Loway

QueueMetrics

call center suite

Accessing QueueMetrics through its JSON interface

Loway SA

Version 22.02, 2022/02/16

Table of Contents

Document contents

This document details how to access and use the JSON access functionality in Loway QueueMetrics. This makes it possible for your programs to leverage the power of QueueMetrics by calling a very simple API, with bindings available in nearly every programming language.

Revision history

- Apr 13, 2021: Using Raw blocks
- Nov 27, 2020: Action customdial and hot-desking
- Apr 7, 2020: Added JSON data upload
- Jan 21, 2020: Updated list of Data Blocks
- Mar 29, 2019: Query filters
- Apr 20, 2018: Autoconfiguration
- Jun 27, 2016: Tomcat 8
- May 04, 2015: Added PBX Actions methods
- Jan 27, 2015: Added auto-generated list of methods
- Oct 15, 2014: New configuration methods
- May 19, 2014: First draft

What is JSON?

Wikipedia defines JSON as:

JSON, or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages.

This means that, whatever your programming language of choice, you can surely find a JSON library for it; and once you have the library, connection to QueueMetrics is straightforward.

The pipe symbol "|" currently is not behaving as intended and should be encoded with **"%7c"**.

Older version of Tomcat allow it to stand as it is. Our proxies instead are more aggressive, and reject it correctly with an error 400, because they cannot decode the request.

Example of working URL:

```
https://localhost/QmRealtime/jsonStatsApi.do?queues=0002%7c5000
```

Not working URL:

```
https://localhost/QmRealtime/jsonStatsApi.do?queues=0002|5000
```

Which functions does QueueMetrics export as JSON?

QueueMetrics uses the JSON API in order to:

- read and update the system configuration - e.g. reading, creating and updating agents, queues, DNIS etc.
- export the results of most analyses in a format that is immediately usable by other software.
- perform actions programmatically, like e.g. filling in QA forms.

Information is divided into blocks, i.e. sets of data that roughly correspond to the tables QM uses for its own output.

This means that you can build software that sits on top of QueueMetrics and uses its results as a starting point for further elaboration, e.g.:

- Visualizing results with complex graphs currently not supported by QueueMetrics.
- Computing period comparison analyses (one period versus another period).
- Accessing agent presence data for payroll computation.
- Creating QueueMetrics users based on an external reference system.

Of course there are many possible scenarios where you might want to use such information.

Should I use JSON or XML-RPC?

QueueMetrics ships with both an (older) XML-RPC API and a JSON API. The JSON API includes more fuctionality (system configuration) and wraps the existing XML-RPC calls. We plan to primarily support the JSON API and keep the XML-RPC one only for compatibility with legacy software. So if you are creating a new piece of software, go for the JSON API.

Example: accessing QueueMetrics from the command-line

The easiest way to interact with the JSON interface is to do it from the command line using a tool like *wget* or *curl*. All QueueMetrics JSON calls require a valid login and password, that must be passed as an HTTP basic auth.

So in order to access the list of configured agents from the command line you would simply type (all in one line):

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET http://qmserver:8080/queuemetrics/agent/jsonEditorApi.do
```

The result is a human-readable data structure that describes configured agents. For example:

```
[ {
  "group_name" : "Default",
  "PK_ID" : "71",
  "location" : "1",
  "group_by" : "1",
  "descr_agente" : "John Doe (101)",
  "chiave_agente" : "",
  "loc_name" : "Main",
  "vnc_url" : "",
  "group_icon" : "default.png",
  "real_name" : "Super Visor",
  "supervised_by" : "41",
  "current_terminal" : "",
  "nome_agente" : "agent/101",
  "xmpp_address" : "agent101@chatserver"
},
  ...more records follow....
]
```

As this format is very easy to see and understand, all JSON APIs in this guide are documented by showing an example of a command-line call using *curl*.

Make sure you remember to enable the user *robot* in QueueMetrics, or have an equivalent user you can login as.

Example: accessing QueueMetrics from Ruby

In this example we'll see how easy it is to access QueueMetrics from a scripted language like Ruby.

```
#! /usr/bin/env ruby

require 'json'
require 'open-uri'

# Settings - edit as needed
url = 'http://127.0.0.1:8084/queuemetrics'
apicall = "agent/jsonEditorApi.do"
login = 'robot'
pass = 'robot'

# call the JSON method
replyHttp = open( "#{url}/#{apicall}", :http_basic_authentication=>[login, pass])
jsonText = replyHttp.read

# decode the JSON response and print it out as a Ruby structure
reply = JSON.parse( jsonText )
puts reply
```

This very simple script gets the list of agents in QueueMetrics and prints it out as a native Ruby data structure.

The JSON Configuration API

This API lets you configure QueueMetrics in a way that is similar to the one used interactively to create and edit records.

General usage

Reading a record

Each editor has a unique name, as decribed below. Let's say we want to configure an agent. The first thing we do is to list available agents by issuing a list for editor *agent*.

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET http://127.0.0.1:8084/queuemetrics/agent/jsonEditorApi.do
```

The result is a set of records, each of which describes one agent configured. For example:

```
[ {
  "group_name" : "Default",
  "PK_ID" : "71",
  "location" : "1",
  "group_by" : "1",
  "descr_agente" : "John Doe (101)",
  "chiave_agente" : "",
  "loc_name" : "Main",
  "vnc_url" : "",
  "group_icon" : "default.png",
  "real_name" : "Super Visor",
  "supervised_by" : "41",
  "current_terminal" : "",
  "nome_agente" : "agent/101",
  "xmpp_address" : "agent101@chatserver"
},
  ...more records follow....
]
```

You can see that each record has an ID with a fixed name of PK_ID. This is the identifier for one specific record. The other information is provided to help you locate the record(s) you are looking for. It is also possible to pass along the parameter "q=Agent/101" to filter for "Agent/101".

When you know which record you want to get details about, you issue a:

```
  curl  --user robot:robot -i -H "Content-Type: application/json"
        -X GET http://127.0.0.1:8084/queuemetrics/agent/71/jsonEditorApi.do
```

The result is a full description of a record, as in:

```
{
  "aliases" : "",
  "chiave_agente" : "",
  "current_terminal" : "-",
  "default_server" : "0",
  "descr_agente" : "John Doe (101)",
  "group_by" : "1",
  "group_by__DECODED" : "Default",
  "id_agente" : "71",
  "location" : "1",
  "location__DECODED" : "Main",
  "ltCodeAssociate" : [ [ "inbound", "All", "Main" ], [ "inbound", "Q DPS", "Main" ]
],
  "nome_agente" : "agent/101",
  "nomedata_insert" : "demoadmin, 18/06/2007, 22:28",
  "nomedata_update" : "demoadmin, 05/05/2014, 12:56",
  "payroll_code" : "",
  "sip_login" : "",
  "sip_pwd" : "",
  "sip_realm" : "",
  "sip_uri" : "",
  "supervised_by" : "41",
  "supervised_by__DECODED" : "demosupervisor",
  "sys_dt_creazione" : "2007-06-18 22:28:20",
  "sys_dt_modifica" : "2014-05-05 12:56:33",
  "sys_optilock" : "63943",
  "sys_user_creazione" : "32",
  "sys_user_modifica" : "32",
  "vnc_url" : "",
  "xmpp_address" : "agent101@chatserver"
}
```


You can notice a few patterns in the data above:

- the ID is passed along in the URL.
- some fields may end in *DECODED and they contain a string version of a numeric ID. When committing, you can leave the main field blank and pass along the textualDECODED* version only. If both the DECODED version and the relevant ID are present, the DECODED version takes precedence.
- the record may contain structures (lists, hashes or tables); those are read-only and are ignored on commit.
- any field starting with *opt_* or _ is a decorator, and is read-only. It is typically used to convey additional information.
- the fields named *nomedata_insert* and *nomedata_update* are textual representation of the current record creator and last updater.
- the field called *sys_optilock* is the current optimistic lock and is needed for updating the record.

- any other fields staring with *sys_* are read-only and can be ignored on commit.
- boolean values (yes/no) are usually encoded as "1" and "0" respectively.
- when an updatable field is made up of multiple entries, they are usually separated by the pipe "|" character.

Updating a record - Optimistic locking

In order to perform an update on an existing record, you need to:

- load the current record,
- edit it,
- commit it.

It is of paramount importance that you reload the record before editing and committing it - as QueueMetrics is a multi-user application, it is possible that a record is changed before you save it. In order to detect and abort such changes, QueueMetrics uses the original value from *sys_optilock* to make sure the record was not modified when you save it. In case the record was modified, subsequent modifications will fail with an OptiLock exception; in this case you need to reload the record and repeat the sequence.

So, for example, you could read record #71 into a file....

```
curl  --user robot:robot -H "Content-Type: application/json"
      -X GET http://127.0.0.1:8084/queuemetrics/agent/71/jsonEditorApi.do
      > agent71.json
```

You would then edit it and save it again, as in:

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X POST -d @agent71.json
      http://127.0.0.1:8084/queuemetrics/agent/71/jsonEditorApi.do
```

After saving, QueueMetrics returns the new record you just saved, so you can check that your changes were saved successfully.

Some QueueMetrics editors might perform additional contextual sanity checks before saving the record and might abort if they find a logical inconsistency. For example, you cannot enter two agents with the same agent code. In this case a meaningful error message will be returned.

Creating a new record

In order to create a new record, you POST to a fake record id "-" or "0", as in:

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X POST -d @new_agent.json
      http://127.0.0.1:8084/queuemetrics/agent/-/jsonEditorApi.do
```

After saving, QueueMetrics returns the new record you just saved, so you can obtain its current PK_ID.

Deleting a record

In order to delete an existing record you perform a DELETE, as in:

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X DELETE http://127.0.0.1:8084/queuemetrics/agent/71/jsonEditorApi.do
```

There is no safety-check on delete, so a deleted record is gone forever. QueueMetrics will return the record you just deleted as it was before being deleted, so you stilll have a chance to save it and restore it.

Hyerarchical records

Some editors are hyerarchical, because they depend on other editors. For example, in order to get the screens in a report, you need to pass the screen editor the report id when listing.

You do this by adding:

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET
      http://127.0.0.1:8084/queuemetrics/reportScreen/8/jsonEditorApi.do?parent=7
```

The id's you get out of listing are then permanent, so you do not need to pass this along when updating or inserting. When inserting, you will generally need to fill in a "__DECODED" field with the name of the parent so that QM knows where to attach the new record.

Available editors

dnis

Edits the DNIS tables.

ivr

Edits the IVR tables.

class

Edits the user class tables.

user

Edits the user list. Any user must belong to a valid class.

queue

Edits the list of queues and the set of agents linked to a sepcific queue.

agent

Edits the agent properties.

agentGroup

Edits agent groups.

location

Edits the list of locations.

outcome

Edits the list of allowed outcomes.

pause

Edits the list of available pause codes.

exportJob

Created and edits export jobs.

cronJob

Created and edits Scheduled Jobs.

exportCall

Lists the set of calls in an export job. Requires the export-job ID as a parent.

This transaction is read-only; commits will not work.

report

Edits availble reports.

reportScreen

Edits the screens for a report. Requires the screen-id as a parent.

reportItem

Edits the items in a screen. Requires the screen-id as a parent.

qaForm

Edits QA forms. It is of paramount importance to make sure that all items exist and that the lengths of each section are correct.

qaItem

Edits QA items.

perftrackRule

Display Performance Tracker rules.

This transaction is read-only; commits will not work.

exportReport

Edits the available export reports.

cfgProps

Reads and edits the configuration.properties file. In order to make changes to the file, the key EDIT_CFG is required. Any changes are appended by the end of the file and the previous entry is commented out.

skills

Reads and updates the set of available skills known to the system.

knownnumbers

Reads and updates the set of known phone numbers present in the system.

customblocks

Reads and updates the set of Custom Data Blocks available for reporting on this system.

The JSON Reports API

All of QueueMetrics JSON Reports API share a common set of guidelines:

- All requests must contain a valid QueueMetrics *username* and *password*, supplied as HTTP "Basic auth". They will usually require a user holding the key ROBOT and may require additional security keys where appropriate.
- Both GET and POST requests are allowed - note that GET requests may have a size limit, while there is basically none for POST.
- Whenever multiple parameters are allowed, for example when querying for multiple data blocks, the same parameter is repeated multiple times.
- In case you need to pass the API an associative array (hash), all keys within the hash have a common prefix. E.g. an array like *{ A:1, B:2 }* which common prefix is defined as *k_* should be passed as *k_A=1&k_B=2*.
- All *dates* are to be written exactly in the format "2000-01-01.00:00:00". Make sure you do not forget the dot between the day and the hour.
- The result of all API Reports is a hash made up of blocks, that is an array of rows, each of which is an array of strings. Usually the first row of each block contains the titles for subsequent rows. It is mandatory when reading data to check which column contains the title you are looking for, as different versions of QueueMetrics might report columns in a different order.
- There is always a block called "result" that specifies whether the operation completed successfully or failed, the time it took to complete and a stack trace of the error (if any).
- Any missing parameters are read as if they were blank (and vice versa).
- The names of the data blocks you need will usually be similar to *OkDO.AgentsOnQueue*. You can get those easily from QueueMetrics - where you see the Excel/CSV export icon, copy the link in your browser and inspect it - it will contain a parameter like "S.OkDO.AgentsOnQueue.123456789". Just remove the first and last parameters.
- For QA methods, allowed grader types are: "unknown", "agent", "grader" and "caller".

For readability's sake, all examples given below that use the *curl* command are written on multiple lines. When testing them on a real system, they must be entered on one single long line instead.

If you use Tomcat 8+ as a servlet container (as it happens with QueueMetrics 17+) then it will return an error 400 if you do not accurately URL-encode all characters according to RFC 7230 and RFC 3986. This often happens when passing a set of queues that are separated by the pipe symbol - so `a|b` will not work, but `a%7Cb` will.

Reports

Obtaining statistics: QmStats

This API call will start up a session in QueueMetrics, check if the user exists and has the privilege to

run the report, run the analysis, prepare the required results and return them. At the end of the call, the QueueMetrics session is destroyed so no data is kept for further elaboration.

This means that it's usually the most efficient thing to do to request all needed response information at once, but it's wise to limit yourself to the minimum data set you will actually need, as each block takes up CPU and memory space to be marshaled between the native Java format, the intermediate JSON format and the resulting client format.

QueueMetrics poses no limits on the size of analyses you may want to run. It is advisable to run large data set analysis at night time or when nobody is accessing the system, as they may take quite a lot of RAM and CPU and this may slow down QueueMetrics for interactive users.

Method	QmStats
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.stats
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmStats/jsonStatsApi.do?
       queues=q1
       &from=2000-01-01.00:00:00
       &to=2020-01-01.00:00:00
       &block=OkDO.RiassAllCalls
       &block=OkDO.AgentsOnQueue"
```

Parameters

- *queues*: the set of queues that must be included in the analysis. They must be separated by a "|" symbol if more than one queue is passed. The queue name is the internal Asterisk queue name. *(*)*
- *from*: the begin of the reporting period, as a date. *(*)*
- *to*: the end of the reporting period. *(*)*
- *filter*: the agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity.
- *block* the output blocks that have to be exported. To specify multiple blocks, add multiple times. It is advisable to request all blocks you need in a single call, as it is way cheaper for QueueMetrics to compute different blocks on a data set in memory than to recompute it from scratch on a different API call.
- *query*: a dynamic query to be run to filter out calls (since QM 19). See Call Queries

Parameters marked with an asterisk are mandatory. All parameters must be URL-encoded to form a valid URL.

Where do I find data blocks?

A complete list of possible QueueMetrics blocks is maintained in the QueueMetrics User Manual, chapter 6 "Report Details", where each block is described in detail. The QueueMetrics User Manual can be obtained from the QueueMetrics website.

For every possible block there is a name, a description, a "shortcut code" for ease of identification and an API code (usually referred to as an "XML-RPC code" for historical reasons). That is the name of the block that has to be retrieved over JSON. In QueueMetrics 20+, this appears as "block name" when iewing a block's detail.

For example, if we want to access the "Disconnection Causes" block, we will look it up in the manual until we encounter "UN03 - Disconnection Causes".

We see that its XML-RPC code is "KoDO.DiscCauses", so that is the name of the block we'll be asking for. Block names are case-sensitive, so make sure you are writing it exactly as it appears on the User Manual.

For a quick reference, data blocks are also listed in Appendix II of this manual at Available blocks for QmStats .

Reading "raw" blocks

While most of the blocks that one can reference from the JSON APIs match the blocks that are visible within the main reports, there are some special data blocks that are designed for machine data consumption.

Those blocks are named **raw** and contain data that is not formatted, where:

- times appear as Unix timestamps and not as a date/time
- agents, pause codes, outcomes, etc. appear undecoded, as they appear on the source log
- no localizations (numbers, formats) is applied

This makes it easy to process this information (that is typically lists of calls) from third-party software.

Example

To read a list of taken calls, you would run:

```
curl --user robot:**** -i
     -H "Content-Type: application/json"
     -X GET "http://my.queuemetrics-live.com/----/QmStats/jsonStatsApi.do?
     queues=500
     &from=2000-01-01.00:00:00
     &to=2022-01-01.00:00:00
     &block=DetailsDO.CallsOkRaw"
```

Apart from the usual data (that matches the one visible in QueueMetrics) you can also see a list of call events, matching the contents of the "call detail" pop-up.

That information is returned in the field `CALLOK_events`, containing a string that in itself encodes a JSON structure, e.g.:

```
"[ {
 \"timestamp\" : 1617788374,
 \"agent\" : \"agent/301\",
 \"duration\" : 0,
 \"type\" : \"NOANSWER\",
 \"descriptionParms\" : \"\",
 \"aa\" : false,
 \"endingTimestamp\" : 1617788374,
 \"eventDuration\" : 0,
 \"agentCode\" : \"agent/301\",
 \"description\" : \"evt_ringnoanswer\"
 }, {
 \"timestamp\" : 1617788374,
 \"agent\" : \"agent/307\",
 \"duration\" : 1,
 \"type\" : \"NOANSWER\",
 \"descriptionParms\" : \"\",
 \"aa\" : false,
 \"endingTimestamp\" : 1617788375,
 \"eventDuration\" : 1,
 \"agentCode\" : \"agent/307\",
 \"description\" : \"evt_ringnoanswer\"
 } ]"
```

Each of those objects is a detailed call event: a variable, a ring attempt, an IVR keypress.... they are all in there.

Live data: QmRealtime

Obtains the real-time status of a system.

This method is very similar to QmStats but it is used to retrieve the real time stats. The same suggestions that are given for QmStats apply.

Please note that there is a difference between results produced by the API realtime calls and the realtime statistics produced through the QueueMetrics GUI when the key *realtime.members_only* is equal to true. The difference is related to the agents list shown. As the list of queues passed through the API does not point to a specific QueueMetrics queue instance, it's not possible to correctly tell elementary queues from aggregate queues having the same name. In this situation the agent list will always be computed as the union of all agents associated to all elementary queues composing the macro queue, even if an existing aggregate queue has a different set of agents assigned to it.

Method	QmRealtime
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.realtime
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmRealtime/jsonStatsApi.do?
      queues=q1
      &block=RealtimeDO.RtAgentsRaw
      &block=RealtimeDO.RtCallsRaw"
```

Parameters

- *queues*: one or more queues, separated by the pipe "|" symbol. *(*)*
- *filter*: the agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity.
- *block* the output blocks that have to be exported. To specify multiple blocks, add multiple times.

Parameters marked with an asterisk are mandatory.

Data blocks: RealtimeDO

Real-time information, as displayed in the main QM real-time page, using system defaults.

Method	Description
RTRiassunto	An overview table of the queues in use
RTCallsBeingProc	Calls being processed in real-time
RTAgentsLoggedIn	Agents logged in and paused
WallRiassunto	The wallboard top panel

Method	Description
WallCallsBeingProc	The wallboard call list
VisitorCallsProc	Calls processed
VisitorTodaysOk	Calls taken
VisitorTodaysKo	Call lost
RtAgentsRaw	Raw agent panel
RtCallsRaw	Raw calls panel

When exporting blocks, it is strongly advisable to use the raw data blocks, *RtAgentsRaw* and *RtCallsRaw*, as they are easier to parse.

For a quick reference, data blocks are aso listed in Appendix II of this manual at Available blocks for QmRealtime .

Accessing audio files: QmFindAudio

This method lets you download audio files and other meta files associated with a call. In order to retrieve the file name QM will invoke the currently configured Pluggable Modules to search within the current recording set. This can be used by third-party software that needs to retrieve audio recordings via HTTP.

Method	QmFindAudio
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.findAudio
Available since	14.06 - 16.07
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmFindAudio/jsonStatsApi.do?
      id=664745.1"
```

Parameters

- *id*: the call's UniqueID. *(*)*
- *server*: mandatory on clustered systems, the server-id.
- *timestamp*: the time-stamp of the call, as number of seconds since the epoch.
- *agent*: the agent code.
- *queue*: the queue.

Parameters marked with an asterisk are mandatory.

If QM is on a clustered setup, the *Server* parameter must be passed to qualify the Asterisk call-id.

Some PM may optionally require the *Call start*, *Agent* and *Queue* parameters; those are used for fuzzy matching of calls, e.g on an external storage. Most PMs that do an exact match do not need those parameters.

Response

There is only one response block returned, named "AudioFiles", where the caller will retrieve the filename of each recorded file and a URL to actually download the file.

The following fields are returned:

- The name for the file, or a symbolic name that is meant for human consumption
- A relative URL to download the file from
- A security token

The response may include zero or more files; it is possible that multiple recordings are present for the same call, e.g. because they are of different media type or because recordings were started and ended multiple times.

Downloading audio files

For security reasons, QueueMetrics does **not** produce hot-links to audio files that can be accessed by unauthenticated clients.

If you want to download audio files, you need to:

- Search for audio on a specific call
- Retrieve the URL and the auth token
- Download the file from the URL that QueueMetrics provides, setting the HTTP client's "User-Agent" to the access token

Example: you want to retrieve a call which unique-id is "1234.1" that was processed on a clustered server named "aleph" on July 7, 2016.

You start by asking for audio recordings for the call:

```
curl --user robot:robot -i
     -H "Content-Type: application/json"
     -X GET "http://127.0.0.1:8080/queuemetrics/QmFindAudio/jsonStatsApi.do
             ?id=1234.1&server=aleph&timestamp=1467814694"
```

QueueMetrics returns the following data block, where an audio recording was positively found:

```
{
  "result" : [ [ "Status", "OK" ],
               [ "Description", "" ],
               [ "Time elapsed (ms)", 9 ],
               [ "QM Version", "16.07-1" ] ],
  "AudioFiles" : [ [ "qm_popup_streamAudio.do?%2Faudio%2aleph%2F2016%2Fcall-1234.1-
x.mp3",
                     "call-1234.1-x.mp3",
                     "QM16071-1616625548" ] ]
}
```

In order to retrieve it, we have to download the URL returned, as relative to the QM webapp, passing the auth token that was returned:

```
curl -v  -A "QM16071-1616625548"
         "http://127.0.0.1:8080/queuemetrics/qm_popup_streamAudio.do
               ?%2Faudio%2aleph%2F2016%2Fcall-1234.1-x.mp3"
         > call-1234.1-x.mp3
```

The auth token returned is only valid for a random time that lasts from a few seconds to a few minutes after it was created. The token changes on every invocation. So you should submit the request immediately after you search for the file.

Inserting and retrieving call tags: QmInsertTag

This method is used to associate a new tag for a specific recording call file. This method is used to retrieve the list of tags related to a specific call.

By leaving the "message" empty, no new tag will be added but a list of existing tags for a specific calls can be retrieved.

Method	QmInsertTag
Auth required?	Yes - user must hold the key ROBOT and CALLMONITOR_ADDTAGS
XML-RPC method	QM.insertRecordTag
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmInsertTag/jsonStatsApi.do?
      id=664745.1
      &filename=abcd.wav
      &time=5
      &duration=7
      &message=This+is+a+tag
      &color=FF0000"
```

Parameters

- *server*: the server name in a cluster setup or empty for a not cluster setup.
- *id*: the Asterisk call-id. *(*)*
- *filename*: the recording filename associated to the tag to be inserted.
- *time*: the time (in seconds) where the tag will be placed. If empty, the tag will be placed at the beginning of the file.
- *duration*: the tag duration (in seconds). It can be empty.
- *message*: a text message. If empty, no tags will be added. This is useful for retrieve the list of tags associated to a specific call.
- *color*: a color in RGB format (from 000000 to FFFFFF).

Parameters marked with an asterisk are mandatory.

Response

There is only one response block returned, named "TagRecords", where the caller will retrieve the list of tags associated to the specific *server* and *UniqueId* parameters.

Add broadcast messages: QmBroadcastMsg

This method is used to insert broadcast messages that will be shown in the realtime broadcast message page.

Method	QmBroadcastMsg
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.broadcastMessage
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmBroadcastMsg/jsonStatsApi.do?
       text=Hello+world
       &everyone=1"
```

Parameters

- *text*: the message to be broadcast. *(*)*
- *queue*: the optional queue name to act as a filter (may be empty).
- *location*: the optional location name to act as a filter (may be empty).
- *supervisor*: a supervisor login to act as a filter (may be empty). The message will be broadcast to people reporting to that supervisor.
- *agent*: a specific destination agent (may be empty). The message will be addressed to the specified agent.
- *everyone*: set to "1" to have the message delivered to everyone.

Parameters marked with an asterisk are mandatory.

Authentication and agent information

Checking logins: QmAuth

This method is used to authenticate a username / password set against the QueueMetrics server. This can be used by third-party software that does not want to keep its own separate user database but wants to use QueueMetrics' instead.

Method	QmAuth
Auth required?	Yes
XML-RPC method	QM.auth
Available since	14.06
See also	Superceded by QmAuthenticate

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmAuth/jsonStatsApi.do"
```

Parameters

None. You only pass the authentication.

Response

The call will return one single block named "auth".

This block contains the following information:

- *UserName*: the login name.
- *Status*: OK if authentication was accepted or ERR if it was refused.
- *FullName*: The user's full name.
- *Email*: The user's email address.
- *Class*: The name of the class the user belongs to.
- *Keys*: The active key set of the user, that is, all keys given to the class plus or minus the keys that have been granted or revoked to this specific user.
- *Masterkey*: If set to 1, this user has a Masterkey, so this user will pass each key ckeck.
- *NLogons*: The number of logons the user has made. Each successful QmAuth call counts as a logon.

Authentication and password changing: QmAuthenticate

This method is used to obtain the profile of a user given its login and password. The profile is made up of both login and - where applicable - agent information. It is possible to obtain the profile either of the very user you are calling (by knowing its login and password) or of a "deferred" separate user, if you have a user that would have the admin keys to view that information in the main GUI.

The deferred username works so that:

- If a deferred username is passed, and...
- If the username/login pair are valid and the user has key USRADMIN and ROBOT
  - Then the User.* output for the deferred user is returned
  - If the user also holds the key USR_AGENT, the Agent.* output is returned as well
- The password-change function applies only to the user who logs in, not to the deferred user.

Method	QmAuthenticate
Auth required?	Yes
XML-RPC method	QM.authenticate
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmAuthenticate/jsonStatsApi.do?
       deferred=agent/101"
```

Logs on as "robot" and obtains information about "agent/101".

Parameters

- *deferred*: the other user you are accessing data for. If left out, the authenticating user is used.
- *newpass*: the new password to be changed for the user. If left blank, the password is not set.

Parameters marked with an asterisk are mandatory.

Response

There is only one response block returned, named "output", where the caller will retrieve all user data, including the live key set for that user.

The output block has the format:

Column 0	Column 1	Explanation
user.user_id	173	Internal user-id
user.login	Agent/101	
user.real_name	John Doe	
user.class_name	AGENTS	
user.keys	USER AGENT XX YY	All computed keys, space-separated
user.n_logons	37	
user.last_logon	2011-10-08 12:34:56	
user.comment		Optional
user.token	876543	Optional
user.email	me@home.it	Optional
user.enabled	true	"true" or "false"
agent.id	86	Internal agent-id
agent.description	Agent J.D. (101)	The name displayed in reports
agent.aliases		A set of aliases (if present)
agent.location	Main	Blank if none
agent.group_name	Experienced agents	Blank if none
agent.current_terminal	Sip/1234	Blank or "-" if none
agent.vnc_url	http://1.2.3.4/vnc	Optional
agent.supervised_by	Demoadmin	Blank if none, or login of the supervisor
agent.xmpp_address	xmpp:101@myserver	XMPP chat address

Column 0	Column 1	Explanation
agent.visibility_key		Optional
agent.payroll_code		Optional
password.changed	OK	OK if password was changed, blank otherwise

The following rules apply:

- Column zero contains the attribute.
- Column one contains the value of the attribute (we supply a sample in the table above).
- Attribute names are not case sensitive.
- If the user is also an agent, that is, there is an agent under the same name as the login, Agent attributes are passed.
- Blank attributes may or may not be present in the list of attributes.

The same information can also be accessed and edited through the Configuration API.

The set of known queues for an agent: QmAgentQueues

Given an agent (like Agent/101) let the caller know on which queue(s) he's supposed to work, as per the configuration on the QM interface. For each queue, we get also back a "level", that is a penalty level, like 0, 1 or 2.

It will also return the composite queues the agent is known on and the level he's scheduled on them.

Method	QmAgentQueues
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.getQueuesForAgent
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmAgentQueues/jsonStatsApi.do?
       agent=Agent/101"
```

Parameters

- *agent*: the agent code we want to retrieve. *(*)*

Parameters marked with an asterisk are mandatory.

Response

If the agent code is unknown, an exception is raised.

There is only one response block returned, named "output", where the caller will retrieve all user data, including the live key set for that user.

The output block has the format:

Column 0	Column 1	Column 2	Column 3
Agent/101	Q1	Queue 1	0
Agent/101	Q2	My Queue 2	2
Agent/101	Q3 Q4	Monitor 3 & 4	1

Explanation as follows:

- Column zero contains the agent code.
- Column one contains the queue or composite queue it is known for. These are a set of the queues as they are known in Asterisk. They are separated by either a space or a vertical pipe (|) symbol.
- Column two contains the name that such queue(s) appear in the QM interface
- Column three contains the agent level, as per:
  - 0: Main (no penalty).
  - 1: Wrap (some penalty).
  - 2: Spill (highest penalty).

The available pause codes for an agent: QmAgentPCodes

Get the current association of agents to queues.

Given an agent (e.g. "Agent/101"), the caller gets back a set of pause codes and their description as it would be visible to this agent. As QM allows protecting pause codes with security keys (so that e.g. you can have some pauses visible by some users only) QM computes the set of allowed pause from the point of view of the agent.

Method	QmAgentPCodes
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.getPauseCodesForAgent
Available since	14.06

See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmAgentPCodes/jsonStatsApi.do?
       agent=Agent/101"
```

Parameters

- *agent*: the agent code we want to retrieve. *(*)*

Parameters marked with an asterisk are mandatory.

Response

If the agent code is unknown, an exception is raised.

There is only one response block returned, named "output", where the caller will retrieve all user data.

The output block has the format:

Column 0	Column 1	Column 2	Column 3
Agent/101	03	Lunch break	PNB
Agent/101	17	E-mail	PB
Agent/101	26	Coffee break	NPNB

Explanation as follows:

- Column 0 contains the agent code.
- Column 1 contains the pause code, as should be reported in Asterisk.
- Column 2 is the description.
- Column 3 is the pause type:
  - *PB* - pause is payable and billable.
  - *PNB* - pause is payable but not billable.
  - *NPB* - pause is not payable but billable (unlikely!).
  - *NPNB* - pause is neither payable nor billable.

Quality Assessment (QA)

Retrieving QA statistics: QmQaReport

This method is very similar to QM.stats but it's used to retrieve Quality Assessment statistics.

Method	QmQaReport
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.qareport
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
       -X GET "http://localhost:8084/queuemetrics/QmQaReport/jsonStatsApi.do
       ?from=2000-01-01.00:00:00
       &to=2015-01-01.00:00:00
       &queues=q1|q2
       &form=MyForm
       &block=QualAssDO.TrkCalls
       &block=QualAssDO.Res1"
```

Parameters

- *queues*: one or more queues, separated by the pipe "|" symbol. *(*)*
- *from*: the beginning of the reporting period, in date-time format. *(*)*
- *to*: the end period. *(*)*
- *agent*: the agent code to be used as a filter.
- *form*: the form you want to report on. *(*)*
- *grader*: the grader type.
- *block*: One or more data blocks that you need to access.

Parameters marked with an asterisk are mandatory.

Allowed data blocks: QualAssDO

Method	Description
TrkCalls	Tracked calls per agent report
TrkCallsQ	Tracked calls per queue report
CallSupervs	Supervisors tracking calls report
Res1	Section 1 (as defined in the form) calls by agent report
Res1Q	Section 1 (as defined in the form) calls by queue report
Res2	Section 2 (as defined in the form) calls by agent report
Res2Q	Section 2 (as defined in the form) calls by queue report
Res3	Section 3 (as defined in the form) calls by agent report

Method	Description
Res3Q	Section 3 (as defined in the form) calls by queue report
Res4	Section 4 (as defined in the form) calls by agent report
Res4Q	Section 4 (as defined in the form) calls by queue report
Res5	Section 5 (as defined in the form) calls by agent report
Res5Q	Section 5 (as defined in the form) calls by queue report
Res6	Section 6 (as defined in the form) calls by agent report
Res6Q	Section 6 (as defined in the form) calls by queue report
Res7	Section 7 (as defined in the form) calls by agent report
Res7Q	Section 7 (as defined in the form) calls by queue report
Res8	Section 8 (as defined in the form) calls by agent report
Res8Q	Section 8 (as defined in the form) calls by queue report
Res9	Section 9 (as defined in the form) calls by agent report
Res9Q	Section 9 (as defined in the form) calls by queue report
Res10	Section 10 (as defined in the form) calls by agent report
Res10Q	Section 10 (as defined in the form) calls by queue report
AgentDetail	Tracked calls details for each defined agent

Data blocks from QualAssFormDO can be queried as well (see below).

Raw form data: QmQaFormReport

Reads a list of filled in QA forms within the requested period.

Method	QmQaFormReport
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.qaformreport
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmQaFormReport/jsonStatsApi.do
      ?from=2000-01-01.00:00:00
      &to=2015-01-01.00:00:00
      &agent=agent/101
      &queues=q1|q2
      &form=MyForm
      &block=QualAssFormDO.FormStructure
      &block=QualAssFormDO.SectionValues
      &block=QualAssFormDO.Comments"
```

Parameters

- *queues*: one or more queues, separated by the pipe "|" symbol. *(*)*
- *from*: the beginning of the reporting period, in date-time format. *(*)*
- *to*: the end period. *(*)*
- *agent*: the agent code to be used as a filter.
- *form*: the form you want to report on. *(*)*
- *grader*: the grader type.
- *block*: One or more data blocks that you need to access.

Parameters marked with an asterisk are mandatory.

Allowed data blocks: QualAssFormDO

Quality Assessment information related to QA Forms.

Method	Description
FormStructure	The data structure of specified form
SectionValues	Raw QA values for each section in forms matching the query
Comments	Comments associated to forms matching the query

QA form summaries: QmQaFormSummary

This method is very similar to QmStats but it's used to retrieve aggregated information about a specific Quality Assessment Form.

The report counts the aggregated QA statistics on calls with timestamp included in the date range specified.

Method	QmQaFormSummary
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.qaformsummary

Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmQaFormSummary/jsonStatsApi.do
      ?from=2000-01-01.00:00:00
      &to=2015-01-01.00:00:00
      &queues=q1|q2
      &form=MyForm
      &block=QualAssDO.OverallAverageFormReport
      &block=QualAssDO.ScoringItemsFormSummary"
```

Parameters

- *queues*: one or more queues, separated by the pipe "|" symbol. *(*)*
- *from*: the beginning of the reporting period, in date-time format. *(*)*
- *to*: the end period. *(*)*
- *agent*: the agent code to be used as a filter.
- *form*: the name of the form you want to report on. *(*)*
- *grader*: the grader type.
- *block*: One or more data blocks that you need to access.

Parameters marked with an asterisk are mandatory.

Allowed data blocks: QualAssDO

Quality Assessment information related to QA Forms.

Method	Description
OverallAverageFormReport	Aggregated information for the overall specified form (scoring and not scoring questions included)
FormSummary	Aggregated information for the specified form (only scoring questions)
ScoringItemsFormSummary	Aggregated information for the specified form (only scoring questions, same as FormSummary)
NonScoringItemsFormSummary	Aggregated information for the specified form (only non scoring questions)

Entering a QA form: QmQaGrading

This method lets you fill a QA form through an API call. It replies with the same raw information reported by the *QmQaFormReport* method and can replace it if QA parameters are empty when calling.

The report counts the aggregated QA statistics on calls with timestamp included in the date range specified.

Method	QmQaGrading
Auth required?	Yes - user must hold the key ROBOT and the key QA_TRACK
XML-RPC method	QM.qaformgrading
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmQaGrading/jsonStatsApi.do
      ?calldate=2014-03-30.15:19:00
      &margin=3600
      &id=475263.1
      &form=MyForm
      &queues=q1
      &item_CLE=73
      &item_HEL=56
      &comment=Comment1
      &comment=Comment2
      &block=QualAssFormDO.FormStructure
      &block=QualAssFormDO.SectionValues
      &block=QualAssFormDO.Comments"
```

Parameters

- *queues*: the set of queues that must be included in the analysis. They must be separated by a "|" symbol if more than one queue is passed. The queue name is the internal Asterisk queue name. *(*)*
- *calldate*: the beginning of the search period for the call. This should be usually a few seconds or minutes before the call was started. *(*)*
- *margin*: This is at least the number of seconds the call was in the waiting status (or the complete call time or a suitable number that comfortably contains the call like, for example, 3600). *(*)*
- *form*: The form name that you need to fill-in. *(*)*
- *id*: The unique identifier for the call to be graded. *(*)*
- *grader*: The grader type used to filter out graded forms.
- *item_*: The list of QA items score supplied as an hash of "item_" prefix (see above). The list should contain all specific form items codes and their relative score. In order to specify N/A values for not mandatory items, an empty string should be specified. If the list is left empty, no QA score will be filled into the form *(*)*

- *comment*: A list of the notes to be filled in the form. Each note must be supplied as a String. If the list is empty, no new comments will be added to the form.
- *block*: a set of blocks to be returned. Possible values are the ones defined in *QmQaFormReport*.

Parameters marked with an asterisk are mandatory.

Finding calls to grade: QmQaCallsToGrade

Runs a Grading transaction.

Method	QmQaCallsToGrade
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.qacallstograde
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmQaCallsToGrade/jsonStatsApi.do
      ?from=2010-01-01.00:00:00
      &to=2014-06-30.00:00:00
      &form=MyForm
      &queues=q1|q2
      &block=QAGradingDO.qagExtendedProposals
      &k_outcome_KN_min=100
      &k_outcome_KN_num=
      &k_agroup_Default_min=1"
```

Parameters

- *queues*: the set of queues that must be included in the analysis. They must be separated by a "|" symbol if more than one queue is passed. The queue name is the internal Asterisk queue name. *(*)*
- *from*: a start date. *(*)*
- *to*: an end date. *(*)*
- *form*: the name of the form. *(*)*
- *agent*: an optional agent code to be used as a filter.
- *k_*: a hash of constrainsts used to find calls (see below).
- *block*: one or more response blocks.

Parameters marked with an asterisk are mandatory.

Understanding constraints

Constraints are a set of *key, value* pairs used by the engine to filter out the calls to be graded. The constraint list should be defined with the proper syntax in order to be correctly interpreted by QueueMetrics.

There are two types of constraints: the percentage values and the absolute values. They should be respectively specified through the suffixes "min" or "num".

The constraints are related to different categories:

- Individual agents: specified through the key *AXG* (like, for example: AGX_min or AGX_num)
- All calls: specified through the key *AC* (like, for example: AC_min or AC_num)
- Outcome code: specified through the key *outcome* followed by the outcome code and separated by an underscore character (like, for example: outcome_KN_num or outcome_KN_min)
- Agent group: specified through the key *agroup* followed by the agent group name and separated by an underscore character (like, for example: agroup_Default_min or agroup_Default_num)

Data Blocks: QAGradingDO

Quality Assessment information related to calls to be graded.

Method	Description
qagExtendedProposals	The set of calls to be graded and related information

Tasks

Tasks have two concepts that you have to keep in mind when accessing them through the API:

- Validity: a task can have a validity period, that might be current or future. A task with a futiure validity will "mature" when it enters the validity period.
- The Task Process Field: This is an optional identifier defined as *ProcessFamily/ProcessId* to be associated to the task. Either ProcessFamily and/or ProcessId might be empty. It is used generally to link tasks to external processes, e.g. the process and ID of an external system that loads tasks for users.

Live data: QmAddNoteTask

Adds a note task for a specific user.

Method	QmAddNoteTask
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.tskAddNote
Available since	14.06

See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmAddNoteTask/jsonStatsApi.do?
      recipient=Agent/101
      &valid_from=2010-01-01.00:00:00
      &message=Hello+World
      &notes=notes+here"
```

Parameters

- *recipient*: the login of the user receiving the task. *(*)*
- *valid_from*: begin of validity (in long date format). See above.
- *valid_to*: end of validity.
- *process*: usually in the form "ProcessFamily/ProcessID". See above.
- *message*: the message associated with the task. *(*)*
- *notes*: an optional textual note.

Parameters marked with an asterisk are mandatory.

Live data: QmAddTrainingTask

Adds a training task for a specific user.

Method	QmAddTrainingTask
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.tskAddTraining
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmAddTrainingTask/jsonStatsApi.do?
      recipient=Agent/101
      &valid_from=2010-01-01.00:00:00
      &message=Hello+World
      &notes=notes+here
      &training_title=QM+Website
      &training_url=http://queuemetrics.com"
```

Parameters

- *recipient*: the login of the user receiving the task. *(*)*
- *valid_from*: begin of validity (in long date format). See above.
- *valid_to*: end of validity.
- *process*: usually in the form "ProcessFamily/ProcessID". See above.
- *message*: the message associated with the task. *(*)*
- *notes*: an optional note.
- *training_title*: a title for this trainig task. *(*)*
- *training_url*: an URL for this training task. *(*)*
- *training_id*: an optional ID for this training task.

Parameters marked with an asterisk are mandatory.

Live data: QmAddMeetingTask

Adds a meeting task for a specific user.

Method	QmAddMeetingTask
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.tskAddMeeting
Available since	14.06
See also	

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmAddMeetingTask/jsonStatsApi.do?
      recipient=Agent/101
      &valid_from=2010-01-01.00:00:00
      &title=Hello
      &message=Hello+World
      &notes=notes+here
      &date=2014-05-09.10:30:00
      &duration=300"
```

Parameters

- *recipient*: the login of the user receiving the task. *(*)*
- *valid_from*: begin of validity (in long date format). See above.
- *valid_to*: end of validity.
- *process*: usually in the form "ProcessFamily/ProcessID". See above.

- *message*: the message associated with the task. *(*)*
- *notes*: an optional note.
- *date*: a date and time for the meeting. *(*)*
- *duration*: the duration of the meeting, in seconds. *(*)*

Parameters marked with an asterisk are mandatory.

PBX Interactions

Triggering PBX actions

This method is used to remotely trigger actions that are performed by the PBX(s) connected to QueueMetrics. By this way an external robot can login/out agents, perform pause/unpause actions and other PBX related operations normally triggered by QueueMetris through the agent's realtime and the realtime view.

Method	qm_jsonsvc_do_pbxactions.do
Auth required?	Yes - user must hold the key ROBOT
XML-RPC method	QM.
Available since	15.02.5
See also	https://github.com/Loway/OpenQueueMetricsAddOns

These interactions may raise an error if they cannot be performed (e.g. wrong AMI port configured on QueueMetrics) but if the request can be queued to an Asterisk server they are considered okay whether they succeed or not a the Asterisk level.

Each operation has its own set of parameters that should be present in order to generate a valid action. A set of optional parameters can be added; these optional parameters (specified by an optional array of string) are set as a channel variable by the dialplan (optional parameters are not set as channel variables anymore, since 16.10.13). A PHP script sample is provided through github (see the table above). The script aims to provide an easy startup for developers needs to interact with PBX actions in QueueMetrics.

Parameters

- *action* : either login, logout, join, remove, pause, unpause, calloutcome, customdial, sendtext, softhangup, transfer, inboundmonitor, outboundmonitor.
- *queues* : array of queues. Needed by join, remove, customdial, actions.
- *extension* : agent extension. Needed by all actions except for logout.
- *server* : If empty the default QueueMetrics server is used, otherwise asterisk server identifier in cluster mode. Needed by all actions.

- *agent* : Agent code. Nededed by login, logout, join, remove, pause, unpause, softhangup, transfer, inboundmonitor, outboundmonitor.
- *pause* : Pause code. Needed by pause action.
- *callid* : Asterisk call Id identifying a specific live call. Needed by calloutcome, softhangup, transfer actions.
- *outcome* : Outcome code. Neded by the calloutcome action.
- *targetext* : Target extension. Needed by customdial, sendtext, transfer, inboundmonitor, outboundmonitor.
- *message* : ASCII message to be sent through the sendtext actions (only available for Asterisk 10+).
- *optValues* : A set of optional values that are propagated by QueueMetrics to the dialplan as a custom channel variables. The array shoudl be populated with key/value set where key and values should be appended to the array as a separate line. For further information please see the PHP sample script provided.

Following there's a list of example calls for each of the available functions:

Join

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"join", "extension":"", "queues": ["300","301"], "agent": "Agent/200",
"server": ""}' "http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Remove

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"remove", "extension":"", "queues": ["300","301"], "agent": "Agent/200",
"server": ""}' "http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Pause

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"pause", "extension":"", "pause": "10", "agent": "Agent/200", "server":
""}' "http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Unpause

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"unpause", "extension":"", "agent": "Agent/200", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Custom Dial

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"customdial", "extension":"200","targetext":"201","queues":"301", "server":
""}' "http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Call Outcome

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"calloutcome", "callid":"1489669688.208", "outcome": "a", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Add Feature

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"addfeature", "callid":"1489669688.208", "outcome": "fc01", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Remove Feature

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"remfeature", "callid":"1489669688.208", "outcome": "fc01", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Soft Hangup

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"softhangup", "extension":"200", "agent":"Agent/200",
"callid":"1489669688.208", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Transfer

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"transfer", "extension":"200", "targetext":"202",
"callid":"1489670009.216", "agent":"agent/200", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Inbound Monitor

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"inboundmonitor", "extension":"SIP/200", "targetext":"202",
"callid":"1489670410.229", "agent":"agent/200", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Outbound Monitor

```
curl --user robot:robot -H "Content-Type: application/json" -X POST -d
'{"action":"outboundmonitor", "extension":"SIP/200", "targetext":"202",
"callid":"1489670410.229", "agent":"agent/200", "server": ""}'
"http://localhost:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

Action customdial and hot-desking

Generally speaking, to do outbound dialling you need to know the agent's extension you want to ring. Of course, this is more complex when running in hotdesking mode, as agent and extension are totally separate.

To make this easier to run, since QM 20.11.3, the behavior is that:

- if hotdesking mode is enabled,
- and if agent extension passed in the field `extension` starts with `Agent/`,
- and if `Agent/XXX` is logged on with a known extension on the outbound queue

then QM will:

- run a report for required queue for the current real-time visibility interval,
- will find whether `Agent/XXX` is logged on and will get its extension code,
- will use that channel for outbound dialling as the agent's channel.

If the agent is not logged on, or the extension is empty, then the webservice will abort with an error. If the agent is paused, then a call will still be made.

In practice

First, make sure that you have an Asterisk queue you use as a placeholder for outbound. Only agents logged on to this queue will be able to dial out. In the example below, we created a queue "999" and marked it as "outbound" in the queue definition in QM, so it appears with a yellow icon.

We then make sure agent/101 is logged on to an extension in hotdesking (in our case, extension 300).

Upon calling:

```
curl --user robot:robot -H "Content-Type: application/json"
    -X POST -d '{"action":"customdial",
                 "extension":"agent/101",
                 "message":"",
                 "targetext":"5551234",
                 "queues":"999",
                 "server": ""}'
    "http://127.0.0.1:8080/queuemetrics/qm_jsonsvc_do_pbxactions.do"
```

You get a response OK and an outbound call is started on extension 300 and a second leg to number 5551234 is started as soon as the agent answers; on the agent's page, on Realtime and in reports, the call belongs to Agent/101.

If you try calling an agent when they are not logged on, you get the following response:

```
{
  "action" : "",
  ....,
  "resultStatus" : "KO: Agent: 'Agent/102' not logged on"
}
```

And no call is placed.

Action customdial and penalties

When joining on a queue, QueueMetrics is not able to retrieve the proper queue/agent penalty association.

If you need to login agents with their own penalty for the specified queues, the calling script should set and specify `QM_AGENT_PRIONUM` and `QM_AGENT_PRIOLBL` as optional variables like in the PHP example below

```
joinMember("204", "agent/101", array('300','400'), 'trix1',
array("QM_AGENT_PRIONUM"=>"1", "QM_AGENT_PRIOLBL"=>"M"));
```

System administration

Updating the activation key: QmSetActivationKey

This method is used to remotely change the license key of the QueueMetrics instance or to query the current license key.

As this operation is potentially critical, the user sending this request must hold the keys ROBOT and KEYUPDATE. We ship such a user named *keyupdater* in the default QM configuration but it has to be manually enabled. Make sure you change the password as well.

As the system must be restarted after setting the new key so that it is picked up (this is done automatically), the QM server may be unavailable for a few seconds during the restart phase and all current user sessions may be forcibly terminated. It is therefore not advisable to run this command on a busy system with many users logged in.

Method	Qm
Auth required?	Yes - user must hold the key ROBOT and KEYUPDATE
XML-RPC method	QM.

Available since	14.06
See also	

Example

```
curl  --user keyupdater:enableme -i -H "Content-Type: application/json"
      -X GET "http://localhost:8084/queuemetrics/QmSetActivationKey/jsonStatsApi.do
      ?key=1234"
```

Parameters

- *key*: the new activation key. *(*)*

Parameters marked with an asterisk are mandatory.

Response

The custom block "KeyResults" is filled with the following parameters:

- *KEY_status* : NOKEY if no new key is given, otherwise OK or ERROR depending on the success of the operation.
- *KEY_plexId* : The server identifier.
- *KEY_message* : A message explaining what went wrong.
- *KEY_current_appl* : The name of the application.
- *KEY_current_user* : The name of the user that the application is licensed to.
- *KEY_current_exp* : The expiration date for the current key.

Please note that when a new key is installed, the current user and expiration date are those of the system on which the key is being installed; you should get the new ones as soon as the system restarts (will usually take between 5 and 20 seconds).

Call Queries

Call queries are expressions, written in a micro-language, that let you filter by call on a report. These filters are applied on the possible data set that is delimited by the time period and the queues specified.

An example query could be:

```
(or (and (CODA_F_calldur_min 10) (CODA_F_calldur_min 20))
    (and (CODA_F_calldur_min 40) (CODA_F_calldur_min 60)))
```

As those expressions can be arbitrarily nested, you can combine multiple clauses to get the exact set of calls you are looking for. In the example above, we are looking for all calls that have a duration between 10 and 20 seconds, or between 40 and 60 seconds.

Valid logical filters are:

- `and`
- `or`
- `nor`: the opposite of `or`, that is, not.
- `=t=`: the value True
- `=f=`: the value False

All logical filters take at least one argument an up to how many you need.

The outermost form of the query must be a logical filter, even if there is only one single form. So `(CODA_F_wait_min 200)` is not a valid filter, but `(and (CODA_F_wait_min 200))` is.

Physical filters, displayed in the table below, accept one or two parameters. They are passed in as strings.

- If the parameter is invalid, the filter is considered invalid and not applied, that is, skipped.
- In general, filters that accept a string value will also accept a regular expression.
- All filters work on the IDs passed by Asterisk, that is, if you have a queue which ID is "300" that is decoded to "Support", the filter will receive the value "300".

All filters you run in QueueMetrics are implemented in terms of a combination of these physical filters.

Physical filter	**Meaning**	**Parameters**
CODA_F_agenteFiltro	The agent code	(s)
CODA_F_outcome	The outcome	(s)
CODA_F_calltags	The call tag	(s)
CODA_F_features	The feature	(s)
CODA_F_callskills	The skill	skill(s), value(i)
CODA_F_atomicQueueFilter	A queue	(s)
CODA_F_server	A server id	(s)
CODA_F_disconnection	A disconnection code	(s)
CODA_F_asteriskid	An UniqueId	(s)
CODA_F_caller	A caller's number	(s)
CODA_F_nrm_caller	A normalized caller's number	(s)
CODA_F_dnis	A DNIS	(s)
CODA_F_ivr	An IVR sequence	(s)
CODA_F_wait_min	Min wait	(i)
CODA_F_calldur_min	Min duration	(i)

Physical filter	Meaning	Parameters
CODA_F_enterpos_min	Min enter position	(i)
CODA_F_attempts_min	Min attempts	(i)
CODA_F_wait_max	Max wait	(i)
CODA_F_calldur_max	Max call duration	(i)
CODA_F_enterpos_max	Max enter position	(i)
CODA_F_attempts_max	Max attempts	(i)
CODA_F_noncont_days	Days	A string with the days of the week (s)
CODA_F_noncont_r1_from	Time 1 from	A time HH:MM or HH:MM:SS (s)
CODA_F_noncont_r1_to	Time 1 to	A time HH:MM or HH:MM:SS (s)
CODA_F_shortcall_wait	Min wait	(i)
CODA_F_shortcall_talk		(i)
CODA_F_shortcall_attempt		(i)
CODA_F_variables	Variable	variable (s) value (i)

In the parameters column, *(s)* is a string parameter, while *(i)* an integer one.

Example

```
curl  --user robot:robot -i -H "Content-Type: application/json"
      -X GET "http://localhost:8080/qm/QmStats/jsonStatsApi.do?
      queues=9003%7C93229004%7C9001%7C9002%7C9007%7C9008%7C9005%7C9104%7C9006
      &from=2000-01-01.00:00:00
      &to=2020-01-01.00:00:00
      &block=OkDO.RiassAllCalls
      &block=OkDO.AgentsOnQueue
      &query=%28and%20%28CODA_F_wait_min%2020%29%29"
```

This query runs a report on queues `9003|93229004|9001|9002|9007|9008|9005|9104|9006` with the filter `(and (CODA_F_wait_min 200))`, that is, returns all calls that waited more than 200 seconds. Note how the parameters have been URL-encoded.

QmPushCfgService

This service allows the user to push an existing QueueMetrics configuration on to the current QM configuration.

A QueueMetrics configuration can be easily represented using a JSON file, by posting the file to the service (i.e. by using a curl request), QM will apply the configuration to the system.

This service makes use of the Synchronizer service of QueueMetrics. From the "Explore System Parameters" page in queuemetrics, various modes can be selected for the Synchronizer service.

By selecting the default mode in the "Explore System Parameters" page, the QmPushCfgService service's behaviour will change.

The different selectable modes are a list of combinations of the following four permissions:

- Permission to create objects (agents, queues, pbx)
- Permission to delete objects (agents, queues, pbx)
- Permission to update objects (agents, queues, pbx)
- Permission to create users

When creating users, the user's passwords will be dictated by the "user_password" field of the agents object in the JSON configuration file. This is because the user creation function will create matching users for each created agent, if the correct permissions are given by selecting the appropriate mode.

If an agent definition in the configuration file has an empty or missing user_password field, the given password will be dicatated by the synchronizer.default_password parameter. If this parameter is not set, thew password will be generated randomly as a string of 40 characters.

This is an example of a valid curl request:

```
curl  --user robot:robot -i -H  -X POST
      "http://localhost:8080/QueueMetrics/QmPushCfgService/jsonStatsApi.do?"
      --data-urlencode data@newcfg.json
```

This is an example of a valid JSON configuration file:

```
{
  "version": "1.0.0",
  "requestor": "FreePbx",
  "requestor_version": "13.xx.yy",
  "tenants": [
    {
      "tenant_id": "",
      "agents": [
        {
```

```
        "agent_id": "Agent/101",
        "agent_name": "Mike Boom",
        "agent_aliases": [],
        "agent_extension": "",
        "user_password": "999"
      },
      {
        "agent_id": "Agent/102",
        "agent_name": "John Doe",
        "agent_aliases": [
          "xyx",
          "nuovo"
        ],
        "agent_extension": "103",
        "user_password": "999"
      }
    ],
    "queues": [
      {
        "queue_id": "300",
        "queue_name": "Sales (300)",
        "queue_composition": "300",
        "queue_wrapup": 0,
        "queue_inbound": true,
        "known_agents": [
          {
            "agent_id": "Agent/101",
            "agent_penalty": 1,
            "agent_dynamic": false
          },
          {
            "agent_id": "Agent/102",
            "agent_penalty": 0,
            "agent_dynamic": false
          }
        ]
      },
      {
        "queue_id": "3000",
        "queue_name": "Sales (3000)",
        "queue_composition": "3000",
        "queue_wrapup": 0,
        "queue_inbound": true,
        "known_agents": [
          {
            "agent_id": "Agent/101",
            "agent_penalty": 1,
            "agent_dynamic": false
          },
          {
            "agent_id": "Agent/102",
```

```
              "agent_penalty": 0,
              "agent_dynamic": false
            }
          ]
        }
      ]
    }
  ]
}
```

Uploading queue_log data to QueueMetrics or QueueMetrics Live

Sending data to a QueueMetrics Live instance is a matter of uploading frequently queue_log data to an HTTP/HTTPS webservice. While this is usually performed by the `uniloader` tool, it can also be implemented as needed, as described below.

The idea is that you have a set of credentials (user, password and token) that identifies all `queue_log` events coming from a specific PBX. We expect only one writer to be active at the same time for a given PBX.

The way this works is:

- You get the current **High-Water Mark** (HWM), that is the last point in time for which we have data
- You upload all data which time-stamp is equals or newer than the HWM. QueueMetrics obeys an at-least-once semantics, so while it is harmless to upload the same data multiple times, it is also useless.
- If the upload succeeds, you get the next batch of data and upload it as well. As you are the only writer, there is no need to go check the HWM again.
- If the upload does not succeed, wait a few seconds and then restart.

We suggest keeping the latency to a minimum, by checking if there is new queue_log data and pushing data immediately, in order to power real-time wallbords; this said, a latency of a few seconds is acceptable for most real-life cases. Still, there should be only one writer per PBX, and it should always wait for the resoult of a previous operation before attempting one again.

Data should be uploaded in order it was created, from older to newer. Many parts of QueueMetrics expect this to be true in order to cache data agressively, and so it might not display events uploaded in the wrong order. In this case, you have to invalidate caches (or restart) for data to be picked up.

Connection to an on-prem QueueMetrics system

You can use this interface on any modern QueueMetrics system. While it is pre-configured on QueueMetrics Live, you can create the an upload user on your on-prem system:

- Your user should have both keys `WQLOADER`, `USER` and `ROBOT`. We suggest adding it to the class ROBOT.
- If you do not have a cluster, leave the token blank. It will use the default data partition.
- If you have a cluster, the token you need to use is the name of the machine in the cluster, as defined in the property `cluster.servers`. The partition to use for server `aleph` will then be taken from `cluster.aleph.queuelog=sql:P001`

If you run a cluster, it is better to use separate users for each cluster member; so it will be easier to spot who is doing what and each of them will have a separate password.

General consideration on web services

Web services share a common format and are based on JSON. To access webservices, you need:

- The base URL
- A login (usually "webqloader")
- A password
- A token / partition ID (usually blank for single Asterisk systems, must be set when uploading data from a cluster of PBXs)

Service is located relative to the base QM URL; so if QM's main URL is at `https://my.queuemetrics-live/somecustomer`, the web services will appear at URL `https://my.queuemetrics-live/somecustomer/jsonQLoaderApi.do`.

They all expect a POST call, with basic HTTP auth, and the payload expressed as a JSON structure in the single parameter named `COMMANDSTRING`. The call might redirect during the reposnse pahse, and you are supposed to follow redirects until you get a response.

The response has the format:

```
{"version":"1.0",
 "token":"",
 "resultStatus":"KO: Http auth missing or failed",
 "result":"",
 "name":"Dummy Invalid"}
```

And you need to check the `resultStatus` to be *OK*.

Retries

On any HTTP status code other than 200, or `resultStatus` other than *OK*, you must consider it an error and retry.

We suggest using an exponential backoff strategy, starting from one second up to 30 minutes. It is useless to keep getting errors every second.

The HWM service

This service checks the highest time-stamp present on the partition, as identified by the token.

Example payload:

```
{"commandId": "checkHWM",
 "version": "1.0",
 "token": ""}
```

Example cURL call:

```
$ curl -H "Accept: application/json" \
   -X POST -u webqloader:77845487 \
   -d 'COMMANDSTRING={"commandId":"checkHWM", "version":"1.0", "token":""}' \
   https://my.queuemetrics-live.com/sometest/jsonQLoaderApi.do
```

You read the `result` field in, and it either contains a timestamp or *null* for an empty partition. For example, an empty response looks like:

```
{"commandId":"checkHWM",
 "version":"1.0",
 "token":"",
 "resultStatus":"OK",
 "result":null,
 "name":"Check High WaterMark"}
```

The batch upload service

The batch upload service lets you upload multiple rows with the same call. We suggest batching up to no more than 250 rows. Of course you can upload any lesser number, from one onwards!

Example payload format:

```
{"commandId": "insertAll",
 "version":   "1.0",
 "token":     "",
 "rows":      [
        {"timeId":     123456,
         "callId":     "123.123",
         "queue":      "aaa",
         "agent":      "NONE",
         "verb":       "ENTERQUEUE",
         "parameters": ["", "", "", "", ""]},
        {"timeId":     123457,
         "callId":     "123.123",
         "queue":      "aaa",
         "agent":      "NONE",
         "verb":       "CONNECT",
         "parameters": ["1", "1234.56", "", "", ""]}
}]
```

Will upload the two queue_log rows:

```
123456|123.123|aaa|NONE|ENTERQUEUE||||
123457|123.123|aaa|NONE|CONNECT|1|1234.56||
```

The service actively de-duplicates data, so sending the same data multiple times will not cause trouble. Still, uploading large data sets still takes up precious resources, so it is better not to upload data that you know is already present.

Appendix A1: A short list of REST/JSON libraries

The following list is by no means exhaustive of all available implementations. In most cases REST/JSON implementation are embedded in the language itself and are logically split between a network library that will take care of the HTTP request and a JSON library that will unmarshal the response.

Java

The excellent Jackson library will help you parse and create JSON - see https://github.com/FasterXML/jackson - while the embedded class *java.net.URLConnection* will take care of the connection.

JavaScript / NodeJS

The *http* module - especially `http.request` - will take care of the connection while *JSON.parse* will decode the output.

Python

The *json* and *urllib2* libraries should be immediately available.

Go

Can be done natively leveraging the *encoding/json* and *net/http* packages.

Ruby

The modules *json* and *open-uri* are available in the standard library.

C# / .Net

Can be done natively - see example at http://msdn.microsoft.com/en-us/library/hh674188.aspx

Perl

You can use the *JSON* module and *LWP::UserAgent* for accessing the remote QM instance.

PHP

You can usually read a remote URL with *php_curl* and retry the JSON structure with *json_decode* . Here's an example of a GET query with php:

```
----
$url = "http://queuemetrics-server:8080/queuemetrics/agent/73/jsonEditorApi.do
```

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "GET");
curl_setopt($ch, CURLOPT_USERPWD, "username:password");
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
```

```
$output = curl_exec($ch);
curl_close($ch);
```

```
print_r(json_decode($output, true));
----
```

Appendix A2: QueueMetrics data blocks

Data blocks come in different flavors, based on what you need to be doing:

- `Raw` data blocks contain queues and agents as their internal ID and not their name, dates as timestamps and durations as seconds. This makes them way easier to be used by an external service for processing, though they will be hard for a human to read is you add them to a QM report. See Reading "raw" blocks
- Data blocks named `Afp`, `ButtonExportCalls` and `ExtSourceDo` are used to draw items on pages or pop-ups and contain no actual data. They are not meant to be used by external services.
- Data blocks `RealTimeDO.RtLive...` are a deprecated feature that polls Asterisk through the AMI interface.

Available blocks for QmStats

Report blocks available in QueueMetrics 19.10:

- AgentsDO.ReportAgents (short code: `AG01`)
- AgentsDO.SessionPauseDur (short code: `AG02`)
- AgentsDO.AgentAvail (short code: `AG03`)
- AgentsDO.AnsCallsQueues (short code: `AG04`)
- AgentsDO.AnsCallsCG (short code: `AG05`)
- AgentsDO.AnsCallsLocation (short code: `AG06`)
- AgentsDO.AnsCallsSG (short code: `AG07`)
- AgentsDO.PerformanceAcdGroups (short code: `AG08`)
- AgentsDO.AgentOccupancy (short code: `AG09`)
- AgentsDO.AgentSessionTimeByHour (short code: `AG10`)
- AgentsDO.AgentPayableTimeByHour (short code: `AG11`)
- AgentsDO.AgentBillableTimeByHour (short code: `AG12`)
- AgentsDO.AgentSessionsView (short code: `AG13`)
- AgentsDO.QueueSessionsView (short code: `AG14`)
- AgentsDO.AgentByHour (short code: `AG15`)
- AgentsDO.TagSessionsView (short code: `AG16`)
- AgentsDO.SessionByQueueTagReport (short code: `AG17`)
- AgentsDO.ProgAgPerf (short code: `AG18`)
- AgentsDO.DetailedPauses (short code: `AG19`)
- AgentsDO.AgentTTPerHour (short code: `AG20`)
- AgentsDO.OutboundProductivity (short code: `AG21`)

- AreaAnDO.Setup (short code: `AC01`)
- AreaAnDO.CallsOK (short code: `AC02`)
- AreaAnDO.CallsKO (short code: `AC03`)
- AreaAnDO.FrequentAreaCodes (short code: `AC04`)
- CallDistrDO.AnsDistrPerDay (short code: `DD01`)
- CallDistrDO.AnsWaitPerDay (short code: `DD02`)
- CallDistrDO.UnansWaitPerDay (short code: `DD03`)
- CallDistrDO.SalesPerDay (short code: `DD04`)
- CallDistrDO.StaffPerDay (short code: `DD05`)
- CallDistrDO.QPosPerDay (short code: `DD06`)
- CallDistrDO.InclSlaPerDay (short code: `DD07`)
- CallDistrDO.TrafficAnPerDay (short code: `DD08`)
- CallDistrDO.SkillsPerDay (short code: `DD09`)
- CallDistrDO.DetailSummaryPerDay (short code: `DD10`)
- CallDistrDO.AnsDistrPerHr (short code: `DH01`)
- CallDistrDO.AnsWaitPerHr (short code: `DH02`)
- CallDistrDO.UnansWaitPerHr (short code: `DH03`)
- CallDistrDO.SalesPerHr (short code: `DH04`)
- CallDistrDO.StaffPerHr (short code: `DH05`)
- CallDistrDO.QPosPerHr (short code: `DH06`)
- CallDistrDO.InclSlaPerHr (short code: `DH07`)
- CallDistrDO.TrafficAnPerHr (short code: `DH08`)
- CallDistrDO.SkillsPerHr (short code: `DH09`)
- CallDistrDO.DetailSummaryPerHr (short code: `DH10`)
- CallDistrDO.AnsDistrPerDOW (short code: `DW01`)
- CallDistrDO.AnsWaitPerDOW (short code: `DW02`)
- CallDistrDO.UnansWaitPerDOW (short code: `DW03`)
- CallDistrDO.SalesPerDOW (short code: `DW04`)
- CallDistrDO.StaffPerDOW (short code: `DW05`)
- CallDistrDO.QPosPerDOW (short code: `DW06`)
- CallDistrDO.InclSlaPerDOW (short code: `DW07`)
- CallDistrDO.TrafficAnPerDOW (short code: `DW08`)
- CallDistrDO.SkillsPerDOW (short code: `DW09`)
- CallDistrDO.DetailSummaryPerDow (short code: `DW10`)
- DetailsDO.AgentSessions (short code: `AD01`)

- DetailsDO.AgentPauses (short code: `AD02`)
- DetailsDO.AgentSessionsRaw (short code: `RW01`)
- DetailsDO.CallsKO (short code: `UD01`)
- DetailsDO.AfpCallsKO (short code: `UD02`)
- DetailsDO.CallsKoRaw (short code: `RW02`)
- DetailsDO.CallsOK (short code: `OD01`)
- DetailsDO.AfpCallsOK (short code: `OD03`)
- DetailsDO.AfpCallsIVR (short code: `OD04`)
- DetailsDO.CallsIVR (short code: `OD05`)
- DetailsDO.CallsOkRaw (short code: `RW03`)
- DetailsDO.ButtonExportCalls (short code: `OD02`)
- DetailsDO.ExecutiveSummary (short code: `OD06`)
- DistrDO.ReportAcd (short code: `AT01`)
- DistrDO.AcdByQueue (short code: `AT02`)
- DistrDO.AcdByTerminals (short code: `AT03`)
- KoDO.ReportKoAll (short code: `UN01`)
- KoDO.ReportKoFully (short code: `UN02`)
- KoDO.DiscCauses (short code: `UN03`)
- KoDO.UnansByQueue (short code: `UN04`)
- KoDO.OutboundKo (short code: `UN05`)
- KoDO.UnansByLen (short code: `UN06`)
- KoDO.InclusiveSLA (short code: `UN07`)
- KoDO.ReportKoKeyPress (short code: `UN08`)
- KoDO.StintsKo (short code: `UN09`)
- KoDO.StintsOkKo (short code: `UN10`)
- KoDO.QPosKo (short code: `UN11`)
- KoDO.QPosOkKo (short code: `UN12`)
- KoDO.IvrKo (short code: `UN13`)
- KoDO.IvrOkKo (short code: `UN14`)
- KoDO.DnisKo (short code: `UN15`)
- KoDO.DnisOkKo (short code: `UN16`)
- KoDO.OverviewOkKo (short code: `UN17`)
- KoDO.InclusiveAnswSLA (short code: `UN18`)
- KoDO.SkillsKo (short code: `UN19`)
- KoDO.SkillsOkKo (short code: `UN20`)

- OkDO.RiassAllCalls (short code: `OK01`)
- OkDO.RiassFullyWithin (short code: `OK02`)
- OkDO.AgentsOnQueue (short code: `OK03`)
- OkDO.ServiceLevelAgreement (short code: `OK04`)
- OkDO.DisconnectionCauses (short code: `OK05`)
- OkDO.Transfers (short code: `OK06`)
- OkDO.AnsweredcallsByQueue (short code: `OK07`)
- OkDO.AnsweredcallsByDirection (short code: `OK08`)
- OkDO.StintsOk (short code: `OK09`)
- OkDO.QPosOk (short code: `OK10`)
- OkDO.IvrOk (short code: `OK11`)
- OkDO.DnisOk (short code: `OK12`)
- OkDO.MOHOk (short code: `OK13`)
- OkDO.HDRRpt (short code: `OK14`)
- OkDO.SkillsOk (short code: `OK15`)
- OutcomesDO.GeneralRep (short code: `OU01`)
- OutcomesDO.CallResByOutcome (short code: `OU02`)
- OutcomesDO.ActivBillable (short code: `OU03`)
- OutcomesDO.ActivNotBillable (short code: `OU04`)
- OutcomesDO.AgentReportDetailed (short code: `OU05`)
- OutcomesDO.AgentOutcomes (short code: `OU06`)
- CallTagDO.CallResByTag (short code: `OU07`)
- OutcomesDO.CallResByFeature (short code: `OU08`)
- IvrDO.IvrReport (short code: `IV01`)
- IvrDO.IvrTiming (short code: `IV02`)
- IvrDO.IvrGoals (short code: `IV03`)
- ExtSourceDO.HTML (short code: `XS01`)
- ExtSourceDO.XMLRPC (short code: `XS02`)
- ExtSourceDO.JSON (short code: `XS03`)
- FcrDO.RecallsOk (short code: `FC01`)
- FcrDO.RecallsKo (short code: `FC02`)
- FcrDO.RecallsAll (short code: `FC03`)
- FcrDO.ClidsOk (short code: `FC04`)
- FcrDO.ClidsKo (short code: `FC05`)
- FcrDO.ClidsAll (short code: `FC06`)

- FcrDO.ClustersOk (short code: `FC07`)
- FcrDO.ClustersKo (short code: `FC08`)
- FcrDO.ClustersAll (short code: `FC09`)
- FcrDO.RetryRate (short code: `FC10`)
- FcrDO.MostFrequentNumber (short code: `FC11`)

Available blocks for QmRealtime

Real-time blocks available in QueueMetrics 19.10:

- RealTimeDO.RTReportHeader
- RealTimeDO.RTRiassunto
- RealTimeDO.RTCallsBeingProc
- RealTimeDO.RTAgentsLoggedIn
- RealTimeDO.WallRiassunto
- RealTimeDO.WallCallsBeingProc
- RealTimeDO.VisitorCallsProc
- RealTimeDO.VisitorTodaysOk
- RealTimeDO.VisitorTodaysKo
- RealTimeDO.RtLiveQueues
- RealTimeDO.RtLiveCalls
- RealTimeDO.RtLiveAgents
- RealTimeDO.RtLIveStatus
- RealTimeDO.RtAgentsRaw
- RealTimeDO.RtCallsRaw
- RealTimeDO.RtAggrByQueueView
- RealTimeDO.RtAggrByTagView
- RealTimeDO.AgentAndOutcomeView