The QueueMetrics Uniloader User Manual

2019/04/02

The QueueMetrics Uniloader User Manual

Loway

2019/04/02

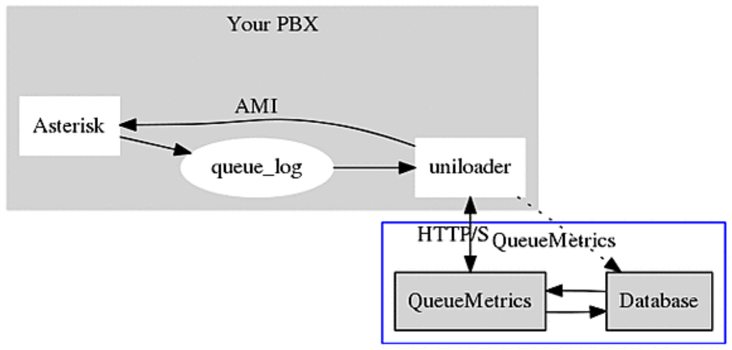
Revision History		
Revision 19.04 - covers Uniloader 0.6.0	*2019/04/02*	*L*

Table of Contents

List of Tables

Chapter 1. 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.

Chapter 2. Installation

If you run a CentOS-based PBX system, you can use the easier RPM install; if not refer to the manual install below.

Automated installation (RPM)

You can just run the following commands on your PBX:

```
wget -P /etc/yum.repos.d http://yum.loway.ch/loway.repo
yum install uniloader
```

Uniloader will be installed under `/usr/local/uniloader` and will be added to the system path. A basic configuration will be set in `/etc/sysconfig/uniloader`; Uniloader will be immediately run as a daemon and it will start automatically on reboot.

You will be able to control Uniloader by entering:

```
service uniloader start
service uniloader stop
service uniloader restart
```

Uniloader also adds a separate "tracking" service for Asterisk events - see Event tracking the section called “Installing event tracking through RPM” [16].

Manual installation under Systemd (CentOS, Debian/Ubuntu)

The Uniloader can be downloaded from http://queuemetrics-live.com/uniloader.jsp

First log in as root, then download the package as TGZ, uncompress it under `/opt` and create a link so the command can be called from anywhere.

```
cd /opt
wget http://downloads.loway.ch/qm/uniloader-0.4.1.tar.gz
tar zxvf uniloader-0.4.1.tar.gz

cd /usr/bin/
ln -s /opt/uniloader-0.4.1/bin/uniloader_amd64 uniloader
```

Now if we run `uniloader` from the shell, it will print its default message.

All files in the examples below are available within the Uniloader package, under `init-scripts/systemd`, so you can just copy them to the right location.

Uniloader

Create a configuration file as `/etc/uniloader` and set its values (edit is as to suit your system):

```
QUEUELOG=/var/log/asterisk/queue_log

# On-premise QueueMetrics instance
URI="mysql:tcp(127.0.0.1:3306)/queuemetrics?allowOldPasswords=1"
```

```
LOGIN=queuemetrics
PASS=javadude
TOKEN=P001

# QueueMetrics-Live
#URI=https://my.queuemetrics-live.com/CHANGEME
#LOGIN=webqloader
#PASS=CHANGEME
#TOKEN=
```

Now create a unit file as `/lib/systemd/system/uniloader.service`:

```
[Unit]
Description=Loway Uniloader
After=syslog.target network.target

[Service]
Type=simple
EnvironmentFile=/etc/uniloader
Environment=UPASSWD=${PASS}
Nice=15
KillMode=process
PIDFile=/var/run/uniloader.pid
ExecStart=/usr/bin/uniloader --src "${QUEUELOG}" \
          upload --uri "${URI}"  --login "${LOGIN}"  --token "${TOKEN}"  \
          --pid /var/run/uniloader.pid
RestartSec=1
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Unitracker

If you also need the tracking service, create a configuration file as `/etc/unitracker` and set its values (edit is as to suit your system):

```
AMIHOST=127.0.0.1
AMIPORT=5038
AMIUSER=admin
AMISECRET=amp123

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

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

OUTBOUNDTHRESHOLD=300
```

Now create a unit file as `/lib/systemd/system/unitracker.service`:

```
[Unit]
Description=Loway Unitracker (Uniloader)
After=syslog.target network.target

[Service]
Type=simple
EnvironmentFile=/etc/unitracker
Environment=AMISECRET=${AMISECRET}
Nice=15
KillMode=process
PIDFile=/var/run/unitracker.pid
ExecStart=/usr/bin/uniloader track --host "${AMIHOST}" --port "${AMIPORT}" --login "${AMIUSER}" \
          --debugfile "${DEBUGFILE}" \
          --moh "${ENABLEMOH}" \
          --parkedcalls "${ENABLEPARK}" \
          --outboundcalls "${ENABLEOUTBOUND}" --outboundthreshold "${OUTBOUNDTHRESHOLD}"
RestartSec=1
Restart=on-failure

[Install]
```

```
WantedBy=multi-user.target
```

Starting and enabling the services

Under Systemd, you need to notify the daemon that there are new init files, tell it that you want them run on boot, and start them.

Run the following commands for Uniloader:

```
systemctl daemon reload
systemctl enable uniloader
systemctl start uniloader
```

To check that the loader is running:

```
systemctl status uniloader
```

And to restart the service after you make some changes to the configuration file:

```
systemctl restart uniloader
```

All logs will be sent to the system journal, and are visible as:

```
journalctl -u uniloader
```

The same procedure must be followed to enable `unitracker`.

Manual installation

The Uniloader can be downloaded from http://queuemetrics-live.com/uniloader.jsp

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:
   0.4.0 - build: 82-20170828.1439 - OS: linux/amd64 - RT: go1.5.3

COMMANDS:
   upload, u    Uploads a source file to a QueueMetrics or QueueMetrics Live instance
   testupload   Tests a data upload connection
   track, t     Tracks Asterisk events and creates relevant queue_log entries.
   amitest      Tests an AMI connection
   help, h      Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --src, -s "/var/log/asterisk/queue_log"      The source queue_log file to be uploaded
   --cacert                                     An optional CA Cert file, in .pem format
   --verbose-back-end                           Enables verbose back-end logging. Default: false.
   --read-pipe                                  The source file is a 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 the section called “Yeastar myPBX” [33].

Chapter 3. 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 simpy 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.

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

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.

Chapter 4. Usage

Uploading data

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

```
NAME:
   upload - uploads a source file to a QueueMetrics instance

USAGE:
   command 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
   --token, -t                In MySQL mode, the partition. In HTTP/S mode, usually blank or server-id
   --splitter, -x             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.
```

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 / Qloader / instances point to 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".

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: at the moment, it must be "any" - meaning that if a string is found, it is considered a match
- 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.

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.

Chapter 5. 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 the section called “Example: Uploading data to a local QueueMetrics system” [5].

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 the section called “Example: Uploading data to a QueueMetrics Live system” [6].

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.

QueueMetris-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_AGENT_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
```

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.

Chapter 6. Event tracking

This feature is experimental.

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 "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]
   --debugfile                  A debug file to dump AMI data to
   --moh "1"                    When set to 1, tracks Music-on-Hold events on queues.
   --parkedcalls "0"            When set to 1, tracks parked calls.
   --outboundcalls "0"          When set to 1, tracks outbound calls.
   --hotdesking "0"             When set to 1, rewrites hotdesking information.
   --outboundthreshold "300"    The answer threshold (in ms) for calls to be tracked as outbound.
   --noeventblacklisting "0"    When 1, events are not blacklisted. Useful for taking traces.
   --pid                        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.

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 6.1. 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 defaut, 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

#Ony 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 distribuited 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 tranfer 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 tranfer 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 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 trasfers.

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 ts 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. This way calls to internal PBX "service" numbers are not tracked. You can adjust the threshold as needed though the `outboundtheshold` 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 an the rest of the call is tracked as a normal inbound call.

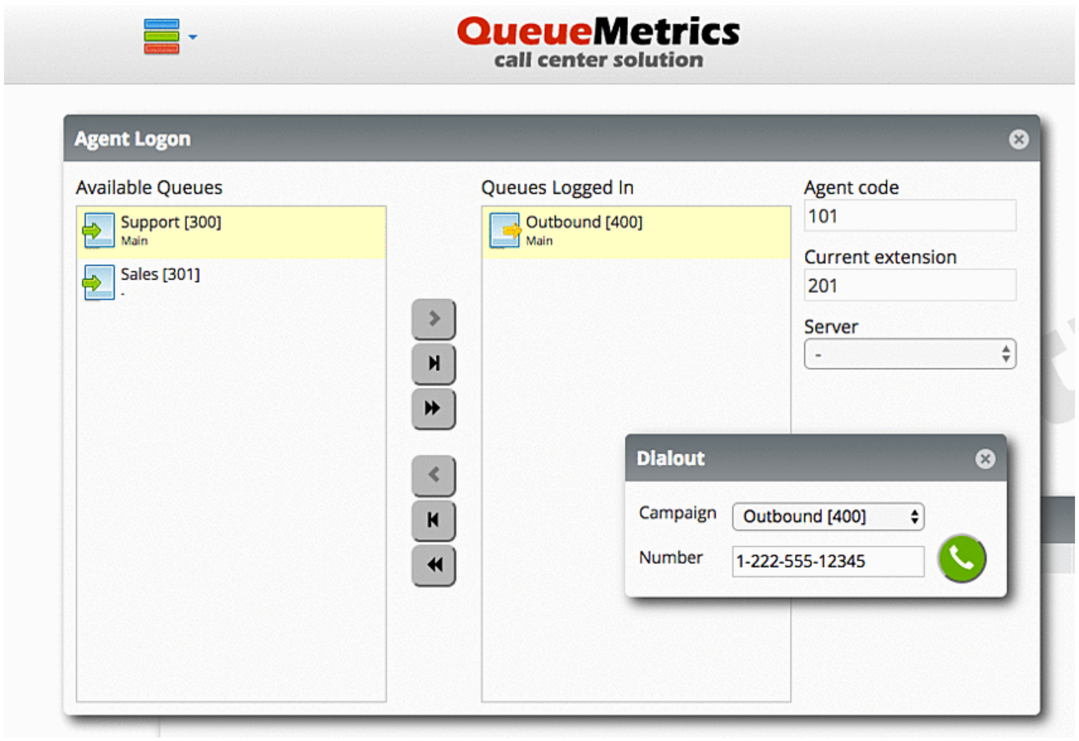
Uniloader will also join the agent to the supposed queue and log her off by the end of the call, so that reports and realtime monitoring in QueueMetrics appear correct.

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.



Agent code: 101
Current extension: 201
Server: -
Campaign: Outbound [400]
Number: 1-222-555-12345

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

If you find that some calls are missing events or have multiple events and you are able to find a consistent pattern, you should run:

```
./uniloader track .... --debugfile debugdata.txt --noeventblacklisting 1
```

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.

When you are done, you should send Loway:

- The file that was just created
- Your queue_log file that was produced (or at least its relevant lines)
- An indication of which call is displaying wrong events

Chapter 7. FreeSwitch / FusionPBX support

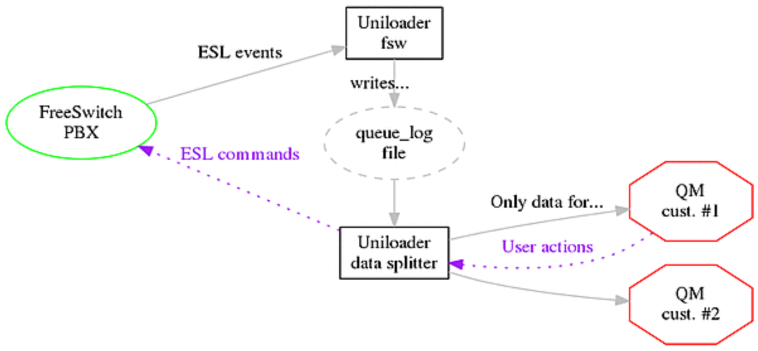
This feature is experimental.

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.

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`.

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*.

Setting up

You can install Uniloader normally; make sure you enable both services `uniloader` and `uniloader-fsw`.

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 "127.0.0.1"                    Your FreeSwitch server
   --port "8021"                         The ESL port on FreeSwitch
   --auth "ClueCon"                      The ESL auth secret [$AUTH]
   --queuelog                            The queue_log file to write
   --events                              A debug file to dump mod_callcenter events to
   --ps-uri "localhost/fusionpbx"        A FusionPbx Postgres database to connect to
   --ps-login "fusionpbx"                A FusionPbx database user
   --ps-pwd                              A FusionPbx database password [$FUSIONPWD]
   --shorten-domain "0"                  If 1, the domain will be shortened
```

When setting up:

- The *queuelog* option should create a queuelog file. It can be 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 use FusionPBX, credentials to the database can be entered in *ps-uri*, *ps-login* and *ps-pwd*. If you don’t use it, set *ps-uri* to a single dash.
- 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 database connection and the ESL connection can be checked using `uniloader test postgres` and `uniloader test fsw-esl`.

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 the section called “Splitting a single queue_log file into multiple back-ends” [10].

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 the section called “Feedback actions: proxying AMI” [9].

In QueueMetrics you need to set:

```
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 by running `uniloader pbxinfo fusionpbx`, as explained in PbxInfo for FusionPBX the section called “FusionPBX” [29].

Chapter 8. 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.

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
```

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_rule_wa
2019/02/22 10:03:29 <ESL: 75082016-6394-4738-b896-b9121c060612|longest-idle-agent|local_stream://default|system|false
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_time|bus
2019/02/22 10:03:29 <ESL: 739a4112-d755-4977-bf2b-d2b9037babd0|single_box||callback|{call_timeout=15}user/200@10.10.5
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:
```

Chapter 9. PBX Information

Uniloader is able to read and display PBX information.

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 the section called “Uploading configuration to a QueueMetrics instance” [30]

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 "localhost/fusionpbx"   A Postgres database to connect to
   --ps-login "fusionpbx"     A database user
   --ps-pwd             A database password [$FUSIONPWD]
   --single-tenant         If you specify a domain (tenant), only that tenat will be read.
```

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).

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.

Chapter 10. Configuration files

Uniloader is able to read and write configuration files.

QueueMetrics configuration.properties

Reading a property

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

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

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 prope
   --default, -d                                                                             The default value
```

Writing a property

You can set a property to a desired value:

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

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, -f "configuration.properties"     The properties file
--property, -p                                                                           The name of the prope
--value, -v                                                                              The new value.
--comment, -c                                                                            An optional comment
```

Chapter 11. 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.

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