisk instance, multiple QueueMetrics-Live instances

You run multiple different clients on one Asterisk instance, and you want to send each of them to their own QueueMetrics Live instance.

In this case, you need to set up splitting rules so that data for each client is uploaded to the right QueueMetrics Live instance.

For example, your *'splitter.json'* file could look like:

```
[
    {
        "uri": "http://my.queuemetrics-live.com/client3",
        "login": "webqloader",
        "pass": "CHANGEME",
        "token": "",
        "matcher": ["client3-"],
        "match": "any",
        "removematch": true,
        "disabled": false,
        "noactions": false,
        "clientname": "client3-"
    },
    ....other clients...
]
```

In order to make sure that Asterisk performs the correct actions at the AMI level, you must specify a "clientname" for each client and use that string in the Asterisk dialplan (where it is returned under the variable "UNILOADER_CLIENT") in order to build the actual queue / agent / channel name to be used on Asterisk.

So you would edit the stanzas you want to use in *'extensions_queuemetrics.conf'* to use the client name, like e.g.:

```
; extension 37: agent removequeuemember with hotdesking (for asterisk v1.4+)
exten => 37,1,Answer
exten => 37,2,NoOp( "QM: RemoveQueueMember (asterisk v1.4+) Agent/${AGENTCODE}
                     at extension SIP/${QM_AGENT_LOGEXT} on queue ${QUEUENAME}
                     made by '${QM_LOGIN}' for '${UNILOADER_CLIENT}'" )
exten =>
37,3,RemoveQueueMember(${UNILOADER_CLIENT}${QUEUENAME},SIP/${UNILOADER_CLIENT}${QM_AGE
NT_LOGEXT})
exten => 37,4,Hangup
```

So if this action is performed on "client3" removing extension "127" from queue "300", the actual action performed would be:

```
RemoveQueueMember(client3-300,SIP/client3-127)
```

That would produce a queue_log record like:

```
1487239051|1487239051.123|client3-300|SIP/client3-127|REMOVEMEMBER
```

But the splitter would then upload it to QueueMetrics as if it was:

```
1487239051|1487239051.123|300|SIP/127|REMOVEMEMBER
```

Because it would match the string "client3-" in both the queue and agent fields. This way each QueueMetrics-Live instance is blissfully unaware of the physical names for queues and agents that are used at the Asterisk level.

Also, as for some actions (chanspy and originate) QueueMetrics need to originate calls directly within your diaplan, you should edit the *configuration.properties* file so that channels where the client is required appear as:

```
callfile.monitoring.channel=SIP/$EM-!UNILOADER_CLIENT
callfile.outmonitoring.channel=SIP/$EM-!UNILOADER_CLIENT
callfile.customdial.channel=SIP/$EM-!UNILOADER_CLIENT
```

See also: Splitting a single queue_log file into multiple back-ends.

Multiple Asterisk instances, multiple QueueMetrics instances

If you have multiple Asterisk instances on which calls are processed, and calls for any client can be processed on each cluster member, you need to set up rewriting rules and create a cluster member (and related partition) on each destination QueueMetrics instance.

Make sure you use the "clientname" variable to have Asterisk perform the correct AMI calls.

Event tracking

Uniloader can be used to connect to an Asterisk server and generate queue events that Asterisk would not normally produce. This works by opening a stream of events from the Asterisk system through AMI and tracking call progress in real-time.

```
$ uniloader track -?

NAME:
   uniloader track - Tracks Asterisk events and creates relevant queue_log entries.

USAGE:
   uniloader track [command options] [arguments...]

OPTIONS:
   --host value                 Your Asterisk server (default: "127.0.0.1")
   --port value                 The AMI port on Asterisk (default: 5038)
   --login value                The AMI user as defined in manager.conf
   --secret value               The AMI secret [$AMISECRET]
   --rewriteuniqueids value     If 1 rewrites UniqueIds; else leave them unchanged
(default: 1)
   --debugfile value            A debug file to dump AMI data to
   --moh value                  When set to 1, tracks Music-on-Hold events on queues.
(default: 1)
   --parkedcalls value          When set to 1, tracks parked calls. (default: 0)
   --outboundcalls value        When set to 1, tracks outbound calls. (default: 0)
   --hotdesking value           When set to 1, rewrites hotdesking information.
(default: 0)
   --outboundthreshold value    The answer threshold (in ms) for calls to be tracked
as outbound. (default: 300)
   --noeventblacklisting value  When 1, AMI events are not blacklisted. Useful for
taking traces. (default: 0)
   --hidden-channels value      A regexp for outbound channels to be logged as hidden
queues.
   --pid value                  The PID file to write. If already present, won't
start.
```

This is meant to be run as a separate Uniloader process, parallel to the one that does data loading, with a separate PID, so that it can be started and stopped separately from the main process.

You should make sure that only one instance of the tracker is running for each Asterisk server, otherwise you will find duplicate events logged.

Restarting the tracker while calls are in progress will in general lead to incorrect data being logged, as some events may be lost. So event tracking should run unattended and be started as soon as Asterisk becomes available.

You can enable or disable different trackers at once, for example if you run:

```
./uniloader track .... --moh=1 --parkedcalls=1
```

It means you want both parked calls and MOH events tracked.

Generally speaking, calls will be tracked with their native Asterisk UniqueID prefixed with a colon; this avoids any possible conflict with manual call outbound or tracked calls that end up traversing queues. This prefix is automatically stripped by QueueMetrics when performing call actions, e.g. listening to recordings or hanging up calls.

This behavior is default after Uniloader 23.09.4 and should be completely transparent. If you don't want this behavior, just set `--rewriteuniqueids=0` to go back to logging unmodified Unique-Ids.

Installing event tracking through RPM

When installing the RPM package of Uniloader, two distinct services will be installed. Both of them rely on the same binary of Uniloader, but are otherwise completely separate.

As event tracking is still experimental, it is NOT started automatically.

Table 1. Services installed in RPM

Service	Description	Configuration file	Started on install?	Starts on reboot?
`uniloader`	Uploads queue_log	`/etc/sysconfig/uniloader`	Yes	Yes
`unitracker`	Tracks events	`/etc/sysconfig/unitracker`	No	No

In order to start tracking of events you need to:

- Configure which features you want enabled (see below)
- Make the service restart on reboots: `chkconfig unitracker on`
- Start the service: `service unitracker start`
- Check its logs in `/var/log/asterisk/unitracker.log`

The configuration file lets you set the credentials to use to connect to Asterisk and it lets you turn on specific features. By default, only MOH tracking is turned on by default.

These are the defaults - feel free to edit them as needed.

```
LOGFILE=/var/log/asterisk/unitracker.log
LOCKFILE=/var/lock/subsys/unitracker
PIDFILE=/var/run/unitracker.pid

AMIHOST=127.0.0.1
```

```
AMIPORT=5038
AMIUSER=admin
AMISECRET=amp123

#Uncomment to enable event logging
#DEBUGFILE=/var/log/asterisk/unitracker_events.log

#Only MOH tracking is enabled by default
ENABLEMOH=1
ENABLEPARK=0
ENABLEOUTBOUND=0

OUTBOUNDTHRESHOLD=300
```

Tracking Music-on-Hold

Uniloader can be used to detect and generate Music-on-Hold events for calls that are being handled on Asterisk queues.

In order to use it, you can launch it as:

```
./uniloader track --login admin --secret amp123 --moh=1
```

Where *admin* and *amp123* are the current AMI credentials for your local Asterisk system. At this point, you should:

- Send a call to an Asterisk queue
- Have an agent handle the call
- Have the agent start and stop music-on-hold

The event will appear in QueueMetrics on the real-time page.

When enabled, MOH events should appear correctly even when tracking calls in parking lots or for automated outbound.

Tracking Parking lots as queues

Uniloader can be used to track parked calls "as if" they were calls handled on a queue.

Parked calls are in a sense very similar to calls in a queue, because:

- You can define one or more separate parking spaces in the PBX
- Calls are parked at some period in time, and are waiting since then.
- The caller might decide to hang up before the call is served
- The call might time-out and be re-routed after a maximum wait time.

- Instead of being distributed by the ACD, calls are "picked up" by the agent who is willing to serve them. Agents are not "logged on" to a parking space in the same way as they are members of a queue.
- An agent may want to transfer a call back to the same (or a different) parking lot for further handling

So by converting events from parked calls to logs that "look like" logs from a queue, it may be possible to:

- See those calls on the Real-time page of QueueMetrics
- Run a wallboard on them
- Run reports on them

The main differences from actual queues are that:

- Agents must be able to see the *park access code* to fetch a call back from the parking lot. This is handled by prepending the caller's number with the park code. By using the wallboard or real-time pages in QM you can then see who is parked.
- The *queue name* comes from the name of the parking lot prepended with a string that means it's a parking lot; for example, calls on lot "default" will be tracked as belonging to queue "pk-default". Queue security features of QM can be used as usual.
- The feeding of a parking lot is usually by some agent transferring the call from an inbound queue. **It is important that such transfer produces a call closure record on the queue**, so that we can track the call correctly and the first call ends before the parking lot starts. This usually works in FreePBX if the physical interface that is connected to a queue is a SIP extension.
- As the agent is not logged on to the parking lot when they pick a call, and QueueMetrics works best when agents are logged in to handle calls, we write a log-in record when the call is picked and we produce a log-off record when the call completes.
- If a call is *transferred* from a parking lot to a different one, we write call and session closure records before opening new ones.
- In Asterisk, calls transferred (on parking lots, or elsewhere) will usually have a different Unique-id from their previous one. At the moment we make no provision to "reuse" the same Unique-id across multiple transfers.

In order to use it, you can launch it as:

```
./uniloader track --login admin --secret amp123 --parkedcalls=1
```

Where *admin* and *amp123* are the current AMI credentials for your local Asterisk system. At this point, you should:

- Create a queue in QueueMetrics with the name of your parking lot (e.g. "pk-default" for the default lot)
- Transfer a call to the parking lot

- Have an agent pick up the call

The call should appear on the real-time page of QueueMetrics with the pick-up code as if it was a call on a queue.

Automatically Tracking Outbound Calls

Uniloader is able to track all calls on the system "as if" they were calls made on a queue, so that they become visible to QueueMetrics.

In order to use it, you can launch it as:

```
uniloader track --login admin --secret amp123 --outboundcalls=1
--outboundthreshold=300
```

In order to "trim down" the number of calls tracked, a call is only tracked if there is a slight delay between its set-up and someone answering it, like a human picking up would do. This way calls to internal PBX "service" numbers are not tracked. You can adjust the threshold as needed though the `outboundthreshold` parameter.

As all calls must belong to a "campaign" in order to be tracked, Uniloader tries to determine the "campaign" based on the account code of the extension currently calling. This way, by setting up different account codes for different groups of people, you can control reporting and visibility of calls in QueueMetrics. If an account code is set, the call appears on campaign `q-ACCOUNTCODE`; if no account code is set, it appears on `q-outbound`.

If a tracked call enters a queue on your PBX; then its logs are closed and the rest of the call is tracked as a normal inbound call.

Uniloader will also join the agent to the supposed queue and log her off by the end of the call, so that reports and realtime monitoring in QueueMetrics appear correct.

Hiding sets of calls

Sometimes, a lot of outbound calls (e.g. outgoing trunks) will be picked up by Uniloader but are not really needed. In this case, if you can spot a pattern, you can tell Uniloader to "hide" those calls by logging them on a queue name that starts with an underscore.

In this way, information is not lost, but it will not be visible in QueueMetrics unless you manually create a queue with the correct name.

In order to enable this mode, you should use the parameter:

```
uniloader track --login adm --secret 123 \
          --outboundcalls=1 --hidden-channels '^PJSIP/'
```

This way, all calls where the channel starts with PJSIP/ will be logged on a queue that starts with an underscore, usually the default `q-outbound` that will then become `_q-outbound`.

As the parameter is a regular expression, you can use it to create complex filters, for example:

```
(?i)^sip/|^zap/
```

Will match (and hide) all channels starting with SIP/ or ZAP/, in a case-insensitive way.

Be aware of the fact that you usually think in terms of a set of *extensions* you want to hide, but Uniloader is monitoring the names of live *channels*. Those usually have names that start with the name of the extension followed by an unique identifier; for example a call made by `PJSIP/301` could show up as `PJSIP/301-00000014` - the exact format depends on the channel type being used and the Asterisk version. To make sure you are doing it right, you should start by going to the PBX and observe how your extensions appear when they are active and you issue the command `core show channels`.

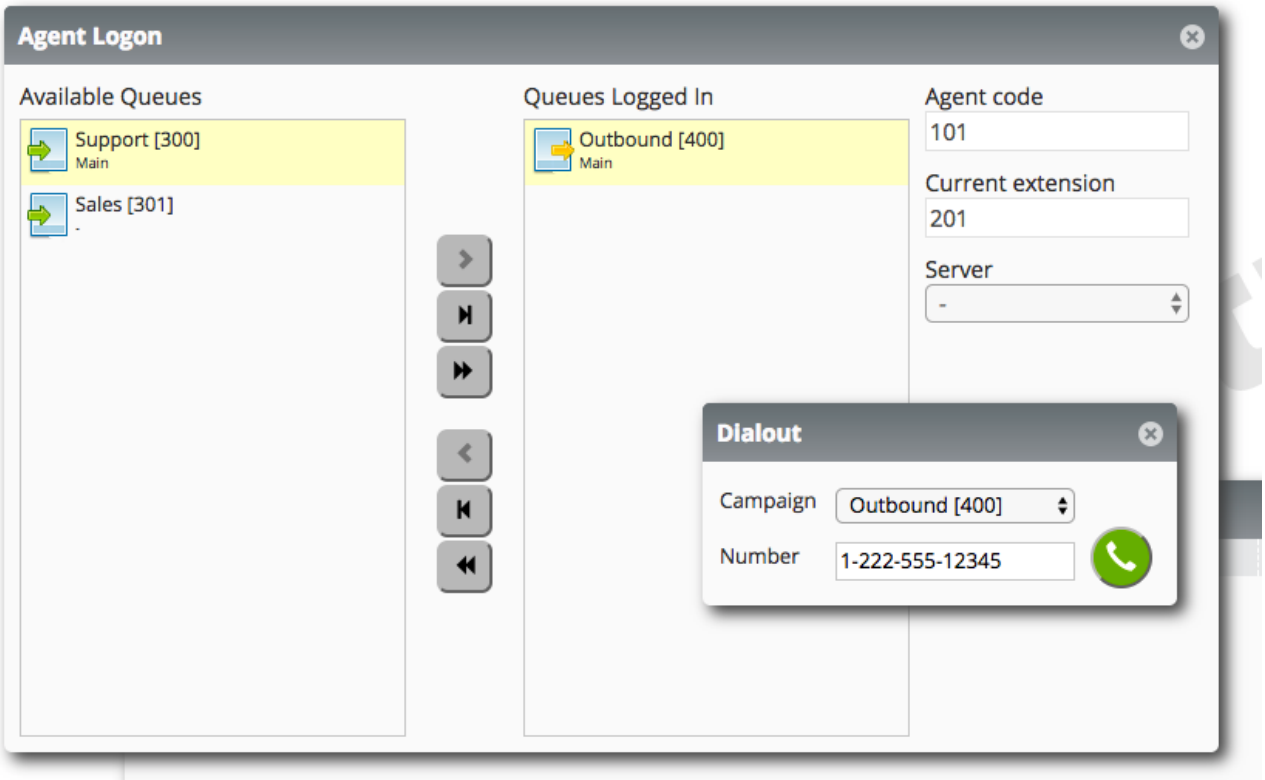
You can use the Regexp tester to develop custom regular expressions without the need to relaunch Unitracker for each test. It lets you test the same regexp on a number of channels at once, so you can check that it matches only the ones you want matched.

Automatically tracking outbound queues in QueueMetrics

When you run automated tracking as described above, if you use the QueueMetrics agent page for your agents to dial out there is no need to include the outbound dial-plan.

You will have to create some specific physical queues in Asterisk to be used as "placeholders" for outbound campaigns; and then you must make QueueMetrics aware of them and set them as "outbound" queues.

From the Icon agent page, you will then be able to dial out by selecting the Dialout panel, choosing one of the outbound queues you are logged on to and entering a number to be dialed.



In order to turn this feature on, you will have to enable:

- DirectAMI
- Outbound
- Tracker outbound

The following configuration can be a good starting point for a FreePBX system:

```
default.hotdesking=86400

platform.pbx=DIRECTAMI
platform.directami.agent=Agent/${num}
platform.directami.extension=SIP/${num}
platform.directami.transfer=${num}@from-internal
platform.directami.outbound.enabled=true
platform.directami.outbound.usetracker=true
platform.directami.outbound.trackerdialout=${num}@from-internal
platform.directami.localext=SIP/${num}
platform.directami.verbose=false
```

Please refer to the QueueMetrics User Manual for a complete description of how DirectAMI works and which options you can use.

Automatically tracking hotdesking events

If you use an external tool to login your agents to Asterisk, and these agents have a name set, you may want to enable tracking of hotdesking.

If hotdesking tracking is enabled, when an agent "joe" is logged on at SIP/71045 and produces an entry like:

```
1552380101|MANAGER|callcenter|SIP/71045|ADDMEMBER|
```

The following entries are added to "reverse" the log-in and create a new one with proper hotdesking:

```
1552380101|MANAGER|callcenter|SIP/71045|REMOVEMEMBER|
1552380101|MANAGER|callcenter|joe|HOTDESKING|SIP/71045|
1552380101|MANAGER|callcenter|joe|ADDMEMBER|
```

Debugging missed events and incorrect logging

If you find that some calls are generating wrong `queue_log` events - say, your Music-on-Hold periods are not being tracked - Loway may ask you to record a dump of all events generated for further inspection by the dev team.

It definitely helps if you are able to find a pattern when something is consistently not working - like, it always happens on Local channels and never on PJSIP channels.

In this case, you run the following command on the shell:

```
./uniloader track \
  --host "127.0.0.1" --port 5038 --login value --secret value \
  --noeventblacklisting 1 --moh 0 --debugfile events.txt
```

Where:

- `host`, `port`, `login` and `pass` are the usual AMI credentials to connect to your Asterisk PBX. The command may be run locally or from a remote server, as much as the AMI is reachable. To test whether a connection is working, see AMI Connection test.
- the other options tell Uniloader to create a debug file called *events.txt* that records AMI traffic.

You can run the command even as Uniloader/Unitracker are running - as it does not perform anything on the PBX but receiving all events, it has no side effects and will cause no event duplication.

Now, while the command is running, place a call and use the function that you want to check (example for our case: send a call to a queue, have the agent put it on-hold and off-hold, terminate

the call). After the call ends, stop the command by pressing Ctrl+C.

When you are done, you should send Loway:

- The file that was just created (in our case, *events.txt*)
- Your *queue_log* file that was produced (or at least its relevant lines)
- An indication of which call is displaying wrong events - e.g. "we called queue 300 from extension 200, then agent 201 answered and put the call on hold". Always use the actual Asterisk codes - we have no way of knowing that Jack is extension 200 and Joe extension 201.

When running in debug mode, a large file is quickly generated; so it is appropriate to run it only for **short periods** and when the system is **otherwise idle**. Make sure no other traffic is running at the time of your tests, as AMI traces quickly become unreadable.

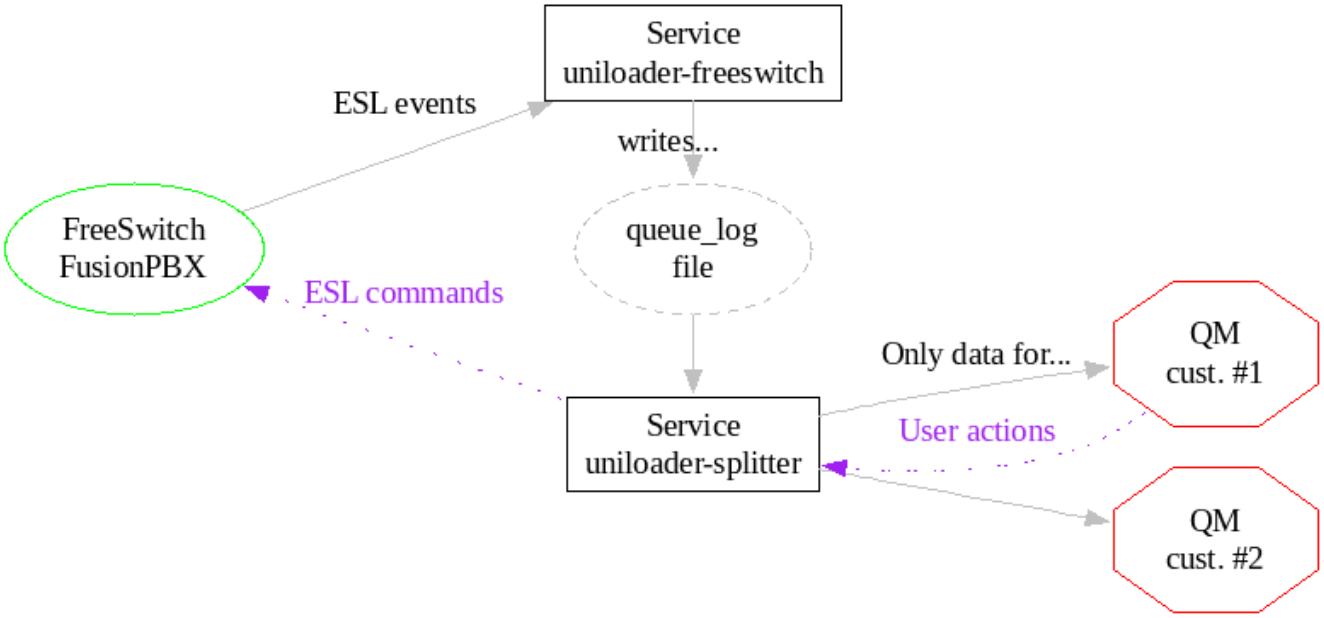
FreeSwitch / FusionPBX support

Uniloader can work with FreeSwitch to act as an adaptor between its *mod_callcenter* features and QueueMetrics. In particular, when using FusionPBX, events are divided by *domain* as to be able to feed them to multiple QueueMetrics Live systems, one for each tenant who subscribed the service.

General overview

Uniloader is able to build a synthesized queue_log file out of a stream of events flowing from a FreeSwitch system. This queue_log file can then be uploaded by a regular Uniloader process to a single QueueMetrics instance, or can be split by tenant and pushed to multiple, separate QueueMetrics Live instances.

Agent state is controlled by QueueMetrics by sending ESL commands to implement agent actions (eg. log-on or log-off).



In order to enable this feature, you need to run two copies of Uniloader in parallel, as different services:

- one runs `uniloader fsw`: it connects to FreeSwitch over ESL, reads events from *mod_callcenter* and generates a queue_log file
- another one runs `uniloader upload`: it reads the generated queue_log file and uploads it to one or more QueueMetrics systems, splitting data as appropriate between different tenants

The system was designed to be run as two different services because log generation is based on events streamed in real-time, so the idea is that the service is run at all times when FreeSwitch is active. Upload also happens in real-time, but it can be restarted with no data loss if you need to change its configuration - e.g. adding or removing a tenant.

You may also want to run a third service with Uniloader in AudioVault mode, setting it in multi-tenant mode, as to make audio recordings created in FusionPBX

available from the QueueMetrics GUI - see AudioVault.

Automatic outbound tracking

Uniloader performs *automatic outbound tracking*, that is, any call between extensions that is not handled on a queue will be tracked as an outbound call. They will be tracked on a queue named `q-outbound` and agents will be logged on to it before the call starts and logged off after the call ends - so you will see them logged on to that queue only when a call is happening.

It is possible to control which calls are tracked - or, if you prefer, which calls are hidden - by setting one of more of the command line options: `--outbound-include-caller`, `--outbound-exclude-caller`, `--outbound-include-callee` and `--outbound-exclude-callee`, each of which takes a regular expression.

For example, you might want to track only outbound calls where the dialed number starts with zero, if you knew that on your PBX all numbers beginning with 0 are routed through an outgoing SIP trunk, making them true external calls.

So you would use:

```
uniloader fsw .... --outbound-include-callee '^0.+'
```

If instead you do not want any calls to be tracked, you could use:

```
uniloader fsw .... --outbound-exclude-caller '^.*'
```

As this regexp matches any possible caller extension, no automatic outbound calls will be logged for this case.

The numbers being matched are the actual extensions or telephone numbers being used - so even if the caller might in the end be logged as `Agent/1234`, the caller check will try matching only extension `1234`.

When developing a regular expression, remember that Uniloader can help you nail it - see Regexp tester. Also remember that in order to be passed as a command-line parameter, the regexp must be properly shell-encoded - usually enclosing it in single-brackets is enough. To avoid mistakes, the actual regexps in use (if any) are printed when `uniloader fsw starts`. If unset, they will be printed as "All", as all numbers would be valid.

You can specify multiple conditions at once:

- If you specify conditions for both caller and callee extensions, they both must match for the call to be logged.
- For any kind of extension, if you specify both *include* and *exclude* conditions, first *include* is applied, and then *exclude*. So if your *include* condition was `^1..` and *exclude* was `^19.`, extension "132" would be tracked, but "192" would not.

The conditions specified apply to all tenants at once.

QueueMetrics outbound tracking

In QM mode, an outbound call is started though the QueueMetrics GUI. This way the agent can specify not only the extension to be dialed, but also the "outbound queue" on which it should be placed. A call is then sent to the agent and, when they pick it up, the actual outbound call is started.

This gives you two advantages:

- The agent can be more granular in specifying why they are calling someone - if the outbound queue is called "SalesOut" the reason is different from "Support" - as you would do with incoming queues
- You can use AMO (Automatic Manual Outbound) that works as a very simple, one-click dialer, with recall rules, scheduling, etc.

When a call happens on Active Outbound, the agent is supposed to be logged in to an existing queue with the correct name. So we suggest creating those queues on Fusion as placeholders and then not sending any calls to them.

The details of how a call should be placed are specified in the QueueMetrics documentation. Uniloader just applies them.

Automatic music-on-hold tracking

On any tracked call, be it inbound or outbound, music on hold events are tracked in real-time and visible on QueueMetrics. You do not have to configure anything - it just happens.

Call actions

A supervisor with the correct grants can use Queuemetrics to:

- hang-up existing calls, whether they are connected or waiting on a queue
- manually transfer existing calls
- start a spy, whisper or barge-in session on any call

These features are controlled by QueueMetrics and require no configuration on Uniloader.

Agent status sync

It is possible at any time to ask Uniloader to query the current status of all agents on Freswitch and write it out. This is helpful if:

- Your agent status appears to be out of sync, and
- You have long-standing agents and they are not visible on QM until they take their first call in the morning

By creating a cron-job that runs this soon after midnight, you can have Uniloader rewrite the

current status of all agents.

This is similar to Asterisk queue-replicate.

To have Uniloader replicate all events, it must be running in `fsw` mode - an then you launch it from a shell like:

```
$ uniloader fs-queue sync --host 127.0.0.1 --port 8021 --auth ClueCon

2025/05/06 18:06:34 Asking for a queue refresh on 127.0.0.1:8021 with pass '*******'
2025/05/06 18:06:35 Event was sent in 34 ms - all went well
```

This actually inserts a request on the FreeSwitch event bus, so that all agent status is written to the queue_log file by `uniloader fsw`.

System diagnostics

It is possible to ask Uniloader to print the current status of all agents and all ongoing calls. This works by inserting an event on the Freeswitch event bus.

```
$ uniloader fs-queue daemon-status

2025/05/06 18:12:11 Asking Uniloader Fsw on 127.0.0.1:8021 with pass '*******' to log
its status
2025/05/06 18:12:11 Event was sent in 58 ms - all went well
```

And then on the service's output (usually available via `journalctl -u uniloader-fsw` or a similar command) you would see an output line for each agent session and open call:

```
2025/05/06 18:12:11 ========== Γνῶθι σεαυτόν - NOSCE TE IPSVM ===========
2025/05/06 18:12:11 Open Calls: 1 - Open agents: 3 (waiting for tiers: 0)
2025/05/06 18:12:11 Ag:b03660af-0b21-4bb0-833f-3791b8ec021c
C:user/997@tenant1.server.my Available,Waiting Qs:NONE
2025/05/06 18:12:11 Ag:c476c911-6280-4e89-8519-67f0db15c854
C:user/998@tenant1.server.my Available,Waiting Qs:401@tenant1.server.my:0-0
2025/05/06 18:12:11 Ag:e5dc2fbc-6526-4f1f-8389-642901876a0e
C:user/999@tenant1.server.my Available,Waiting Qs:400@tenant1.server.my:0-0
2025/05/06 18:12:11 C:4f67a(ma)-Q=OUTBOUND@tenant1.server.my-
A=Agent/998@tenant1.server.my-r=40915-n:
2025/05/06 18:12:11 -----------------  That's all folks -----------------
```

- For each agent you can see their current id, user, status and state, and a list of queues they are known members of.
- For each call, you can see their current queue and agent

Agent and queue translation

With FusionPBX, `uniloader fsw` can be given access to the FusionPBX's Postgres database, in order to decode queue and agent names (that appear as UUIDs) into their own tenant and extension.

For example, a queue with id `75082016-6394-4738-b896-b9121c060612` that belongs to domain (tenant) *abc.example.com* where it is reachable under extension 200 will be logged as `abc.example.com-200`. An agent working from extension 300 under the same tenant will appear as `Agent/abc.example.com-300`.

On current versions of FusionPBX (5.x), some queue events are logged with their UUID and some use the format `agent@domain`. Starting from Uniloader 21.04.5 onwards, such events are recognized and translated trasparently. If you get a crash that says `invalid input syntax for type uuid: "1600@tenant.server.com"`, you have an older Uniloader that needs to be upgraded.

When uploading data, you can then use the splitter feature to send only data for domain *abc.example.com* to a QueueMetrics Live instance named (for example) *my.queuemetrics-live.com/customer-abc*, where the data mentioned above appears as queue code *200* and agent code *Agent/300*.

While the settings work for FusionPBX, it is possible to override the database and the queries used by setting `sql-agent` and `sql-queue`. For example, if we run:

```
uniloader fsw --queuelog my_log.txt --ps-uri 127.0.0.1/fusionpbx
    --shorten-domain 1  --sql-agent 'SELECT 1000 as TENANT, $1 as AGENT'
```

we will always get back the same agent and a tenant "1000". If your logic is on MySQL, we could get the same result by issuing:

```
uniloader fsw --queuelog my_log.txt --ps-uri "10.10.5.10/sugarcrm"
        --ps-database "mysql" --ps-login queuemetrics --ps-pwd javadude
        --shorten-domain 1
        --sql-agent 'SELECT "abcd" as TENANT, ? as AGENT'
```

This will return the same agent with a tenant called "abcd".

Any query that receives a single input value and returns two columns will work. The first column is the tenant, while the second one is the decoded queue or agent id.

Disabling database translation

Database translation is needed from FusionPBX 4.4 onwards. On earlier versions, you may want to switch it off by explicitly setting the `ps-uri` option to blank or a dash, as in `--ps-uri "-"`.

Writing custom queries

MySQL and Postgres use a different format for allowed SQL strings:

- The placeholder used in Postgres is `$1` while Mysql uses `?`.
- The placeholder in Postgres can be used multiple times. On MySQL you have only one, so you need to write a Common Table Expression instead, as shown in the examples below.
- Literal strings must be quoted with a single quote in Postgres `'` while MySQL also allows a double-quote `"`.

The names of fileds don't really matter, as Uniloder will take the first one as the tenant and the second one as the queue or agent.

Examples

Split an entry like `1600@tenant.server.com` into queue and tenant (Postgres):

```
SELECT split_part($1, '@', 2) as TENANT, split_part($1, '@', 1) as QUEUE
```

Split an entry like `1600@tenant.server.com` into queue and tenant (MySQL). As we cannot reuse the placeholder, we support the Common Table Expression syntax, where we create a literal table `cte` with a single column named `v` and only one row of data that contains our value:

```
WITH cte (v) as (values ( ? ))
SELECT substring_index(v, '@', -1) as TENANT, substring_index(v, '@', 1) as QUEUE
FROM cte
```

Given an agent code like `1000`, add a fixed tenant "ABCD" (MySQL):

```
SELECT "abcd" as TENANT, ? as AGENT
```

Deprecated: ADDMEMBER mode

With Uniloader 25.05, agent login is always logged on a queue-by-queue basis. Queue-by-queue login is available from the QueueMetrics GUI, while the systems remains compatible with any change you make on the FusionPBX GUI. So this option should not be used.

FusionPBX does not allow the selection of which queues an agent is supposed to work on; an agent becomes available (or unavailable) on all of the queues they are configured on.

In terms of reporting, this means that QueueMetrics will display an agent as available on a fake queue called `* ALL *`, as we have no information on which queues an agent is actually working on.

If you enable ADDMEMBER mode by passing `--use-addmember "1"` to `uniloader fsw` (requires Uniloader 21.04.6), then each time an agent logs on or off, their current set of queues is queried on the FusionPBX database, and separate log-on records are emitted for each queue. This gives QueueMetrics the information needed to display the current set of queues an agent is working on.

This causes a database access on each agent log-on or log-off; the query run can be overridden

though `sql-qs-for-agent`, that given the agent's UUID, should return the UUIDs of all queues this agent is supposed to be working on. The UUIDs for agent and queues will then be translated through the normal mechanism described above.

This does not change the fact that logging on, logging off or pausing happen at once on all queues, and never separately. A possible workaround is explained in our **FusionPBX Integration Guide**.

Setting up

You can install Uniloader normally; make sure you enable both services `uniloader` and `uniloader-fsw`. Both will be active at once: `uniloader-fsw` wil create a "fake" queue_log file and `uniloader`, as always, will upload it to your QM server(s).

You can also run it manually to test it.

```
$ ./uniloader fsw -?

NAME:
   uniloader fsw - Parses FreeSwitch mod_callcenter events

USAGE:
   uniloader fsw [command options] [arguments...]

DESCRIPTION:
   This command listens on FreeSwitch's Event Socket.

It reacts to mod_callcenter events and attempts to coerce them
to queue_log format, to make them compatible
with QueueMetrics.
It only generates a queue_log file; it should then be
uploaded by a separate instance of 'uniloader upload'.

OPTIONS:
   --host value                    Your FreeSwitch server (default: "127.0.0.1")
   --port value                    The ESL port on FreeSwitch (default: 8021)
   --auth value                    The ESL auth secret (default: "ClueCon") [$AUTH]
   --queuelog value                The queue_log file to write
   --events value                  A debug file to dump mod_callcenter events to
   --allevents                     Read (and possibly log) all events, not just the
ones we need
   --ps-database value             The kind of database we  can connect to. (default:
"postgres")
   --ps-uri value                  A FusionPbx database to connect to (default:
"localhost/fusionpbx")
   --ps-login value                A FusionPbx database user (default: "fusionpbx")
   --ps-pwd value                  A FusionPbx database password [$FUSIONPWD]
   --sql-agent value               The query used to extract (tenant,agent) from
agentId. Blank for default.
   --sql-queue value               The query used to extract (tenant,queue) from
queueId.  Blank for default.
```

```
   --sql-qs-for-agent value         The query to extract queue UUIDs that agentId
works on.  Blank for default.
   --outbound-include-caller value  A regexp to be applied on the caller. Only numbers
matching it will be included.
   --outbound-exclude-caller value  A regexp to be applied on the caller. Any numbers
matching it will be rejected.
   --outbound-include-callee value  A regexp to be applied on the callee. Only numbers
matching it will be included.
   --outbound-exclude-callee value  A regexp to be applied on the callee. Any numbers
matching it will be rejected.
   --shorten-domain value           If 1, the domain will be shortened (default: 0)
   --use-addmember value            If 1, agent log-ons will be logged queue-by-queue.
(default: 0)
   --pid value                      The PID file to write. If already present, won't
start.
```

When setting up:

- The `--queuelog` option should create a queue log file. It can be put anywhere - you must make sure that it is the same location that will be read by the `uniloader` service
- The file `--events` is optional, but we suggest creating it so anomalies can be tracked. If you add `--allevents` it will process and log all events being created on the PBX and not only the ones it needs.
- If you use FusionPBX, credentials to the database can be entered in `--ps-uri`, `--ps-login` and `--ps-pwd`. If you don't want to use it, set *ps-uri* to a single dash. The option *ps-database* can be set to `postgres` or `mysql`.
- You can specify custom agent resolution queries in `--sql-agent`, `--sql-queue` and `--sql-qs-for-agent`. See Custom agent and queue translation.
- The options `--outbound-include-caller`, `--outbound-exclude-caller`, `--outbound-include-callee` and `--outbound-exclude-callee` let you pick which outbound calls you want tracked automatically. See Automatic outbound tracking.
- The `--shorten-domain` option will try shortening the domain name to the first element in it, e.g. *abc.example.com* will be shortened to *abc*.
- The `--use-addmember` option will generate ADDMEMBER records instead of AGENTLOGIN - see ADDMEMBER. Currently deprecated.
- The `--pid` is a file to write the current PID to.

The database connection and the ESL connection can be checked using `uniloader test postgres` and `uniloader test fsw-esl`.

If you see that calls generated by `uniloader-fsw` are incomplete - that is, they seem to be made only of an *ENTERQUEUE* verb and nothing else, it's likely that there is an issue on your database when trying to decode unique-ids to agent codes. As the service restarts automatically after each crash, you will always see the `uniloader-fsw` service as "up", but its logs will show a lot of restarts.

Setting up mult-tenant systems with QueueMetrics Live

When using QueueMetrics Live with multiple instances on a multi-tenat system, you need to run the `uniloader` service as:

```
./uniloader -s qlog.txt  u  -x splitter_rules.json
```

Where *splitter_rules.json* is a file that contains multiple tenants, defined as:

```
[
    {
        "clientname": "Acme Company Ltd",
        "uri": "https://my.queuemetrics-live.com/acmeco",
        "login": "webqloader",
        "pass": "itsasecret",
        "token": "",
        "matcher": ["acmeco-"],
        "match": "any",
        "removematch": true,
        "disabled": false,
        "noactions": false
    }
]
```

Note that:

- *uri*, *login* and *pass* are the ones that you are given for your QueueMetrics Live instance
- *matcher* contains the domain (tenant) and a trailing slash.
- *clientname* is not needed in this scenario, but we suggest setting it for readability

You can safely restart the service when you make changes to the rules, as data is queued on the *queue_log* file.

You do not need to have all tenants configured; only the ones that match will be fed, and other data will be ignored. If you create a new tenant, and there is existing data for it on the log file, it will be uploaded on the first run.

A complete explanation of the splitter logic is available at Splitter.

Enabling user actions

If you want, your QueueMetrics system can send login/logoff actions back to your FreeSwitch server. An explanation of how this works at AMI Feedback.

In QueueMetrics you need to set (at least) the following properties:

```
callfile.dir=fsw:ClueCon@127.0.0.1
```

```
default.webloaderpbx=true
platform.pbx=FREESWITCH
```

You do not need to include any dial-plan, as actions work directly.

In QueueMetrics, you also need to enter the "External Reference ID" identifier in the Agent (and possibly Queue) page, as this code will be used to generate ESL login/logoff commands. The external reference for queues and agents is easily found and can be uploaded automatically to QM by running `uniloader pbxinfo fusionpbx`, as explained in PbxInfo for FusionPBX.

Diagnostics and tools

Uniloader is meant to help automate a number of little tasks that pertain to administering and running a QueueMetrics system.

Diagnostics: AMI connection test

Uniloader lets you test an AMI port from the command line. It will also check that the *queuemetrics* context is present on the system, and will make sure that the AMI user has the required "originate" privilege.

```
$ ./uniloader test ami -?

NAME:
   uniloader test ami - Tests an AMI connection

USAGE:
   uniloader test ami [command options] [arguments...]

DESCRIPTION:
   This command test an AMI connection.

It checks that the `queuemetrics` context is present and its
functions are present. It checks originates to `10@queuemetrics`
and prints available queues.

OPTIONS:
   --host "127.0.0.1"               Your Asterisk server
   --port "5038"                The AMI port on Asterisk
   --login                  The AMI user as defined in manager.conf
   --secret                     The AMI secret [$AMISECRET]
   --testChannel "Local/10@queuemetrics"    The channel to use when testing
originates.
   --testExtCtxt "10@queuemetrics"      The ext@ctxt to use when testing originates.
```

In order to use it, you can call it like:

```
$ uniloader test ami --login admin --secret amp111
```

It will print out a comprehensive report, like:

```
Testing AMI connection to 10.10.5.27:5038 - Username 'admin' secret '******'
AMI Connected: Asterisk Call Manager/1.3

  N.   Meaning                             ext@queuemetrics         Present?
-_-------------------------------------------------------------------------
```

```
    1    Dummy extension                          10                Ok
    2    Chanspy □ inbound calls                  11                Ok
    3    Sets a call status                       12                Ok
    4    Chanspy □ outbound calls                 14                Ok
    5    Add call feature                         16                Ok
    6    Remove call feature                      17                Ok
    7    Agent pause                              22                Ok
    8    Agent unpause                            23                Ok
    9    AddMember                                25                Ok
   10    RemoveMember                             26                Ok
   11    Custom dial                              28                Ok
□  12    Send SMS to SIP device                   29                MISSING
   13    Soft hangup of call in queue             30                Ok
   14    Redirect call in queue                   31                Ok
   15    Agent pause with hotdesking              32                Ok
   16    Agent unpause with hotdesking            33                Ok
   17    Addmember with hotdesking                35                Ok
   18    Removemeber with hotdesking              37                Ok

Known Queues                                Completed   Abandoned
-_-----------------------------------------------------------------
 - 300                                            10           5
 - 301                                             0           0
 - default                                         0           0
 - 400                                             0           0

Originate on 10@queuemetrics worked.
```

This shows:

- The version of AMI in use
- Whether all default queuemetrics extensions are present, and which ones are missing
- The queues configured in Asterisk, and their current usage statistics
- Whether the user has "originate" privileges

If connection is possible, it returns with a status code of zero; if not possible, or wrong credentials are used, it returns with an error code so that you can script it.

Originating custom channels

It is possible to use have Uniloader originate arbitrary channels on the PBX by telling it the channel and the extension and context to connect.

```
$ uniloader amitest --login admin --secret 123 --testChannel SIP/701 --testExtCtxt
706@from-internal
```

In the example above, first channel *SIP/701* is brought up, and then it is connected to extension *706* in context *from-internal*.

Diagnostics: Test upload link

If you want to make sure that your upload credentials to a server (either HTTP/S or SQL) are working, you can test them by using:

```
$ ./uniloader test upload -?

NAME:
   uniloader test upload - Tests a data upload connection

USAGE:
   uniloader test upload [command options] [arguments...]

OPTIONS:
   --uri, -u            The connection URI. Valid URIs start with file:, mysql:,
http:, https:
   --login, -l "webqloader" The login for your connection
   --pass, -p "qloader"     The password for your connection [$UPASSWD]
   --token, -t          In MySQL mode, the partition. In HTTP/S mode, usually blank or
server-id
   --timeout "10"       The time-out to wait for (in seconds) on errors.
```

If the command runs and succeeds, it will print out the current high water mark for the back-end and return with a status of zero. If there is any error, or the format of the connection is invalid, it will return with a status different than zero.

For example, this command tests a local database that contains data:

```
$ uniloader testupload  --uri
"mysql:tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1" \
                        --login queuemetrics --pass javadude --token P001
Testing upload credentials.
2017/07/27 14:28:14 Error: no db object
2017/07/27 14:28:14 Assert: DB Connection works
2017/07/27 14:28:14 [,P001] Driver error: retrying in 200 ms
```

```
High Water Mark is 1472824811 [2016-09-02 16:00:11 +0200 CEST]
Connection OK
```

As back-ends keep on retrying for errors automatically, the tool waits for a missed answer within *timeout* seconds before giving up and marking the connection as invalid.

If the driver supports it, the test also prints:

- The complete version of the remote QueueMetrics server

- The time it took to establish a connection
- The current time, according to the server
- The current time, according to Uniloader

For example:

```
Server version : 23.09.8 / 141 - 2024.01.16-12:11 - ec6180d2ef6@rel_23_09_m
Connection time: 158 ms
Server time    : Fri Apr 05 19:27:22 CEST 2024
Uniloader time : Fri Apr 05 19:27:22 CEST 2024
```

If there is a difference of more than one second (no matter the time zone) between your QM server and Uniloader, you are in for some anomalies in real-time displays. Make sure that both servers are aligned using NTP - as QueueMetrics Live servers are.

Diagnostics: Test connection to Mysql/MariaDB database

This tool will check that you have a working connection to your Mysql or MariaDB database, and that the credentials you use are correct.

```
NAME:
   uniloader test mysql - Tests a MySQL/MariaDB connection

USAGE:
   uniloader test mysql [command options] [arguments...]

DESCRIPTION:
   This command tests connection credentials.

It just connects to the database and runs an empty query
to confirm everything is working as expected.

OPTIONS:
   --dburi "tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1" The database to
connect to
   --login "root"             A database user
   --pwd                A database password
```

An example run:

```
uniloader test mysql --dburi 10.10.5.27/somedb --login user --pwd pass
```

Will print:

```
2019/04/02 09:04:13 Testing MySQL connection to
'user:pass@tcp(10.10.5.27:3306)/somedb?allowOldPasswords=1'

2019/04/02 09:04:13  -- Connection took 54.183µs
2019/04/02 09:04:13  -- Query took 21.387871ms
2019/04/02 09:04:13 Local time on database is: 2019-04-02 09:04:13
```

Diagnostics: Test connection to Postgres database

This tool will check that you have a working connection to your Postgres database, and that the credentials you use are correct.

```
$ uniloader test postgres -?

NAME:
   uniloader test postgres - Tests a Postgres connection

USAGE:
   uniloader test postgres [command options] [arguments...]

DESCRIPTION:
   This command tests a Postgres connection.

It just connects to the database and runs an empty query
to confirm everything is working as expected.

OPTIONS:
   --ps-uri "localhost/fusionpbx"   A Postgres database to connect to
   --ps-login "fusionpbx"       A database user
   --ps-pwd                 A database password [$FUSIONPWD]
```

Example run:

```
uniloader test postgres --ps-uri 10.10.5.182/fusionpbx --ps-login fusionpbx --ps-pwd
""
```

Will print:

```
2019/02/22 09:59:32 Testing Postgres connection to
'postgres://fusionpbx:@10.10.5.182/fusionpbx'

2019/02/22 09:59:32  -- Connection took 298.534µs
2019/02/22 09:59:32  -- Query took 21.780999ms
```

```
2019/02/22 09:59:32 Local time on database is: 2019-02-22T00:50:58.357799+01:00
```

Connection tips and tricks

- If you know that the database credentials are correct and you get a connection error that says `unknown authentication response: 10`, it is likely that your database is using SCRAM-SHA-256 authentication. This is supported in Uniloader from version 23.09.3 onwards - in case, update Uniloader and try again.
- If you have an error connecting that says `SSL is not enabled on the server`, you may bypass it by disabling SSL - e.g. by changing your ps-uri to `localhost/fusionpbx?sslmode=disable`. Of course this means that the connection will be unencrypted, so you may want to enable SSL on the server instead.

Diagnostics: Open a SQLite database

This tool opens a SQLite file (read-only), lists the tables it contains with their number of rows and the space they use, and prints some general information on the database. It never modifies the file.

Use `-f` to give the name of the file:

```
$ uniloader test sqlite -f /tmp/x.db

Tables in '/tmp/x.db':
+-------+------+--------------+
| TABLE | ROWS | SIZE ON DISK |
+-------+------+--------------+
| alpha | 1000 |     160.0 KB |
| zeta  |    0 |       4.0 KB |
+-------+------+--------------+

SQLite version 3.51.2 - Space used: 164.0 KB - Pages 41 (free 0) @ 4096 bytes each -
Journal: delete
  Application: '' - User version: '' - Encoding: UTF-8 - Quick check: ok
```

The size of a table does not include its indexes. `Quick check` is `ok` when no problem is found in the file; on very large files it may take some time.

Diagnostics: Test Freeswitch's ESL port

This tool tries connecting to Freeswitch's ESL port and tries displaying queues and agents defined.

```
$ uniloader test fsw-esl -?

NAME:
   uniloader test fsw-esl - Test Freeswitch's ESL port

USAGE:
```

```
   uniloader test fsw-esl [command options] [arguments...]

DESCRIPTION:
   This command tries to connect to FreeSwitch's Event Socket.

OPTIONS:
   --host "127.0.0.1"   Your FreeSwitch server
   --port "8021"    The ESL port on FreeSwitch
   --auth "ClueCon" The ESL auth secret [$AUTH]
```

For example:

```
uniloader test fsw-esl --host 127.0.0.1 --port 8021 --auth ClueCon
```

Will display a successful dialog:

```
2019/02/22 10:01:37 Testing Freeswitch connection to '127.0.0.1:8021' with auth token
'ClueCon'

2019/02/22 10:03:29 <ESL: Content-Type: auth/request
2019/02/22 10:03:29 <ESL:
2019/02/22 10:03:29  ======= Attempting log in
2019/02/22 10:03:29 >ESL auth ClueCon

2019/02/22 10:03:29 <ESL: Content-Type: command/reply
2019/02/22 10:03:29 <ESL: Reply-Text: +OK accepted
2019/02/22 10:03:29 <ESL:
2019/02/22 10:03:29  ======= Login OK
2019/02/22 10:03:29  ======= Showing queues in mod_callcenter
2019/02/22 10:03:29 >ESL api callcenter_config queue list

2019/02/22 10:03:29 <ESL: Content-Type: api/response
2019/02/22 10:03:29 <ESL: Content-Length: 543
2019/02/22 10:03:29 <ESL:
2019/02/22 10:03:29 <ESL:
name|strategy|moh_sound|time_base_score|tier_rules_apply|tier_rule_wait_second|tier_ru
le_wait_multiply_level|tier_rule_no_agent_no_wait|discard_abandoned_after|abandoned_re
sume_allowed|max_wait_time|max_wait_time_with_no_agent|max_wait_time_with_no_agent_tim
e_reached|record_template|calls_answered|calls_abandoned|ring_progressively_delay|skip
_agents_with_external_calls|agent_no_answer_status
2019/02/22 10:03:29 <ESL: 75082016-6394-4738-b896-b9121c060612|longest-idle-
agent|local_stream://default|system|false|30|true|true|900|false|0|90|5||1|10|0|true|O
n Break
2019/02/22 10:03:29 <ESL: +OK
2019/02/22 10:03:29  ======= Showing agents defined in mod_callcenter
2019/02/22 10:03:29 >ESL api callcenter_config agent list

2019/02/22 10:03:29 <ESL: Content-Type: api/response
2019/02/22 10:03:29 <ESL: Content-Length: 464
```

```
2019/02/22 10:03:29 <ESL:
2019/02/22 10:03:29 <ESL:
name|system|uuid|type|contact|status|state|max_no_answer|wrap_up_time|reject_delay_tim
e|busy_delay_time|no_answer_delay_time|last_bridge_start|last_bridge_end|last_offered_
call|last_status_change|no_answer_count|calls_answered|talk_time|ready_time|external_c
alls_count
2019/02/22 10:03:29 <ESL: 739a4112-d755-4977-bf2b-
d2b9037babd0|single_box||callback|{call_timeout=15}user/200@10.10.5.182|Available|Wait
ing|0|10|90|90|30|1550762836|1550762841|1550763105|1550762723|0|1|5|1550763195|0
2019/02/22 10:03:29 <ESL: +OK
2019/02/22 10:03:29  ======= Logging off
2019/02/22 10:03:29 >ESL exit

2019/02/22 10:03:29 <ESL: Content-Type: command/reply
2019/02/22 10:03:29 <ESL: Reply-Text: +OK bye
2019/02/22 10:03:29 <ESL:
```

Diagnostics: Test data download from Enswitch

This tool will check that you can download data from your Enswitch PBX, as one of its tenants, as QueueMetrics Live would do.

Example run:

```
uniloader test enswitch
         --uri https://www.my.pbx
         --login qm@cli
         --pwd ab1234
         --single-tenant 1234
```

Note that you always need to pass the numeric Customer ID in the `single-tenant` parameter.

Will print:

```
2021/03/31 19:08:07 Testing Enswitch connection to 'esw/https://www.my.pbx qm@cli
ab1234 123'
2021/03/31 19:08:09 Downloaded 6 rows in 1748 ms
2021/03/31 19:08:09 Showing last 10 records:
2021/03/31 19:08:09 # 1: {1617156858|1617156848.656|q1|NONE|ENTERQUEUE||0447xxx}
2021/03/31 19:08:09 # 2: {1617156859|1617156848.656|q1|NONE|ABANDON|1|1|1}
2021/03/31 19:08:09 # 3: {1617166397|1617166387.152|q1|NONE|ENTERQUEUE||0427xxx}
2021/03/31 19:08:09 # 4: {1617166412|1617166387.152|q1|Agent/621|RINGNOANSWER|15000}
2021/03/31 19:08:09 # 5:
{1617166425|1617166387.152|q1|Agent/621|CONNECT|28|Local/621@enswitch-local/n|8000}
2021/03/31 19:08:09 # 6:
{1617166440|1617166387.152|q1|Agent/621|COMPLETEAGENT|28|15|1}
```

You can check credentials and download queue/agent configuration with the sub-command pbxinfo enswitch.

Diagnostics: Test an AudioVault server

It is possible to run a query on an AudioVault server as-if QueueMetrics would do it. See AudioVault for more details on how to enable it.

This test is useful to:

- detect connectivity issues: if you use a proxy in front of AudioVault for HTTPS, you may want to make sure that it is working correctly
- demonstrate slow search performance
- print the correct configuration to be entered on QueueMetrics to enable AudioVault

To set up an AudioVault server, our suggested operational sequence is the following:

- find a unique-of a call that you want to find from one of the recordings you have on disk
- run `uniloader av find` to make sure that the search path is set up correctly and that your call-id can be found
- run `uniloader av serve` to expose the service on a local port
- run `uniloader test audiovault` pointing to the local HTTP URL to make sure that the service is configured correctly
- set up an HTTPS proxy
- run `uniloader test audiovault` pointing to the public HTTPS URL to make sure that everything is still working with the proxy in front

To run it, just use:

```
uniloader test audiovault --public-url http://127.0.0.1:4012 --token Sup3rZ3brA -c
443652.1
```

Note that there is no need to pass a timestamp if the call-id has the format "12345678.12" like is the default for Asterisk systems.

If you are testing a multi-tenant system, make sure you add the option `for-tenant`:

```
uniloader test audiovault --token zebra
    --public-url 'https://pbx.example.com'
    -c 7ded3683-2335-41e4-915f-60de54d353e0 --timestamp 1606400744
    --for-tenant t1.example.com
```

If everything goes well, you will get the results of your inquiry:

```
Connecting to server on http://127.0.0.1:4012/search/ with  token 'Sup3rZ3brA'
Searching for UID '443652.1' around timestamp '443652'
... took 4.553417ms
========= RESULTS FOUND: 1
  #| Type|     Size|                                    File
---|-----| --------|-----------------------------------------
  1|  MP3|    66821|                      call-443652.1.mp3
```

It will also print out working QueueMetrics `configuration.properties` lines that will use the same configuration, taking into consideration the tenant when appropriate:

```
audio.server=it.loway.app.queuemetrics.callListen.listeners.JsonListener
audio.jsonlistener.url=http://127.0.0.1:4012/search/
audio.jsonlistener.method=POST
audio.jsonlistener.searchtoken=Sup3rZ3brA
audio.jsonlistener.verbose=false
audio.html5player=true
```

For a complete test, it is possible to add the flag `--download-results` that will download all results found to the local working directory, thus making sure that both search and retrieval work as expected.

Diagnostics: Direct QueueMetrics database access

It is possible to use Uniloader to directly access a QueueMetrics `queue_log` database to perform low-level operations.

- Direct DB access: Database contents
- Direct DB access: Exporting a partition
- Direct DB access: Deduplicating data
- Direct DB access: Creating private clones of reports

Direct DB access: Database contents

It is possible to print the contents of a database:

```
$ uniloader qmdb  --dburi "localhost/queuemetrics" --login queuemetrics  --pwd
javadude info

   #       Partition  # Entries          From date            To date    Time span
   1            P001   51129261   2018.09.03 07:09   2019.09.03 09:09     1y 0d 1h
```

This displays the contents of each partition.

Direct DB access: Exporting a partition

You can export a partition to a text file in `queue_log` format; this can be useful for backup purposes or to upload it again to a different instance or a different partition:

```
$ uniloader qmdb  --dburi "localhost/qm" --login qm  --pwd 123 \
                  export --partition P001 --filename mylog.txt

Exporting partition: P001 (from: 0 to: 999999999999) to 'mylog.txt' in batches of
50000 rows
Finding zones: ....................................
Source zones: 27 -> Packed zones: 5 (Efficiency: 80%)
Processing zone 1 of 5: 0% done
Processing zone 2 of 5: 20% done
Processing zone 3 of 5: 40% done
Processing zone 4 of 5: 60% done
Processing zone 5 of 5: 80% done
Success: Written 154599 lines to 'mylog.txt'
```

When this runs, the contents of the `queue_log` table are split into "zones" that have a maximum of rows as defined in the `batchsize` parameter. The larger the `batchsize`, the more memory Uniloader uses, but of course the operation is quicker. As a rule of thumb, you can expect each 1000 rows to take ~700k of memory.

Exporting and reimporting

A common case for exporting data is to re-import it somewhere else or to a different partition within the same database.

This can be easily achieved:

```
uniloader qmdb  --dburi "localhost/queuemetrics" --login qm --pwd 234 \
                                    export --partition P001 --filename mylog.txt


uniloader -s mylog.txt upload --uri "mysql:otherserver/queuemetrics" \
                                    --login qm --pass 123 --token P002
```

Direct DB access: Deduplicating data

If you happen to have duplicate data on a partition (eg. because the loader was run multiple times in parallel), you can easily detect it and, if needed, clean it up.

If the process is called with the `--write` flags, a deduplication will be performed; if not, as default, it will only inspect the partition.

Make a backup of the database before you attempt a clean-up. Data is deleted for

> good and is **not** recoverable.

To perform its task, dedupe performs a series of steps:

- The contents of the `queue_log` table is split into a number of "zones", where each zone has a maximum of `batchsize` rows (default 500k).
- Each zone is separately checked for duplicates
- Only zones that actually have duplicate data are de-duplicated

You can run it like:

```
$ uniloader qmdb  --dburi "localhost/qm" --login qm  --pwd 1234 dedupe --write

Deduplicating P001 (0-999999999999) with batches of 30000 rows - write: true
Finding zones: ..............................................................
Source zones: 51 -> Packed zones: 12 (Efficiency: 75%)
Scanned zone 1 of 12 - 8% done - Dupes found: 0
Scanned zone 2 of 12 - 16% done - Dupes found: 0
Scanned zone 3 of 12 - 25% done - Dupes found: 19338
Scanned zone 4 of 12 - 33% done - Dupes found: 28094
Scanned zone 5 of 12 - 41% done - Dupes found: 20226
Scanned zone 6 of 12 - 50% done - Dupes found: 20606
Scanned zone 7 of 12 - 58% done - Dupes found: 10438
Scanned zone 8 of 12 - 66% done - Dupes found: 20126
Scanned zone 9 of 12 - 75% done - Dupes found: 21046
Scanned zone 10 of 12 - 83% done - Dupes found: 25302
Scanned zone 11 of 12 - 91% done - Dupes found: 14790
Scanned zone 12 of 12 - 100% done - Dupes found: 27438
 -- Total duplicates found: 207404

Cleaned zone 1 of 10 - 10% - Removed: 19338
Cleaned zone 2 of 10 - 20% - Removed: 28094
Cleaned zone 3 of 10 - 30% - Removed: 20226
Cleaned zone 4 of 10 - 40% - Removed: 20606
Cleaned zone 5 of 10 - 50% - Removed: 10438
Cleaned zone 6 of 10 - 60% - Removed: 20126
Cleaned zone 7 of 10 - 70% - Removed: 21046
Cleaned zone 8 of 10 - 80% - Removed: 25302
Cleaned zone 9 of 10 - 90% - Removed: 14790
Cleaned zone 10 of 10 - 100% - Removed: 27438
 -- Total duplicates removed: 207404
```

As duplicate rows may be loaded in memory, make sure that Uniloader has enough RAM to run in (it's ~700K every 1000 records). It is possible to set e.g. `--batchsize 30000` to specify a maximum size for each zone.

As the actual deduplication is performed by the database, it will create large temporary files during this process. Make sure that thee database server has plenty of disk space available. Using a smaller batch size helps if you see the process failing.

The process can be run safely on a live system that is uploading data, though it will severely affect database performance; so we suggest running it off-hours. We also suggest running a database optimization afterwards to compact the database if a lot of deletions were performed.

After deleting data, you need to clean the caches of QueueMetrics or restart it.

Direct DB access: Creating private clones of reports

Sometimes, you want users to have private copies of "scratchpad" reports. This can be easily done from the main interface of QM, but to explain to all users how to make a private copy of a report can be burdensome.

We suggest instead that:

- You create "template" reports, and make them public but protect them with an editing key that users do not have;
- Using this command, you create a private copy of the report for each user, and set its editing key to ''.

This way, each user will have its own scratch copy of said reports, and shared changes won't trigger optilock exceptions (or just unwanted modifications).

To do this, we imagine that:

- the report you want to share exists and is named "A". Its name must be unique in the system.
- you want each user to have a private copy called "A (pvt)", so we need to add a suffix "(Pvt)"
- the users you want to give it to have their logins named named `alice` and `bob` in QM

```
uniloader qmdb --login queuemetrics --pwd javadude \
   clonereport --sourceTitle 'A' --destSuffix '(pvt)' --toUsers alice,bob --editingKey
''
```

This command will make sure that everything is in order; users and reports exist, and users do not have a report with the same name as the one that will be cloned.

To actually make changes, just add the *--forReal* parameter.

```
uniloader qmdb --login queuemetrics --pwd javadude \
   clonereport --sourceTitle 'A' --destSuffix '(pvt)' --toUsers alice,bob --editingKey
'' --forReal
```

To leave keys unchanged, just set them to "=". That is the default. Any other value will be used to overwrite the key itself, so '' means "no key".

A sucessful output may look like:

```
2023/10/13 16:16:30 Will attempt cloning report 'A' for users [alice bob]
2023/10/13 16:16:30 Report #1 'A' for user demoadmin (Administrator) has 14 screens
and 101 items (VK '' EK 'XXX')
2023/10/13 16:16:30 Starting to copy reports
2023/10/13 16:16:30  Cloning for alice...
2023/10/13 16:16:30   Done
2023/10/13 16:16:30   Report #4734 'A (pvt)' for user alice (Alice Cooper) has 14
screens and 101 items (VK '' EK '')
2023/10/13 16:16:30  Cloning for bob...
2023/10/13 16:16:30   Done
2023/10/13 16:16:30   Report #4850 'A (pvt)' for user bob (Bob Moog) has 14 screens
and 101 items (VK '' EK '')
2023/10/13 16:16:30 Reports cloned
```

The values VK are the visibility key for that report and EK the editing key, respectively.

Before you run such transactions, you may want to make a backup of the QM database.

Diagnostics: Regexp tester

Writing regular expressions to be used in Uniloader is sometimes unwieldy; the syntax is arcane and it is not always clear which expressions will match.

Uniloader lets you test a Golang regular expression from the command line. It will make sure that the syntax is correct and will check it against a number of cases you supply, printing out their average execution time so you can optimize it.

```
$ uniloader test regexp -?


NAME:
   uniloader test regexp - Tests a Go Regular Expression (RegExp)

USAGE:
   uniloader test regexp [command options] [arguments...]

DESCRIPTION:
   This command test a Regular Expresion.

It makes sure that the RegExp is correct, shows how
it behaves against a number of inputs you provide and
tries measuring its performance.

A complete reference guide on Go RegExps is available
at https://pkg.go.dev/regexp/syntax
```

```
OPTIONS:
   --regexp value, -r value  The regular expression you want to test (default: "^.*")
```

In order to use it, you can call it like:

```
$ uniloader test regexp -r '^(?i)sip/|zap/' SIP/1234 PJSIP/4576 SIP/trunk-1234
```

And it will print out a table where you can see which items are a positive match (the ones with someting under the column "Matches?") versus the ones that are skipped.

```
Testing Regexp '(?i)^sip/|^zap/' - compilation took 24.542µs

| # |     TARGET     | MATCHES? |  TIME  |
+---+----------------+----------+--------+
| 1 | SIP/1234       | [SIP/]   | 15.8µs |
| 2 | PJSIP/4576     | -        | 10.1µs |
| 3 | SIP/trunk-1234 | [SIP/]   | 11.8µs |

Average regexp execution time (across all cases): 12.6µs
```

The elapsed time is not meant as an exact duration (though it is the average of a number of repeats) but as a guide to optimize your regexp - for example you can see significant changes by adding or removing the `^` that is used to anchor the regexp to the beginning of a line.

Some examples you can start from:

- `(?i)^sip/|^zap/`: matches anything that starts with `sip/` or `zap/` in a case-insensitive way
- `^(?i)pjsip/[34]..($|-)`: matches any string like `pjsip/3XX` or `pjsip/4XX`, optionally followed by a dash or just ending there. So it won't match `pjsip/523` or `pjsip/3120`, but will match `pjsip/321-1234`
- `^.{4,6}$` matches any code made of 4 to 6 characters, so it would match "1234" or "123456" but not "123" or "12345678"

A complete reference guide on Go RegExps is available at https://pkg.go.dev/regexp/syntax

To improve matching efficiency and reduce CPU usage, whenever posibile you should "anchor" the regexp to the beginning of the line by using the caret symbol. Regular expressions are very powerful, but, as you know, with great power comes great responsibility and higher CPU load.

Diagnostics: Transcriptions

Lists the drivers and task tags compiled into this Uniloader build:

```
$ uniloader test transcribe
```

```
Known drivers:
+---------------+----------------------+--------+
|       ID      |      DESCRIPTION     |  TAGS  |
+---------------+----------------------+--------+
| openai-tts    | OpenAI TTS           | TD, TR |
| openai-api-tts | OpenAI-compatible API | TD, TR |
| deepgram-tts  | Deepgram TTS         | TD, TR |
| krokoai-tts   | Kroko-AI TTS         | TR     |
+---------------+----------------------+--------+

Known task tags:
+----+-----------------------------+
| ID |         DESCRIPTION         |
+----+-----------------------------+
| TD | Transcribe (STT) and diarize |
| TR | Transcribe only (STT)       |
+----+-----------------------------+
```

The **TAGS** column lists the task tags each driver can serve. A driver that cannot label speakers offers `TR` only: `krokoai-tts`, for instance, transcribes but never diarizes, so it is not a candidate for `TD` work.

This command takes no flags and makes no network calls. Use it to verify driver and task IDs when setting up your QueueMetrics instance.

Diagnostics: WebSocket

Checks that a WebSocket server is reachable and answering. Uniloader opens the connection, sends a ping, waits for the pong and prints how long the round trip took.

```
$ uniloader test ws -?

NAME:
   uniloader test ws - Tests a WebSocket connection

USAGE:
   uniloader test ws [command options] [arguments...]

DESCRIPTION:
   This command opens a WebSocket connection, sends a ping,
waits for the pong and prints the round-trip time.

When a login is given, it authenticates with HTTP basic auth;
when it is empty, no authentication is attempted.

OPTIONS:
   --url value       The WebSocket to connect to, e.g. 'ws://host:8080' or
'wss://host'
```

```
   --login value     The user name for HTTP basic auth. If empty, no authentication is
used
   --password value  The password for HTTP basic auth [$WS_PASSWORD]
```

A successful run looks like:

```
$ uniloader test ws --url ws://127.0.0.1:8080 --login wombat --password dials

Testing WebSocket connection to 'ws://127.0.0.1:8080' - user 'wombat' password '*****'
Pong received - round trip took 173.833µs
```

The password can also be supplied through the `WS_PASSWORD` environment variable, to keep it out of your shell history. It is never printed back in clear.

When `--login` is empty, no credentials are sent at all. Otherwise Uniloader authenticates with HTTP basic auth, which travels in the opening handshake. As basic auth is only base64-encoded, prefer `wss://` whenever you send credentials, so that they are protected by TLS.

The command exits with status 0 when the pong arrives, and 1 on any error, so you can use it in monitoring scripts. Errors are printed rather than raised as stack traces:

```
$ uniloader test ws --url http://host:8080
*** 'http://host:8080' is an http URL: use 'ws://' instead

$ uniloader test ws --url ws://host:8080
*** cannot connect to 'ws://host:8080': dial tcp 10.0.0.1:8080: connect: connection
refused

$ uniloader test ws --url ws://host:8080
*** cannot connect to 'ws://host:8080': websocket: bad handshake (server said: 401
Unauthorized)
```

Only the `ws://` and `wss://` schemes are accepted; any other URL is rejected before connecting. A `401 Unauthorized` on a server that requires credentials usually means a wrong or missing `--login` / `--password`.

This command is a quick way to check a Kroko-AI transcription server, that Uniloader reaches over a WebSocket - see Transcribing calls.

Diagnostics: 3CX

Checks that Uniloader can talk to a 3CX PBX before you start `uniloader 3cx`. It takes the same connection flags, and connects exactly the way `uniloader 3cx` does:

1. it gets an access token for the API application;
2. it reads the version of the PBX, together with a summary of its licence;

3. it prints the grants in the access token: who it belongs to, and which roles it has;
4. it lists the DNs - extensions, queues and the API application itself - that the Call Control API lets it see;
5. it lists the queues and the ring groups, with their members;
6. it lists the agents of the queues, with the state each one is in and the queues it belongs to;
7. it opens the Call Control WebSocket, where call events arrive, sends a ping, waits for the pong, prints how long the round trip took and closes the connection.

```
$ uniloader test 3cx --help

NAME:
   uniloader test 3cx - Tests the connection to a 3CX PBX

USAGE:
   uniloader test 3cx [command options] [arguments...]

DESCRIPTION:
   This command connects to a 3CX PBX the way 'uniloader 3cx' does:
it gets an access token, prints the version of the PBX, the grants
in the token, the DNs it can see, the queues and ring groups and the
agents of the queues, then opens the Call Control WebSocket, sends
a ping and prints the round-trip time.

OPTIONS:
   --url value            Base URL of the 3CX PBX, e.g. https://pbx.example.com
   --client-id value      Client ID of the 3CX API application
   --client-secret value  Client secret of the 3CX API application [$PBX3CX_PWD]
```

A successful run looks like:

```
$ export PBX3CX_PWD="the-client-secret"
$ uniloader test 3cx --url https://pbx.example.com --client-id uniloader

2026/09/22 18:30:36 Testing 3CX connection to 'https://pbx.example.com'
('uniloader'/'******************')
Access token:
  ok - expires at 2026-09-22T19:30:36+02:00

PBX version:
  System:      pbx.example.com
  Version:     20.0.9.995
  Reseller:    Example Reseller Ltd
  Extensions:  8 (4 registered)
  Trunks:      4 (4 registered)
  License:     3CXPSPROFENTSPLA - activated, active, expires 2035-01-01
  Maintenance: support, expires 2035-01-01
  Max calls:   16 simultaneous
```

```
Grants in the access token:
  Client id:   uniloader
  MaxRole:     system_owners
  Roles:       system_owners

DNs visible on the Call Control API:
|    DN     |    TYPE     |                              DEVICES
|
+-----------+-------------
+-------------------------------------------------------------------+
| 103       | Wextension  | 103 (3CX Mobile Client) @
sip:103@127.0.0.1:5483;rinstance=cbf |
| 104       | Wextension  | 104 (Yealink T54W) @ sip:104@10.0.0.9:5060
|
| 800       | Wqueue      | 800 (3CX Queue Extension) @ sip:800@127.0.0.1:5483
|
| uniloader | Wroutepoint | uniloader (3CX  Extension) @ sip:uniloader@127.0.0.1:5483
|

Queues and ring groups:
| NUMBER |    TYPE    |    NAME    |             MEMBERS             |
+--------+------------+------------+---------------------------------+
|    800 | Queue      | Sales      | Agent/103, Agent/104, Agent/105 |
|    850 | Ring group | Front desk | Agent/103, Agent/104            |

Agents on the queues:
|   AGENT   |        STATE         | QUEUES |
+-----------+----------------------+--------+
| Agent/103 | Paused (Outofoffice) | 800    |
| Agent/104 | Logged on            | 800    |
| Agent/105 | Logged off           | 800    |

WebSocket wss://pbx.example.com/callcontrol/ws
  pong received - round trip took 23.412ms

3CX connection OK
```

The client secret can also be supplied through the `PBX3CX_PWD` environment variable, to keep it out of your shell history. It is never printed back in clear, and neither is the licence key of the PBX.

The sections of the report are:

- **Access token**: Uniloader logged in with the client ID and secret, and got a token valid until the time shown. If this fails, nothing else can be tested, and the command stops here.
- **PBX version**: the name of the PBX, its version, and a summary of its extensions, trunks, licence and maintenance. `NOT activated` or `NOT active` on the licence line means that the PBX licence needs attention. An API application with a restricted role may not be allowed to read this information: then the section says `not available`, but this is not counted as a failure, as `uniloader 3cx` does not need it.

- **Grants in the access token**: the roles the PBX actually granted to the API application. These are what counts, not what the admin panel shows: if queues or agents cannot be read, check them here.
- **DNs visible on the Call Control API**: the extensions and the queues this API application is allowed to see, each with the devices registered on it, and the application itself, which is a DN of type `Wroutepoint`. **This list is what Uniloader will be able to track**: calls on an extension or a queue that is not here produce no events at all, and a queue missing from it is never written to the queue_log. If something you expect to track is missing, add it to the extensions selected for the API application in the 3CX admin panel.
- **Queues and ring groups**: every queue and every ring group of the PBX, with its name and its members, as pbxinfo 3cx creates them in QueueMetrics. A ring group is tracked as a queue: see 3CX support. If either cannot be read, the command stops here: without them, `uniloader 3cx` cannot run at all.
- **Agents on the queues**: every agent that is a member of at least one queue, with the queues it belongs to and its state, decided exactly as `uniloader 3cx` writes it to the queue_log:
  - `Logged on`: the agent is logged in to the queues, with the profile `Available` (or `Default`, the PBX's own);
  - `Paused (<reason>)`: the agent is logged in, with any other profile; the reason is the one the `PAUSEALL` row will carry, i.e. the profile name without spaces (`Out of office` becomes `Outofoffice`);
  - `Logged off`: the agent is not logged in to the queues.

  On 3CX the login and the profile belong to the user, not to the queue, so an agent has the same state in all of its queues and appears on one row only. If the queues and the agents cannot be read, this test fails: without them, `uniloader 3cx` cannot track agents at all.
- **WebSocket**: call events arrive on this connection. If it cannot be opened while the other tests pass, check that no firewall or proxy between Uniloader and the PBX blocks WebSockets.

The command exits with status 0 when every test passes, and 1 otherwise, so you can use it in monitoring scripts. When a test fails, the others are still run - except for the access token and the queues and ring groups, without which there is nothing left to test - and the last line lists the ones that failed:

```
$ uniloader test 3cx --url https://pbx.example.com --client-id uniloader
...
WebSocket wss://pbx.example.com/callcontrol/ws
  *** cannot connect: websocket: bad handshake (HTTP 403 Forbidden)

*** failed: websocket
```

Without an access token, instead, the command stops at once:

```
$ uniloader test 3cx --url https://pbx.example.com --client-id uniloader --client
-secret wrong
```

```
2026/09/22 18:30:36 Testing 3CX connection to 'https://pbx.example.com'
('uniloader'/'*****')
Access token:

*** cannot log in to 3CX: POST /connect/token: HTTP 400: {"error":"invalid_client"}
```

For how to create the API application and which permissions it needs, see 3CX support.

PBX Information

Uniloader is able to read and display PBX information; this is useful both for debugging/inspecting a remote system, and pushing a ready-made configuration to a QueueMetrics instance, so that you can automatically keep it in sync with the underlying PBX.

FreePBX

If you connect to the MySQL database of a FreePBX instance with:

```
uniloader pbxinfo freepbx --dburi 10.10.5.27/asterisk --login myuser --pwd mypass
```

You get a screen print-out of the status of the system, with an added queue "00 All".

```
=  Tenant # 1:

-- Queues found: 6
   #            Code            Name                        Ext.Ref.
   1           00all          00 All
   2             300         Support
   3             301           Sales
   4             307      Some queue
   5             400          Recall
   6             401   Queue timeout

-- Agents found: 15

   #            Code            Name                        Ext.Ref.
   1       Agent/200             200
   2       Agent/201             201
   3       Agent/202             Joe
   4       Agent/203            Mick
   5       Agent/204             Gru
   6       Agent/205             205
   7       Agent/210             210
   8       Agent/211             211
   9       Agent/220             220
  10       Agent/221             221
  11       Agent/240             240
  12       Agent/241             241
  13       Agent/242             242
  14       Agent/250             250
  15       Agent/251             251
```

This configuration can be uploaded to QueueMetrics in one go - see Uploading configuration to a QueueMetrics instance

Issabel

If you connect to the MySQL database of an Issabel PBX with:

```
uniloader pbxinfo issabel --dburi 10.10.5.27/asterisk --login myuser --pwd mypass
```

You get a screen print-out of the status of the system, with an added queue "00 All".

```
=  Tenant # 1:

-- Queues found: 2
| # | CODE  |   NAME    | EXT REF  |
+---+-------+-----------+----------+
| 1 | 00all | 00 All    |          |
| 2 |   301 | Queue 301 |          |

-- Agents found: 1

| # |   CODE    | NAME | EXT REF  |
+---+-----------+------+----------+
| 1 | Agent/400 |  400 |          |
```

This configuration can be uploaded to QueueMetrics in one go - see Uploading configuration to a QueueMetrics instance.

As agents do not have a specific role, in order to upload agents we create an agent for each SIP extension.

FusionPBX

It is possible to get a list of agents and queues configured in *mod_callcenter* on a FusionPBX instance, by accessing the database directly.

By running:

```
uniloader pbxinfo fusionpbx --ps-uri 10.10.5.182/fusionpbx
```

You get a list of all agents and queues, divided by tenant:

```
=  Tenant # 1: mycustomer1.mypbx

-- Queues found: 1
   #          Code           Name                         Ext.Ref.
   1          300            q300    75082016-6394-4738-b896-b9121c060612

-- Agents found: 1
```

```
   #    Code                 Name                        Ext.Ref.
   1    Agent/200             200    739a4112-d755-4977-bf2b-d2b9037babd0
```

The complete syntax is:

```
# uniloader pbxinfo fusionpbx -?

NAME:
   uniloader pbxinfo fusionpbx - Gets information on a FusionPBX instance

USAGE:
   uniloader pbxinfo fusionpbx [command options] [arguments...]

DESCRIPTION:
   Downloads FusionPBX settings.

It connects to the Postgres database and downloads
agents and queues as currently defined.

OPTIONS:
   --ps-uri value         A Postgres database to connect to (default:
"localhost/fusionpbx")
   --ps-login value       A database user (default: "fusionpbx")
   --ps-pwd value         A database password [$FUSIONPWD]
   --single-tenant value  If you specify a domain (tenant), only that tenant will be
read.
   --as-tenant value      In single-tenant mode, the ID for this tenant. Can be set as
''.
```

The flag `single-tenant` is useful when you have a multi-tenant system that feeds different QueueMetrics system, and you want the configuration of only one tenant to be sent to one specific system for autoconfiguration (see below). It can be useful to use it together with the flag `--as-tenant ''` so that when pushing a single tenant's data to a QueueMetrics instance, the tenant's code is not sent and QueueMetrics behaves as-if it was a totally separate system.

Uploading configuration to a QueueMetrics instance

It is possible to upload the configuration just read to a QueueMetrics instance in one go.

If you enable the user `robot`in QueueMetrics (that always exists, but is disabled by default), you can run:

```
uniloader pbxinfo --mode syncqm \
                  --uri http://127.0.0.1:8080/queuemetrics --login robot --pass robot
--with-password 123 \
          freepbx --dburi 10.10.5.27/asterisk --login  --pwd pippo
```

Will print out:

```
2019/04/02 09:16:53 Connecting to QueueMetrics at http://127.0.0.1:8080/queuemetrics
(user: robot pass: *****)
2019/04/02 09:16:53 Transaction Results: okay - Entries: 35 created / 0 deleted / 1
updated
```

And send the current configuration to QueueMetrics.

Useful flags are:

- `--with-password`: users are created with a default password you specify. If not, they will be created with a random password, so in order to use them the QueueMetrics administrator will have to change their password manually
- `--all-queues`: if set to 1, a queue called `00 All`, of which all agents are known members, will be created. This is the default. Set to 0 to disable.

Integrics Enswitch

You can connect to an Integrics Enswitch system (v3.14+) through its HTTP APIs, in order to:

- verify that your credentials are valid
- generate the correct KNumber to create QueueMetrics-Live instance over its APIs
- generate or inspectt an automatic configuration for a tenant

You can run with or without a Customer ID; if you run without, it tries determining it on its own.

```
uniloader pbxinfo enswitch
          --uri https://www.my.pbx --login qm@cli --pwd ab1234
```

or

```
uniloader pbxinfo enswitch
          --uri https://www.my.pbx --login qm@cli --pwd ab1234
          --single-tenant 123
```

You get a screen print-out of the status of the system, with an added queue "00 All". The Tenant field is the numeric value of your Enswitch customer-id.

```
2021/03/31 18:19:40 ** API configuration: esw/https://www.my.pbx qm ab1234 123 **

=  Tenant # 1: 123

-- Queues found: 2
| # |      CODE       |      NAME       | EXT REF  |
```

```
+---+----------------+----------------+----------+
| 1 | 00all          | 00 All         |          |
| 2 | myqueue        | My Super Queue |          |

-- Agents found: 1

| #  |     CODE      |              NAME              | EXT REF  |
+----+---------------+--------------------------------+----------+
|  1 | Agent/18408   | John Doe (18408)               |          |
```

This configuration can be uploaded to QueueMetrics in one go - see Uploading configuration to a QueueMetrics instance

Once you know that your credentials are correct, you can test that they are valid to download data through the sub-command test enswitch.

3CX

3CX support is experimental.

You can read queues, ring groups and agents from a 3CX PBX through its API, as uniloader 3cx does. It uses the same credentials: the client ID and the client secret of an API application - see 3CX support for how to create one.

```
uniloader pbxinfo 3cx \
          --url https://pbx.example.com --client-id uniloader --client-secret ab1234
```

The client secret can also be supplied through the `PBX3CX_PWD` environment variable, to keep it out of your shell history.

You get a screen print-out of the status of the system, with an added queue "00 All":

```
2026/09/22 19:56:22 Reading 3CX at https://pbx.example.com (client: 'uniloader'
secret: '******')

=  Tenant # 1:

-- Queues found: 10
| #  |    CODE    |         NAME         | EXT . REF . |
+----+------------+----------------------+-------------+
| 1  | 00all      | 00 All               |             |
| 2  | 800        | Sales                |             |
| 3  | 800-QCB    | Callback for Sales   |             |
| 4  | 801        | Support              |             |
| 5  | 801-QCB    | Callback for Support |             |
| 6  | 850        | Front desk (R/G)     |             |
| 7  | _IVR       | IVR                  |             |
```

```
| 8  | _DIRECT    | Direct calls         |            |
| 9  | _OUTBOUND  | Outbound calls       |            |
| 10 | _VOICEMAIL | Voicemail            |            |

-- Agents found: 3

| # |   CODE    |     NAME     | EXT . REF . |
+---+-----------+--------------+-------------+
| 1 | Agent/103 | Rino Bianchi |             |
| 2 | Agent/104 | Luca Rossi   |             |
| 3 | Agent/105 | Lenz Verdi   |             |
```

What is read from the PBX:

- every user becomes an agent, whether it is a member of a queue or not. Its code is `Agent/` plus its extension - the same code that `uniloader 3cx` writes in the queue_log - its name is the first and last name of the user, and its e-mail address is taken from the user's own, so that QueueMetrics can send reports to it;
- every queue becomes a queue, named as on the PBX, and the agents that are members of it are set as its known agents;
- every queue also gets its callback queue, e.g. `800-QCB`, described as `Callback for <name>` and with no known agents: it is where `uniloader 3cx` writes the search for an agent to serve a callback;
- every ring group becomes a queue too, with its members as known agents: `uniloader 3cx` tracks ring groups as queues, named after their number. Its description is its name followed by `(R/G)`, e.g. `Front desk (R/G)`, so that it can be told from a queue;
- when the queue `00 All` is created (`--all-queues`, on by default), the queues on which `uniloader 3cx` writes calls that are not in any 3CX queue are created as well, with no known agents: `_IVR`, `_DIRECT`, `_OUTBOUND` and `_VOICEMAIL`.

As 3CX has one tenant only, the Tenant field stays empty.

This configuration can be uploaded to QueueMetrics in one go - see Uploading configuration to a QueueMetrics instance

If the connection fails, or queues and agents come up empty, check the credentials and the permissions of the API application with the sub-command test 3cx.

Asterisk tools

Uniloader is able to perform maintenance operations on the queues of an Asterisk system.

As many tools use a regexp to identify a set of agents, you can use the Regexp tester to develop custom regular expressions easily.

Queue-Add: add an extension to some or all queues

This command lets you log on an agent (with optionally an hotdesking extension and/or a separate state interface) on a set of queues as specified by a regex.

This way Uniloader will:

- connect to Asterisk and fetch the list of availble queues
- select all those that match the given regexp
- add the member to all those queues.

Like other Asterisk subcommands, it supports a "dry run" mode where you can see what the command would do without actually doing it.

For example, running:

```
uniloader asterisk queue-add
  --queues ".+"
  --agent SIP/101
  --host 127.0.0.1 --login qm --secret 1234
  --dry-run 0
```

will add member `SIP/101` to any queue on the system.

Running instead:

```
uniloader asterisk queue-add
  --queues "3.."
  --agent agent/101
  --hotdesking  PJSIP/203
  --host 127.0.0.1 --login qm --secret 1234
  --dry-run 0
```

will create a hot-desking record specifying that `Agent/101` is working at extension `PJSIP/203`, and will add that extension to all three-character queues starting with "3".

Full invocation:

```
NAME:
```

```
   uniloader asterisk queue-add - Add an agent to one or more queues.

USAGE:
   uniloader asterisk queue-add [command options] [arguments...]

DESCRIPTION:
   Adds an agent to one or more existing queues, handling
hot-desking and presence.


OPTIONS:
   --queues value, -q value      A regexp that details which queue ids to match, e.g.
'^3..$' or '.+' for all
   --agent value, -A value       An agent ID or channel, e.g. Agent/101 or PJSIP/1234
   --hotdesking value, -H value  A channel the agent is hotdesking on, e.g.
PJSIP/1234 or Local/123@from-internal/n
   --presence value, -p value    A device to use as a StateInterface, e.g.  PJSIP/1234
   --host value                  Your Asterisk server (default: "127.0.0.1")
   --port value                  The AMI port on Asterisk (default: 5038)
   --login value                 The AMI user as defined in manager.conf
   --secret value                The AMI secret [$AMISECRET]
   --dry-run value, -d value     When 1, this is a dry run. Set to 0 to perform
changes on the PBX. (default: 1)
```

Queue-Kick: log-off all (or some) users

This command lets you log off some (or all) agents, from some (or all) queues, at a given time.

This is useful to avoid agents that "forget" to log off and remain logged in at all time, therefore impacting agent presence statistics. Their presence also impacts queues, as the queue will always think that there is someone ready to serve calls even if they are not there anymore, and will keep callers waiting uselessly e.g. at night.

Agents are unpaused before being logged off.

This is also useful to forcibly log off agents that managed to log in using impossible extensions or agent codes, in this case it can be run manually on demand or, for the hardest cases, scripted to run every few minutes to forcibly resolve such cases.

By using a regular expression for queues and/or agents, you can target a set of specific entities, e.g. all queues that start with 3, or all agents with a space in their name.

For example:

- `^3..$` will match all codes that start with a 3 followed by two other characters
- `^(301|302)$` will match queues 301 or 302 but not 303
- `^\d\d$` will match any two-digit code like "12", or "34", but not "A7"
- `^.+?/\d+$` will match any agent like "SIP/123" but not e.g. "SIP/123a4"

- `^(?i)sip/\d+$` will do a case-insensitive, with match SIP/123 or sip/123

When writing regular expressions, we suggest always add the start and end line anchors (`^` and `$`), otherwise a partial match will be enough to trigger inclusion, and to enclose the expression within your shell's single quotes.

In any case, when you run the command, it will be in `dry-run` mode; so it will print out the changes that it would do, but won't perform them. To have the command actually do changes, call it with `--dry-run 0`.

For example, if you run:

```
$ uniloader asterisk queue-kick --host 1.2.3.4 --dry-run 0
Connecting via AMI to 1.2.34:5038 - Username '' secret '****'
ACTIONS PERFORMED: 3

* Agent SIP/265
  #1   - Leave Queue 300                          OK
  #2   - Leave Queue 301                          OK
  #3   - Leave Queue 400                          OK
```

All agents on all queues will be logged off, and it will print a recap of the actions that were performed.

Full invocation:

```
NAME:
   uniloader asterisk queue-kick - Removes agents from queues

USAGE:
   uniloader asterisk queue-kick [command options] [arguments...]

DESCRIPTION:
   Removes all agents matching the agent/queue filters
specified as regular expressions on the command line.

This is meant to be run  as a cron job at the end of the work day.


OPTIONS:
   --queues, -q   A regexp that details which queue ids to match, e.g. '^3..$'
   --agents, -a   A regexp that details which agents to match, e.g. '^SIP/\d+$' or
'^.+?/523$'
   --host "127.0.0.1"   Your Asterisk server
   --port "5038"  The AMI port on Asterisk
   --login     The AMI user as defined in manager.conf
   --secret       The AMI secret [$AMISECRET]
   --dry-run, -d "1" When 1, this is a dry run. Set to 0 to perform changes on the
```

```
PBX.
```

Queue-Replicate: replicate presence events

Queue-Replicate is an extension of Queue-Kick, that is, it will log agents off from a queue and will then log them back in, unser the same name and state interface, and paused if they are.

This is useful because sometimes you just want to replicate the current agent state soon after midnight, so that the Real-Time page of QM appears with the correct agent presence even when agents had no call yet for the day. In this case you create a cron-job like:

```
# At one minute past midnight, replicate Asterisk presence
1      0       *       *       *       uniloader asterisk queue-replicate --dry-run 0
```

While this is not something we suggest (if an agent is always logged on then their presence information is meaningless), sometimes smaller sites do not really care about agent productivity and want agents to always remain logged on.

This does not work correctly if your QueueMetrics runs in hot-desking mode, as hot-desking events are not correctly replicated.

Full invocation:

```
NAME:
   uniloader asterisk queue-replicate - Replicates agent presence events

USAGE:
   uniloader asterisk queue-replicate [command options] [arguments...]

DESCRIPTION:
   Removes agents from queues as specified by the agent/queue
filters and then adds them in again, so that agent presence events
are replicated on the queue_log.

This is meant to be run as a cron job soon after the midnight, so that
agents that do not log in every day still appear on  the real-time page.

DO NOT USE if you are running in hotdesking mode, as hotdesking agents
will not be replicated correctly.

OPTIONS:
   --queues, -q   A regexp that details which queue ids to match, e.g. '^3..$'
   --agents, -a   A regexp that details which agents to match, e.g. '^SIP/\d+$' or
'^.+?/523$'
   --host "127.0.0.1"   Your Asterisk server
   --port "5038"  The AMI port on Asterisk
   --login     The AMI user as defined in manager.conf
```

```
   --secret       The AMI secret [$AMISECRET]
   --dry-run, -d "1" When 1, this is a dry run. Set to 0 to perform changes on the
PBX.
```

User Information

Uniloader is able to edit users for Loway products. This is handy when scripting, as you can create users and classes, lock and unlock them, change passwords and whatever else is needed. In order to achieve this, Uniloader must be able to connect directly to the database (so the product does not need to be running).

When editing, changes are **idempotent**, this means that you declare the desired status and Uniloader will update the system as to reach it. If something is already in the correct state, it will not be updated. This is very handy when scripting, as you do not need to check that something needs to be done before doing it - e.g., if you add a user twice in a row, it will be only added once.

Editing users

To create a user (or assert that they exist), you might run:

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    add-user --username loway --classname ADMIN --fullname "Loway Suisse"
```

This will make sure that a user called "Loway" is present and belongs to class "ADMIN". If the user is created from scratch, it will be locked by default (they exist, but cannot log-in) and will have an unusable password. In order to use them, you will need to unlock and set a proper password.

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    add-user --username loway --classname ADMIN --email "me@home"
```

This will set the e-mail for user *loway*.

The following options are available, and can be mixed and matched as necessary:

- *username*: the user's login (mandatory). It can also be a string with multiple log-ins, separated by space.
- *classname*: the class this user belongs to. Mandatory if a user needs to be created.
- *keyring*: A set of required keys, separated by spaces. Keys can be explicitly "turned off" by prepending them with a minus sign. Keys will be added or removed on the basis of the current keyring.
- *fullname*: A textual description
- *email*: The e-mail address
- *locked*: User is locked: NO or YES - defaults to YES: user exist but cannot login. To keep unchanged, set it to blank with `--locked ''`. Newly created users are locked by default.
- *new-password*: The new password to set. This password is encrypted and cannot be recovered if lost.
- *signed-as*: The login of a user that will *sign* any updates

- *expires-in*: In how many days this user becomes available for automated expiry. Set -1 for no expiry, or 0 for immediate expiry.
- *must-reset-password*: If set to YES, the user is blocked so that its password must be reset. See section Password reset below.
- *pwd-reset-link-duration*: The number of days the password reset token will be valid for (including the present day)

Searching for users

To search for all users with or without a specific key in their keyring, you can run:

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    find-users --by-key -AGENT
```

Will find all users that do not hold the key `AGENT`.

This makes it possible to edit them directly, like in:

```
VS=$(uniloader user find-users --by-key -AGENT) &&
      uniloader user add-user --username "${VS}" --new-password 1234
```

That will change the password to "1234" for all users that are not agents.

If you need to find all users on the system, you can use the syntax `--by-key -`.

Editing classes

You can easily edit classes:

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    add-class --classname CLOUDOPS --fullname "System Operators"
              --keyring "USER USRADMIN USR_AGENT"
```

Will create a new class called CLOUDOPS with an initial keyring of `USER`, `USRADMIN` and `USR_AGENT`.

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    add-class --classname CLOUDOPS --keyring "-USR_AGENT USR_ADD"
```

Removes the key `USR_AGENT` from class CLOUDOPS (if present) and adds the key `USR_ADD` (if not present).

The following options are available, and can be mixed and matched as necessary:

- *classname*: the class to create. Mandatory. It can also be a string with multiple classes, separated

by space.

- *keyring*: A set of required keys, separated by spaces. Keys can be explicitly "turned off" by prepending them with a minus sign. Keys will be added or removed on the basis of the current keyring.
- *fullname*: A textual description
- *signed-as*: The login of a user that will *sign* any updates

Searching for classes

To search for all classes with or without a specific key in their keyring, you can run:

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    find-classes --by-key AGENT
```

Will find all classes that contain the key `AGENT`.

This makes it possible to edit them directly, like in:

```
VS=$(uniloader user find-classes --by-key -AGENT) &&
      uniloader user add-class --classname "${VS}" --keyring "ABC"
```

That will add the key `ABC` to all classes that do not contain `AGENT`.

If you need to find all classes on the system, you can use the syntax `--by-key -`.

Expiring users

If you run:

```
uniloader user --dburi 127.0.0.1/queuemetrics --login queuemetrics --pwd javadude
    expire --sign-as demoadmin
```

Any users that have an expiry date set that is in the past will be locked. The password will not be changed.

Password reset

It is possible to "lock" a QueueMetrics 21.04.11+ user so that they must access QueueMetrics through a special link to have their password reset. This way the administrator never knows about the actual password used by the user.

Links have a definite validity (usually 5 days, including the current one) and when they expire, you need to repeat the process again to generate a new one.

To perform this, you would first lock the user by issuing:

```
uniloader user --login queuemetrics --pwd javadude user
       -u agent/123 --email agent123@home.my  --locked NO
       --must-reset-password YES --pwd-reset-link-duration 3
```

Note how in this example we contextually set their email address and make sure they are not system-locked.

After running, Uniloader returns an access code:

```
2021/12/06 17:16:07 [Agent/123]:  Currently in class AGENTS
2021/12/06 17:16:07 [Agent/123]:  Class: AGENTS - Custom keys: '' - Name: 'Agente 123'
- Email: 'agent123@home.my' - Enabled? true
2021/12/06 17:16:07 [Agent/123]:  User must reset password - Token:
'KC13ADF0CKSHQ0DU2DSVY7EUZ819QT-20211208'
```

At this point the user cannot log-on anymore. See how the token contains the expiry date after the dash, so you can easily tell if a link is expired or not.

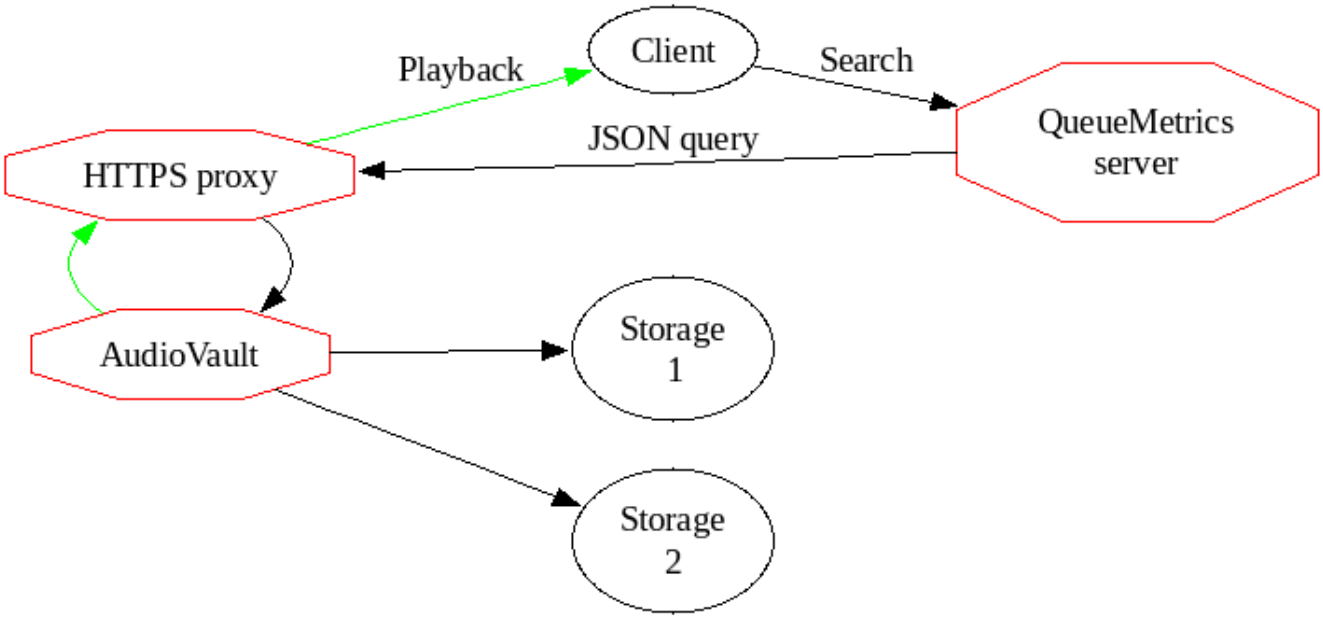
Then you would craft a special URL, like:

```
http://my.qm.server/queuemetrics/qm_password_reset.do?AUTH_user=agent/123&AUTH_token=K
C13ADF0CKSHQ0DU2DSVY7EUZ819QT-20211208
```

That allows its owner to set a new password for Agent/123, and is valid for three days from now. The password is stored in an encrypted format and therefore cannot be recovered if lost.

AudioVault

AudioVault is a remote storage for media files (usually, audio recordings) that allows a remote QueueMetrics find them and play them back. This was designed for QueueMetrics Live systems, as they have no access to the local file system that the PBX is saving files on, but you can use it on on-prem systems just as well.



AudioVault runs as a local HTTP server, and is easily exposed through a front-end proxy to the internet. QueueMetrics will call a webservice on AudioVault in order to find whether there are any recordings, and will play them back as needed.

Media files found are returned by exposing a public URLs that can be downloaded by the client's browsers; to reduce related risks, they are encoded with an **anti-tamper** mechanism so that a third party cannot retrieve arbitrary files on arbitrary paths. They also have an **expiry date** embedded that ensures a link is only valid for a short period after it is generated. Also, on each run a specific random **link secret** is generated, so that only links generated within the current execution are valid.

AudioVault has the concept of **storage drivers**, each of which is able to search using a different strategy; multiple drivers are queried in sequence, from first to last, until a match is found.

For example, if you set your drivers up as `file:/var/media,file:/mnt/nas/%YY-%MM`, AudioVault will first attempt to find your media under a local folder, and then, if nothing found, will attempt the NAS.

In order to run searches, a QueueMetrics instance must hold a security **token**, and that is checked each time a search is performed.

A number of countries have enacted strict legal regulations in terms of how and why call recordings are taken, stored and distributed; and may have harsh penalties for violations. Before you start working with call recordings, make sure you understand your local laws and regulations and their implications.

The FileSearch driver

At the moment there is only one kind of driver, and its prefix is `file:`.

This driver expects a root directory to start searching from. The root directory may contain placeholders that are expanded to some information about the call, as in the list below:

- **%YY** is the 4-digit year when the call was made
- **%MM** is the 2-digit month when the call was made
- **%ME** is the three-letter English short name for a month (e.g. *Jan* for January)
- **%DD** is the 2-digit day of month when the call was made
- **%SE** in a clustered environment, the server name (all lower case)
- **%QU** is the queue name (all lower case)
- **%TE** is the current tenant

As directories are rescanned on each search, it is important that for efficiency that you split your files in different folders based on when they were recorded. Scanning a month's worth of recordings will be 10x as fast as scanning one year's!

Serve: runs the AudioVault service

You can easily serve one or more local folders by running AudioVault like:

```
uniloader av serve \
   --path file:/mnt/audio \
   --token Sup3rZ3brA  \
   --public-url https://audiovault.my.server \
   --bind-to :4000
```

This command lets QueueMetrics search in the local path `/mnt/audio`, by running an HTTP server on port 4000 that will be available externally as `https://audiovault.my.server` and will protect it with a shared security token `Sup3rZ3brA`.

To allow a different public URL, you need an HTTPS proxy to handle the request at that address and forward it to AudioVault.

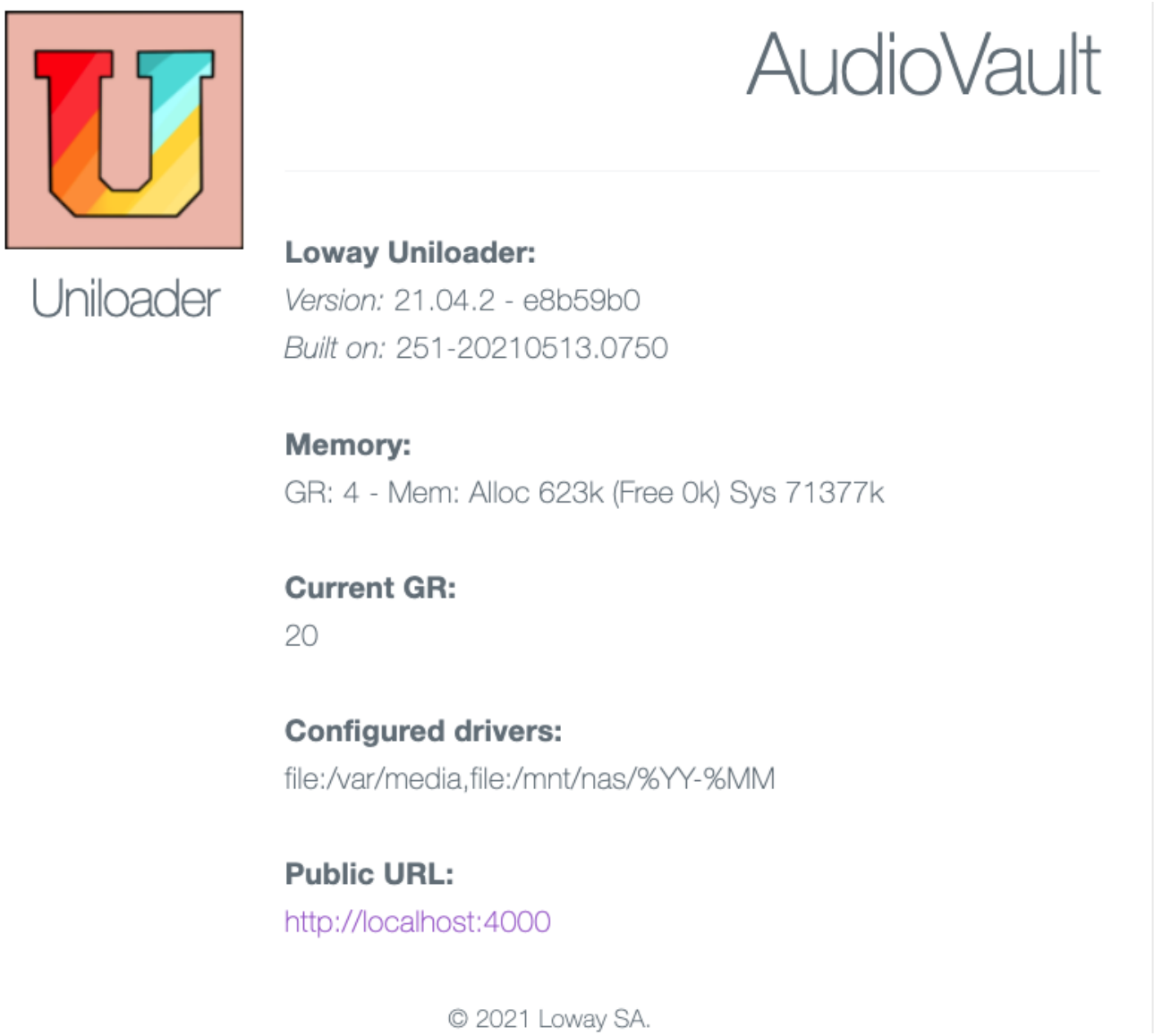
The full list of options are:

```
OPTIONS:
   --bind-to value          The address and port that AudioVault will listen on
(default: ":4000")
   --public-url value       The public URL that is proxied through to AudioVault
   --token value            Your own security token, as set on QueueMetrics. [$TOKEN]
   --display-info           Whether to display system information on the base address
   --path value             One or more search driver(s), separated by ','
```

```
   --link-duration value    The validity of links returned, in seconds. 0: Valid
forever (default: 3600)
   --link-secret value      A secret key used to verify links. If blank, set randomly.
[$LINKSECRET]
   --tenants value          The path of a JSON file describing allowed tenants
   --certificate value      A file containing the full TLS cert chain (eg.
fullchain.pem)
   --certificate-key value  A file containing the private TLS cert key (eg.
privkey.pem)
   --pid value              The PID file to write. If already present, won't start.
```

The default link duration is one hour and the default link secret is randomly assigned every time AudioVault starts, but you are free to change these settings as you best see fit.

Once AudioVault is up, you can see a visual confirmation by accessing its main URL:



The example above is generated with `--display-info` set; usually, sensitive information/configuration is not disclosed.

Handling multi-tenancy

If you have a shared repository of recordings that is supposed to be accessed by different tenants, each of which runs its own QueueMetrics instance, you can have AudioVault check security on each tenant so that:

- they don't share the same secret, and
- each tenant has its own "root folder" for recordings

To do this, you define tenants in a JSON file that is passed to Uniloader through the `--tenants` parameter on start, like:

```
[
     {
          "tenant": "t1.example.com",
          "secret": "superSecret",
          "note":   "A custom note for this tenant"
     },
     {
          "tenant": "t2.example.com",
          "secret": "N0b0dyKn0ws",
          "note":   ""
     }
]
```

Now, when there is a request for tenant `t1.example.com`, its secret is checked with the one on file; if it matches, the variable `%TE` is filled with the name of the tenant, so your search driver can make good use of this information. Any tenants not defined will not be able to access their recordings.

For example, by default FusionPBX stores recordings in files that have names like:

```
/var/lib/freeswitch/recordings/t1.example.com/archive/2025/Apr/01/7ded3683-2335-41e4-
915f-60de54d353e0.wav
```

So a path of `file:///var/lib/freeswitch/recordings/%TE/archive/%YY/%ME/%DD` will point to the right folder for each tenant.

If you make changes to the tenants definition file, you need to restart AudioVault.

Setting up QueueMetrics

In order for QueueMetrics to access those recordings, you need to set the following configuration properties:

```
audio.server=it.loway.app.queuemetrics.callListen.listeners.JsonListener
audio.jsonlistener.url=https://audiovault.my.server/search/
audio.jsonlistener.method=POST
```

```
audio.jsonlistener.searchtoken=Sup3rZ3brA
audio.jsonlistener.verbose=false
audio.html5player=true
```

Note that the URL is based on the public URL as set in AudioVault plus the `/search/` suffix. If there is a defined tenant, it is added to the URL as `?tenant=t1.example.com`

The easiest way to make sure this configuration is correct is to have Uniloader generate it for you after making sure it is correct - see Diagnostics: Test an AudioVault server for details.

While it's very easy to do, you should **never** send audio recordings unencrypted over the internet! Make sure you have an HTTPS proxy correctly configured instead of just opening up a firewall port and pointing it to AudioVault. See Securing AudioVault with HTTPS.

Easy testing with ngrok

The easiest way to make AudioVault available to an external QueueMetrics is by running ngrok - https://ngrok.com/

By running:

```
ngrok  http 4000 --region eu
```

It will print a public HTTP and HTTPS address that you can configure in AudioVault and QueueMetrics-Live and it will be your inbound tunnel.

As a paid service, ngrok offers custom or reserved domains (so that they don't change on each run) and traffic plans to match your production usage.

Securing AudioVault with HTTPS

You definitely don't want to serve recordings as unencrypted traffic. You have two options:

- having AudioVault serve HTTP requests, and binding it so that it is accessible from `localhost` only. This presumes that you will put an HTTPS proxy in front of it. This option gives you the highest flexibility in terms of logging, rewriting, etc.
- having AudioVault serve answers in HTTPS directly. It's very handy in case your PBX server already has (and likely renews automatically) an HTTPS certificate. We can use the same certificate ourselves, to start a new server on a different port.

On start-up, AudioVault will print whether it's running in HTTP or HTTPS mode.

AudioVault's native HTTPS server

If you run AudioVault on a server where other HTTPS services are configured, you can simply use

the same set of certificates. Say for example that it uses a script called `dehydrated` - https://dehydrated.io/ - to create them and keep them fresh.

You could run then:

```
uniloader av serve \
   --path file:/mnt/audio \
   --token Sup3rZ3brA  \
   --public-url https://audiovault.my.server:4000 \
   --bind-to :4000 \
   --certificate /etc/dehydrated/certs/popk.net/fullchain.pem \
   --certificate-key /etc/dehydrated/certs/popk.net/privkey.pem
```

If both `--certificate` and `--certificate-key` parameters are specified, the server will try and use them and start in HTTPS.

This works just as well if you use a certificate you bought from a commercial provider.

Make sure you restart AudioVault when certificates change, or, if you want to be on the safe side, just do it nightly.

Please note that the public URL, in this case, will contain the custom port we use for HTTPS.

If the certificate is a wildcard, AudioVault will reply to any valid name. This is useful in the case you use multiple domains for multiple tenants, so that you can have the same AudioVault reply to each customer's domanin name.

Using an HTTPS proxy

In the example above, you should set up an HTTPS proxy (e.g. *nginx*) that will redirect over HTTP to a local instance of AudioVault. In this case, it is advisable to use the parameter `--bind-to localhost:4000` so that AudioVault is not reachable externally.

A very simple, all-in-one proxy service is Caddy; it can easily be installed on CentOS with `yum install caddy` where will automatically generate HTTPS keys for you through Let's Encrypt and will manage renewals all on its own.

A very basic configuration in `/etc/caddy/caddy.conf` could then be:

```
audiovault.my.server {
    proxy / http://127.0.0.1:4000 {
         header_upstream X-Forwarded-Proto "{scheme}"
         header_upstream X-Forwarded-Host "{host}"
         header_upstream X-Forwarded-Port "{server_port}"
    }
    gzip
    tls "youremail@audiovault.my.server"
}
```

And all HTTPS request would transparently be sent to AudioVault.

Caddy is a very powerful piece of software and has many tricks up its sleeve - see https://github.com/caddyserver/caddy - you should e.g. add request logging, and possibly IP-address based security checks, as appropriate for your company.

find: Testing a configuration

Testing drivers, especially if there are many of them, can sometimes be tricky; as a debug aid, there is a way to run everything from the command line:

```
uniloader av find \
   --path file:/mnt/audios/a-%YY \
   --call-id 1606400741.4 --timestamp 1606400744

2021/01/20 18:25:50 [FileSearch] Mounted path: /mnt/audios/a-%YY
2021/01/20 18:25:50 ========= RESULTS FOUND:
2021/01/20 18:25:50   #| Type|     Size|                                  File
2021/01/20 18:25:50 ---|-----| --------|--------------------------------------
2021/01/20 18:25:50   1|  MP3|  4634277|                audio_1606400741.4.mp3
2021/01/20 18:25:50   2|OTHER|     1365|                 chat_1606400741.4.txt
```

In this case, we can see that there are two media available for this call; an audio recording and a chat transcript.

Note that we need the original timestamp in order to expand date tokens - e.g. to know the day or month of the call you are looking for. There is no need to pass it explicitly if the call-id has the format "12345678.12" like is the default for Asterisk systems.

If you are testing a multi-tenant configuration, do not forget to set your tenant through the `--for-tenant` option.

A complete debug cycle when setting up AudioVault can be found in the documentation for the command test audiovault.

Configuration files

Uniloader is able to read and write configuration files and properties.

QueueMetrics configuration.properties / *arch_properties* table

QueueMetrics properties used to be stored in text file called `configuration.properties`. With version 24.10, they were moved to a database table called `arch_properties`. Uniloader works identically - but if you want to connect to the database of a QM system, you need to specify it using the following properties:

```
OPTIONS:
   --dburi value  The database to connect to. E.g.
'tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1'
   --login value  A database user (default: "queuemetrics")
   --pwd value    A database password [$PWD]
```

If no database is specified, the text file will be attempted. An example of these properties are used is given in the example below.

Reading a property: get

You can easily check the status of a QueueMetrics property programmatically, by calling:

```
uniloader cfgfile get -p realtime.agentPausedOnLogin
false
```

or (using the database):

```
uniloader cfgfile --dburi "tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1" --pwd
javadude get -p realtime.agentPausedOnLogin
```

You will usually save the property to a bash variable for further decisions.

Full invocation:

```
NAME:
   uniloader cfgfile get - Reads a property and prints it on STDOUT.

USAGE:
   uniloader cfgfile get [command options] [arguments...]

DESCRIPTION:
```

```
   Reads a configuration.properties file and
prints on STDOUT the value that was found.

Can be used in Bash like:
  AUDIO=$(uniloader cfgfile get -f configuration.properties -p audio.url)
as to capture the value in a variable.

OPTIONS:
   --properties-file, -f "configuration.properties"     The properties file
   --property, -p                                       The name of the property to
read
   --default, -d                                        The default value
```

If you specify a default value, it will be returned in case the property was never set.

Writing a property: put

You can set a property to a desired value:

```
uniloader cfgfile put -p realtime.agentPausedOnLogin -v true -c "Customization 1"
```

In file mode, the optional comment will be prepended to the property, like:

```
# Customization 1
realtime.agentPausedOnLogin=true
```

Full invocation:

```
NAME:
   uniloader cfgfile put - Sets a property in a properties file.

USAGE:
   uniloader cfgfile put [command options] [arguments...]

DESCRIPTION:
   This command will set the property you define in a Java
properties file or similar.

If the property is already present with the same value, it is not changed;
otherwise the previous value is commented out and the new one is appended
to the end of the file with an optional comment.

The file is always overwritten.

OPTIONS:
   --properties-file value, -f value  The properties file. If --dburi is specified,
unused.
```

```
   --property value, -p value         The name of the property to set.
   --value value, -v value            The new value.
   --comment value, -c value          An optional comment
   --forced-replacement value         If set to 1, the property and its comments will
be replaced (default: 0)
   --locked value                     If set to 1, the property is locked and cannot
be changed within QM (default: 0)
   --signed-as value                  The login of a user that will 'sign' any
updates. Only works for database mode.
```

In case the property is unchanged, the transaction is not performed. A property won't be changed if you set it to the same value multiple times - Uniloader detects that the value does not need changing and won't perform the transaction. In this case, a message will be printed for your reference.

Locking a property

By **locking** a property, that is setting it to a given value and adding `--locked 1` to the invocation, that property will not be changeable from within QueueMetrics. Any attempt to do so will trigger an error. Any property created without that flag will by default be unlocked.

To unlock a property, you need to set it again adding `--locked 0` to its invocation. If the property value is actually unchanged, and its lock status is supposed to change, no new property is created, but its lock status is changed. If both the property value and its lock status are unchanged, then no change is performed.

When a lock status is changed, a message is printed, e.g.:

```
2024/10/29 17:53:27 Property 'my.special.property' unchanged - no update needed
2024/10/29 17:53:27 Property 'my.special.property' lock changed - Now locked=false
```

The `Now locked` part shows whether the property is currently locked or not.

Signing a change

By default, changes made through `cfgfile put` (database mode only) are recorded with a system user id of `0`, so QueueMetrics' property history cannot show who made the change. You can attribute the change to a real QueueMetrics user by adding `--signed-as <login>` to the invocation, e.g.:

```
uniloader cfgfile --dburi "tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1" --pwd
javadude \
    put -p realtime.agentPausedOnLogin -v true -c "Customization 1" --signed-as
demoadmin
```

Uniloader looks up `demoadmin` in `arch_users` and uses the resulting `user_id` as `sys_user_creazione` (when a new property row is inserted) and `sys_user_modifica` (when the previous row is closed, or

when the lock status is changed via `--locked`). If `--signed-as` is not given, `0` is used, same as before. If a login is given but does not match any existing user, the command aborts with an error:

```
User 'demoadmin' not found
```

Transcribe

The `transcribe` subcommand provides automated Speech-to-Text (STT) transcription of call recordings. It watches a directory for audio files, coordinates work allocation with QueueMetrics via its leaderboard, sends audio to a configured transcription service, and uploads the resulting transcripts back to QueueMetrics.

For ease of use, the default `./install.sh` script in Uniloader will set up a systemd unit called `uniloader-transcribe` that is ready to be enabled. It can be fully configured in its

Overview

Uniloader acts as a worker agent in a distributed transcription pipeline:

1. QueueMetrics tracks which recordings have already been transcribed and manages channel reservations.
2. One or more Uniloader instances, each using their own separate login, watch a shared audio directory and advertise themselves to QueueMetrics.
3. QueueMetrics assigns a transcription channel and service credentials to each worker.
4. The worker sends the audio file to the transcription API, polls for results, and uploads the transcript to QueueMetrics.

Multiple Uniloader instances can run in parallel against the same QueueMetrics system — QueueMetrics ensures each file is assigned to only one worker.

Setting up transcription users in QM

On the QueueMetrics side, you can easily create new login credentials for Uniloader processes running transcribers:

- user class `ROBOT`
- custom keys: `CHAT_WRITE`
- enabled: Yes

In recently installed systems, a ready-made `transcriber` user is ready to be used - just set a password and enable it for logins.

Each Uniloader instance **must use their own unique QueueMetrics login**. If two instances share the same login, QueueMetrics cannot distinguish them and will mismanage their channel reservations, leading to duplicate or missed transcriptions. As you need one per server, if you have multiple servers A and B, we suggest naming them `transcriber-a` and `transcriber-b`.

Transcription Servers and Capacity

Transcription servers are configured entirely within QueueMetrics. Each one points to a transcription service (local or remote), carries its credentials and configuration parameters, and has a maximum parallel capacity — the number of files that service can process simultaneously.

QueueMetrics can hold multiple transcription servers, each with a configurable *priority*. When a worker requests a channel reservation, QueueMetrics assigns from the highest-priority server that still has capacity. Lower-priority servers act as overflow: they are only used when higher-priority ones are full. This lets you combine a fast local server for normal load with a cloud API that absorbs peaks.

Servers, their capacities, and their priorities can be added, removed, or changed in QueueMetrics at any time without restarting Uniloader. Changes take effect immediately.

Drivers

A *driver* is the transcription backend Uniloader routes audio to. The driver to use is assigned by QueueMetrics at reservation time; you do not configure it locally. It supports a specific backend's API and carries credentials and any configuration parameters set on the QueueMetrics side.

openai-tts — OpenAI Transcription

Sends audio to the official OpenAI transcription API at `api.openai.com`. The API key and model are configured on the QueueMetrics side; Uniloader receives them as part of the channel reservation.

For supported audio formats, language codes, and model options, refer to the OpenAI Speech-to-Text documentation.

While the model will auto-detect the language used, it is a good idea to set it explicitly if it is known in advance.

openai-api-tts — OpenAI-Compatible API

Sends audio to any API that implements the OpenAI transcription endpoint, such as a local model running with whisper.cpp or a self-hosted proxy. The OpenAI transcription endpoint has become a de-facto standard for STT and diarization, so most local services and many STT providers implement it.

The base URL, API key, and model are configured on the QueueMetrics side. Audio format support depends on the specific backend in use.

In most cases, you will need a model that supports both transcription (speech-to-text) and diarization (speaker identification).

deepgram-tts — Deepgram

Sends audio to the Deepgram cloud API at `api.deepgram.com`. The API key is configured on the

QueueMetrics side and sent as an `Authorization: Token` header.

Transcription options travel in the request URL. Uniloader defaults to English with paragraph segmentation, and adds Deepgram's diarization model when the task tag is `TD`. Any parameter set on the QueueMetrics side overrides these defaults, so you can select a different model or language without changing anything locally.

For the available options, refer to the Deepgram documentation.

krokoai-tts — Kroko-AI

Sends audio to a self-hosted Kroko-AI server over a WebSocket connection. Only the server URL is configured on the QueueMetrics side, in the form `ws://host:port`; this driver uses no API key.

Kroko-AI transcribes but does not label speakers, so it accepts the `TR` task tag only. To tell agent and customer apart with this driver, use a stereo recording together with `--stereo-agent`, which transcribes each channel separately.

The Kroko-AI server only accepts 16 kHz mono audio. Uniloader converts every file with `ffmpeg` before sending it, so `ffmpeg` must be available on the `PATH` when this driver is in use.

Kroko-AI returns word-level timings rather than usable sentences. Uniloader rebuilds the segments from the individual words, starting a new segment after a sentence ends and the speaker pauses for at least two seconds.

Task Tags

A *task tag* tells QueueMetrics what type of work this worker can handle. You configure this with `--task-tags`.

Tag	Meaning
`TD`	Transcribe + Diarize — STT with automatic speaker labelling
`TR`	Transcribe only — STT without speaker detection

Only one tag may be active at a time. The tag must match what QueueMetrics expects for the configured transcription channels.

You can get a list of supported drivers and tags through uniloader test transcribe.

Running the Command

```
uniloader transcribe [flags]
```

Required Flags

Flag	Type	Description
`--qm-url`	string	Base URL of the QueueMetrics instance (e.g. `https://qm.example.com`)
`--qm-login`	string	QueueMetrics username (see Creating users).
`--watch-dir`	string	Directory to watch recursively for audio files

The password can be provided via `--qm-password` or the `QM_PASSWORD` environment variable (preferred, to avoid exposing credentials in shell history or process listings).

Optional Flags

Flag	Type	Default	Description
`--qm-password`	string	—	The password for the user defined in `qm-login`. You could also set it in the `QM_PASSWORD` environment variable.
`--extensions`	string	`.wav,.mp3,.raw`	Comma-separated audio file extensions to process
`--workers`	int	2	Number of parallel transcription workers
`--batch-size`	int	500	Files per QueueMetrics existence-check batch (max 1000)
`--retry-delay`	int	10	Seconds between reservation retries when all channels are busy
`--poll -interval`	int	5	Seconds between driver status polls
`--quiet-period`	int	20	Seconds a file's modification time must be stable before processing; 0 disables
`--task-tags`	string	`td`	Task tag to advertise to QueueMetrics (`td` or `tr`)
`--skip-prc`	int	0	Percentage of reserved files to skip without transcribing (0–99; 0 disables)
`--http-log`	string	—	File path for a full HTTP request/response trace log
`--stereo-agent`	string	—	Stereo channel assignment: `L` (agent on left) or `R` (agent on right)

Headerless recordings (`.raw`)

Asterisk can write its recordings as `.raw` files: raw samples, with no header saying how they should be read. Uniloader processes them by default, and assumes what Asterisk writes for a stereo recording: signed 16-bit little-endian samples, 8 kHz, two channels.

This is achieved in the dial-plan using the following command:

```
...
exten => 551,n,MixMonitor(stereo_recording_${UNIQUEID}.raw,Db)
...
```

The option `D` in MixMonitor writes a stereo file without headers, where left and right frames are interleaved; `b` starts recording only when there is a bridged channel (that is, when both caller and agent are available).

Uniloader can process those files natively. When the two legs are split with `--stereo-agent`, a single `ffmpeg` run reads the file and separates the channels; without it, the recording is downmixed to mono. Either way the audio is then handed to the driver in the format it expects.

A `.raw` file recorded at a different rate, such as 16 kHz `slin16`, is read at the wrong speed, and the transcription will be poor. Record at 8 kHz, or convert those files to WAV before Uniloader sees them.

Sizing `--workers`

The number of workers should be in the same order of magnitude as the total parallel channel capacity configured in QueueMetrics. Too few workers leave channels idle; too many workers spend most of their time waiting for a free channel.

As a starting point, set `--workers` equal to the total channel capacity across all transcription servers. If Uniloader is distributed across multiple hosts, divide that number among them.

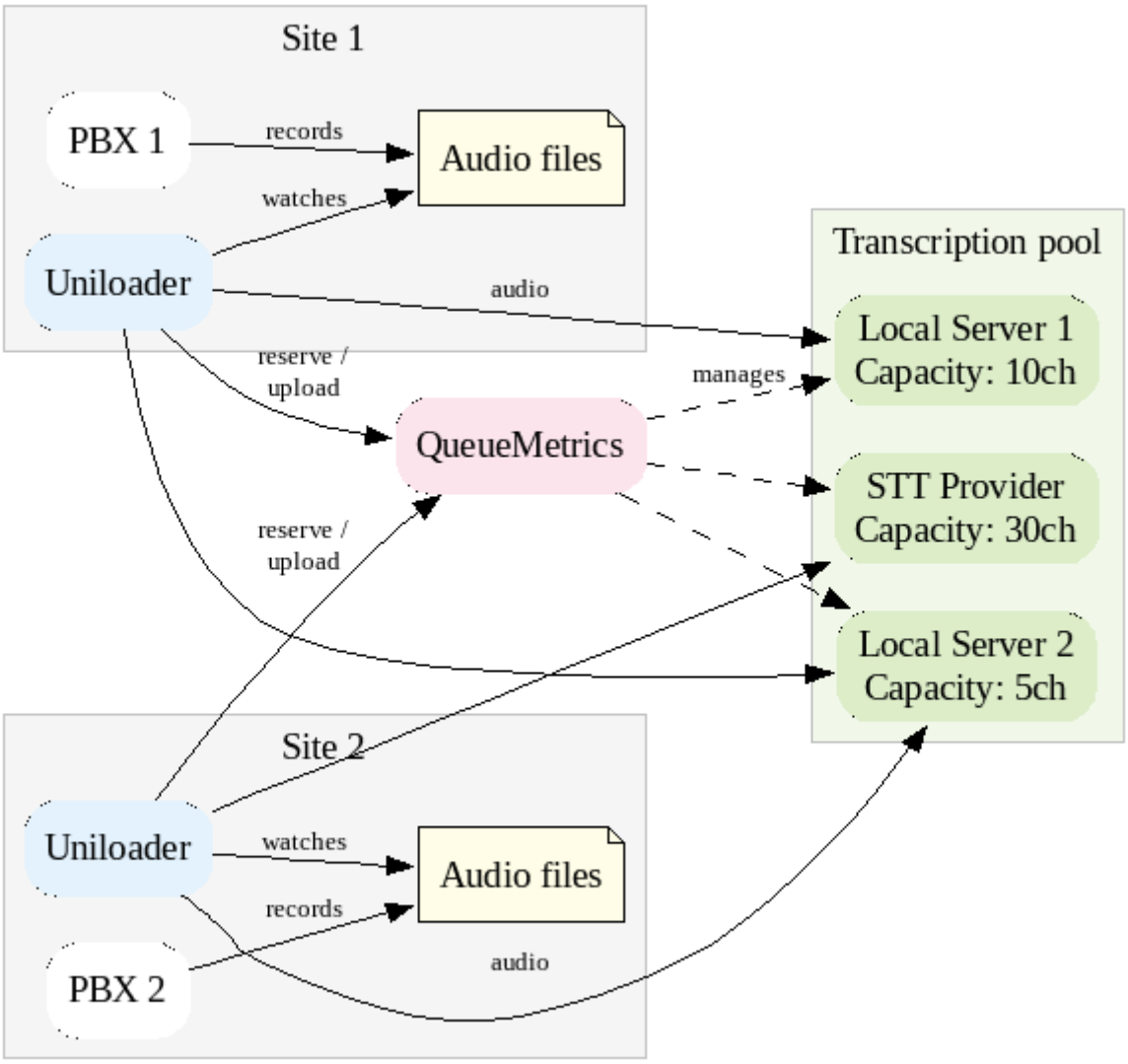
Minimal Example

```
uniloader transcribe \
  --qm-url "https://qm.example.com" \
  --qm-login "admin" \
  --watch-dir "/var/spool/asterisk/monitor"
```

Full Example

```
uniloader transcribe \
  --qm-url "https://qm.example.com" \
  --qm-login "admin" \
  --watch-dir "/var/spool/asterisk/monitor" \
  --extensions ".wav,.mp3" \
  --workers 4 \
  --quiet-period 30 \
  --task-tags "tr" \
  --http-log "/var/log/uniloader-http.log"
```

How It Works



The two Uniloader instances watch their local audio directories independently and compete for transcription capacity via QueueMetrics. QueueMetrics manages the transcription server pool and assigns channels; Uniloader sends audio directly to the assigned server and uploads the resulting transcript back to QueueMetrics.

File Discovery

At startup, Uniloader walks `--watch-dir` recursively and queries QueueMetrics for which files have not yet been transcribed. The resulting backlog is sorted newest-first by modification time, so recent recordings are processed before older ones.

After the initial walk, a filesystem watcher monitors the directory tree for new files. Newly detected files are placed at the front of the queue and processed before backlog items.

Quiet Period

To avoid processing files that are still being written (e.g. by a live recorder), files are held in quarantine until their modification time has been stable for `--quiet-period` seconds. Set this to `0` if

mtime updates are unreliable on your storage (e.g. some SMB/NFS configurations).

Worker Loop

Each worker:

1. Takes a file from the queue.
2. Requests a channel reservation from QueueMetrics. If all channels are busy, it waits `--retry-delay` seconds and retries.
3. Submits the audio file to the assigned transcription driver.
4. Polls the driver every `--poll-interval` seconds. Sends a heartbeat to QueueMetrics every 30 seconds to keep the reservation alive.
5. On completion, uploads the transcript to QueueMetrics and releases the reservation.

Transient errors (network failures, server errors) are retried up to 3 times. Permanent errors (unsupported file format, bad audio) are reported to QueueMetrics without retry.

Transcription Result

Each uploaded transcript contains:

- **Speaker list** — speakers in order of first appearance (`SPEAKER_00`, `SPEAKER_01`, ...)
- **Segments** — each with speaker label, start time (ms), transcribed text, and confidence score
- **Language** — detected language code (e.g. `en`)
- **Driver ID** — identifies which service and channel processed the file

Stereo Agent Mode

For stereo recordings where one channel is the agent and the other is the customer, use `--stereo-agent` together with `--task-tags tr`.

```
uniloader transcribe \
  --qm-url "https://qm.example.com" \
  --qm-login "admin" \
  --watch-dir "/var/audio" \
  --task-tags "tr" \
  --stereo-agent L
```

`--stereo-agent L` means the agent's voice is on the left channel; `R` means the right channel.

`ffmpeg` must be available on the system PATH.

How it works:

1. The stereo file is split into two mono WAV files via ffmpeg.

2. Each mono file is transcribed separately (no diarization within a channel).
3. The results are merged and re-sorted by timestamp.
4. The output transcript has exactly two speakers: `Agent` and `Customer`.

This approach produces cleaner speaker separation than diarization on a mixed mono track.

`--stereo-agent` requires `--task-tags tr`. The two flags are validated together at startup.

HTTP Trace Log

`--http-log <file>` writes a detailed trace of every HTTP request and response to the specified file. This is useful for debugging connectivity problems, inspecting transcription API responses, or measuring service latency.

Each exchange is written as a block:

```
----- w1-7: transcribe -- 2026-04-30 14:26:24 -> 2026-04-30 14:26:25 -------
POST https://api.openai.com/v1/audio/transcriptions multipart: file=call.wav (245632
bytes), ...

Status: 200 - took 1234 ms
{"language":"en","segments":[...]}
```

The caller identity (`w1-7`) encodes the worker number and attempt count. Other identities include `engine` (startup checks) and operation names such as `reserve`, `heartbeat`, and `upload` for QueueMetrics interactions.

Sampling (`--skip-prc`)

To reduce transcription costs when processing high call volumes, `--skip-prc N` causes N% of reserved files to be skipped without transcription. Skipped files are reported to QueueMetrics with status `skip`.

For example, `--skip-prc 20` transcribes roughly 80% of files, skipping the rest at random. Custom PBX settings

```
Uniloader can run on several Asterisk-based PBXs - on custom hardware, or in specific
distributions.
Here are reported some configuration hints for specific systems.


[[YEASTARMYPBX]]
Yeastar myPBX
~~~~~~~~~~~~~~
```

```
MyPBX related setup
^^^^^^^^^^^^^^^^^^^

For this system you need to download the Uniloader Installation script and run it, it
will install Uniloader automatically.

- Go into _/persistent_ in case you have a Yeastar U PBX or _/ysdisk/support/tmp_ if
you have a Yeastar S PBX
- Download the script with _wget http://get.queuemetrics-live.com/yeastar_
- Execute the script with _sh yeastar_
- Follow the instructions
- When finished restart the PBX

QueueMetrics Live related setup
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Integrating MyPBX with QueueMetrics Live requires some modifications on the
QueueMetrics Live settings.
This can be easily performed through a web page tool reachable from the QueueMetrics
Live home page by following the below steps:

- Log on QueueMetrics Live with administrative rights
- Click on the "Edit system parameters" under the "Administrative tools" subset
- Look at the configuration key _callfile.monitoring.channel=Local/$EM@from-internal_
and change as _callfile.monitoring.channel=SIP/$EM_
- Repeat the same for the configuration keys _callfile.outmonitoring.channel_ and
_callfile.customdial.channel_
- Save, then log-off from QueueMetrics Live
```