
Accessing QueueMetrics through its XML-RPC interface

Loway

Revision \$Revision: 1.24 \$ -
covers QueueMetrics 12.01

Revision History
\$Date: 2012/01/19 10:16:11 \$

L

Table of Contents

Document contents	1
Revision history	1
What is XML RPC?	2
Which functions does QueueMetrics export as XML-RPC?	2
Example: accessing QueueMetrics from Python	3
Example: Accessing QueueMetrics from Java	3
Example: Accessing QueueMetrics from PHP	4
Example: Accessing QueueMetrics from JavaScript	6
Understanding call parameters	7
The method QM.stats	7
The method QM.realtime	7
The method QM.qareport	8
The method QM.qaformreport	8
The method QM.qaformssummary	9
The method QM.qaformgrading	9
The method QM.qacallstograde	10
The method QM.auth	10
The method QM.authenticate	10
The method QM.findAudio	11
The method QM.insertRecordTag	12
The method QM.tskAddNote	12
The method QM.tskAddTraining	13
The method QM.tskAddMeeting	13
The method QM.setActivationKey	14
The method QM.broadcastMessage	14
Understanding results	15
Using an external auth source for QueueMetrics	15
Log on procedure	15
XML-RPC call parameters	16
A. Response block names	17
Response block names for QM.stats	17
Response block names for QM.realtime	17
DataObject: RealtimeDO	17
Response block names for QM.qareport	17
DataObject: QualAssDO	17
Response block names for QM.qaformreport and QM.qaformgrading	18
DataObject: QualAssFormDO	18
Response block names for QM.qaformssummary	18
DataObject: QualAssDO	18
Response block names for QM.qacallstograde	18
DataObject: QAGradingDO	18
Response block names for QM.auth	18
Response block names for QM.findAudio	19
Response block names for QM.insertRecordTag	19
Appendix II: A short list of XML-RPC libraries	19

Document contents

This document details how to access and use the XML-RPC 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

- Nov 13, 2006: First draft

- March 22, 2007: Added PHP example
- April 13, 2007: "raw" blocks available
- May 11, 2007: real-time blocks and auth server
- Jun 11, 2007: multi-stint calls
- Nov 11, 2007: enter-queue position and schedule adherence
- Nov 10, 2008: Added Outcomes data blocks and external XML-RPC Authentication description
- Jan 14, 2009: Added support for JavaScript
- Feb 11, 2009: Moved to AsciiDoc format for ease of updating
- Feb 17, 2009: Added QM.findAudio methods
- Aug 07, 2009: Added QM.qareport and QM.qaformreport reports
- Aug 10, 2009: Added AgentsDO.AnsCallsCG for "Answered calls by custom group" reporting
- Dec 2, 2009: Added QM.qaformssummary report
- Jul 8, 2010: Added new blocks "Call Overview", "Traffic analysis", "Agent performance"
- Jul 21, 2010: Added QM.qaformgrading method
- Aug 20, 2010: Added new block "Inclusive Answered SLA"
- Oct 1, 2010: Referenced QM manual for possible data blocks.
- Dec 2, 2010: Added new QA related optional parameter, Added new QM.tskAddNote and QM.qacallstograde methods
- May 16, 2011: New method setActivationKey()
- Oct 6, 2011: Added new QM.tskAddTraining methods. Modified QM.tskAddNote in order to receive the processField
- Oct 7, 2011: Added new QM.broadcastMessage method.
- Oct 24, 2011: Added new QM.tskAddMeeting method.
- Nov 14, 2011: New method QM.authenticate()
- Nov 23, 2011: New method QM.insertRecordTag()

What is XML RPC?

Wikipedia defines XML-RPC as:

XML-RPC is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. It is a very simple protocol, defining only a handful of data types and commands, and the entire description can be printed on two pages of paper. This is in stark contrast to most RPC systems, where the standards documents often run into the thousands of pages and require considerable software support in order to be used.

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

Which functions does QueueMetrics export as XML-RPC?

QueueMetrics exports the full results of an analysis in XML-RPC, so you can access whatever information you feel you may need. 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 result 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

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

The XML-RPC interface is available in all version of QueueMetrics starting from version 1.3.1.

Example: accessing QueueMetrics from Python

In this example we'll see how easy it is to access QueueMetrics from a scripted language like Python. You can enter the following statements interactively using a Python IDE like IDLE, or make them a part of a larger program.

The following code connects to the XML-RPC port of a QueueMetrics instance running at `http://qmserver:8080/qm130` and asks for a couple of tables, namely the distribution of answered calls per day and the Disconnection causes.

```
> import xmlrpclib
> server_url = 'http://qmserver:8080/qm130/xmlrpc.do';
> server = xmlrpclib.Server(server_url);
> res = server.QM.stats( "queue-dps", "robot", "robot", "", "", "2005-10-10.10:23:12", "2007-10-10.10:23:10" )

> res.keys()
['CallDistrDO.AnsDistrPerDay', 'result', 'KoDO.DiscCauses']

> res['result']
[['Status', 'OK'], ['Description', ''], ['Time elapsed (ms)', 3008], ['QM Version', '1.3.1']]

> res['result'][2][1]
3008
```

As you can see, it only takes four lines of Python code to connect to QueueMetrics and get all the results back!

Example: Accessing QueueMetrics from Java

This is an example functionally equivalent to the one above in Python, but it's written in Java using the Redstone XML-RPC client library.

```
import java.io.IOException;
import java.util.HashMap;
import java.net.URL;
import redstone.xmlrpc.XmlRpcClient;
import java.util.ArrayList;

public class xmlRpcTestClient {
    public void perform() {
        String stUrl = "http://server:8080/qm130/xmlrpc.do";
        System.setProperty(
            "org.xml.sax.driver",
            "org.apache.xerces.parsers.SAXParser"
        );

        try {
            XmlRpcClient client = new XmlRpcClient( stUrl, false );

            ArrayList arRes = new ArrayList();
            arRes.add( "OkDO.AgentsOnQueue" );
            arRes.add( "KoDO.DiscCauses" );
            arRes.add( "KoDO.UnansByQueue" );
            arRes.add( "DetailsDO.CallsKO" );

            Object[] parms = { "queue-dps", "robot", "robot",
                               "", "", "2005-10-10.10:23:12",
                               "2007-10-10.10:23:10",
                               "", arRes };

            Object token = client.invoke( "QM.stats", parms );
            HashMap resp = (HashMap) token;
            System.out.println( "Resp: " + resp );
        } catch ( Exception e ) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        xmlRpcTestClient c = new xmlRpcTestClient();
        try {
            c.perform();
        } catch (Exception e) {
        }
    }
}
```

```

        e.printStackTrace();
    }
}

```

We'll have to explicitly set which XML parser to use; for the rest, the code looks very much alike the Python one.

Example: Accessing QueueMetrics from PHP

This example is based on PHP's XML_RPC class, that is a part of PEAR and so should be preinstalled with any modern PHP installation.

```

<h1>A QueueMetrics XML-RPC client in PHP</h1>
<?
require_once 'XML/RPC.php';

$qm_server = "10.10.3.5"; // the QueueMetrics server address
$qm_port   = "8080";     // the port QueueMetrics is running on
$qm_webapp = "queuemetrics-1.3.3"; // the webapp name for QueueMetrics

// set which response blocks we are looking for
$req_blocks = new XML_RPC_Value(array(
    new XML_RPC_Value("DetailsDO.CallsOK"),
    new XML_RPC_Value("DetailsDO.CallsKO")
), "array");

// general invocation parameters - see the documentation
$params = array(
    new XML_RPC_Value("queue-dps"),
    new XML_RPC_Value("robot"),
    new XML_RPC_Value("robot"),
    new XML_RPC_Value(""),
    new XML_RPC_Value(""),
    new XML_RPC_Value("2007-01-01.10:23:12"),
    new XML_RPC_Value("2007-10-10.10:23:10"),
    new XML_RPC_Value(""),
    $req_blocks
);

$msg = new XML_RPC_Message('QM.stats', $params);
$client = new XML_RPC_Client("/$qm_webapp/xmlrpc.do", $qm_server, $qm_port);
// $client->setDebug(1);
$response = $client->send($msg);
if (!$response) {
    echo 'Communication error: ' . $client->errstr;
    exit;
}
if ($response->faultCode()) {
    echo 'Fault Code: ' . $response->faultCode() . "\n";
    echo 'Fault Reason: ' . $response->faultString() . "\n";
} else {
    $val = $response->value();
    $blocks = XML_RPC_decode($val);

    // now we print out the details....
    printBlock( "result", $blocks );
    printBlock( "DetailsDO.CallsOK", $blocks );
    printBlock( "DetailsDO.CallsKO", $blocks );
}

// output a response block as HTML
function printBlock( $blockname, $blocks ) {
    echo "<h2>Response block: $blockname </h2>";
    echo "<table border=1>";
    $block = $blocks[$blockname];
    for ( $r = 0; $r < sizeof( $block ); $r++ ) {
        echo "<tr>";
        for ( $c = 0; $c < sizeof( $block[$r] ); $c++ ) {
            echo( "<td>" . $block[$r][$c] . "</td>" );
        }
        echo "</tr>\n";
    }
}

```

```

        echo "</table>";
    }
?>

```

In this next example, the PHP code is able to submit QA grading information in order to fill a specific form:

```

<?php
    require_once 'XML/RPC.php';

    $qm_server = "10.10.3.5"; // the QueueMetrics server address
    $qm_port   = "8080";     // the port QueueMetrics is running on
    $qm_webapp = "queuemetrics-1.3.3"; // the webapp name for QueueMetrics

    // set which response blocks we are looking for
    $req_blocks = new XML_RPC_Value(array(
        new XML_RPC_Value("QualAssFormDO.FormStructure"),
        new XML_RPC_Value("QualAssFormDO.SectionValues"),
        new XML_RPC_Value("QualAssFormDO.Comments"),
    ), "array");

    // set notes we want to update
    $comments_block = new XML_RPC_Value(array(
        new XML_RPC_Value("This is a form note added with XMLRPC"),
        new XML_RPC_Value("This is the next form note added with XMLRPC"),
    ), "array");

    // set QA infor scores to be
    // Each item code should be defined.
    // Each item will be associated to a score number
    // or an empty string if N/A (only for Not Mandatory fields)
    $grading_info = new XML_RPC_Value(array(
        'NMYN' => new XML_RPC_Value("", 'string'),
        'NMNUM' => new XML_RPC_Value("20", 'string'),
        'MNUM' => new XML_RPC_Value("60", 'string'),
        'MMUL' => new XML_RPC_Value("100", 'string'),
    ), "struct");

    // general invocation parameters - see the documentation
    $params = array(
        new XML_RPC_Value(""),
        new XML_RPC_Value("robot"),
        new XML_RPC_Value("robot"),
        new XML_RPC_Value("2010-06-16.12:32:00"),
        new XML_RPC_Value("3600"),
        new XML_RPC_Value("Test GUI items"),
        new XML_RPC_Value("1276684326.309"),
        $grading_info,
        $comments_block,
        $req_blocks
    );

    $msg = new XML_RPC_Message('QM.qaformgrading', $params);
    $cli = new XML_RPC_Client("/$qm_webapp/xmlrpc.do", $qm_server, $qm_port);

    // $cli->setDebug(1);
    $resp = $cli->send($msg);

    if (!$resp) {
        echo 'Communication error: ' . $cli->errstr;
        exit;
    }

    if ($resp->faultCode()) {
        echo 'Fault Code: ' . $resp->faultCode() . "\n";
        echo 'Fault Reason: ' . $resp->faultString() . "\n";
    } else {
        $val = $resp->value();
        $blocks = XML_RPC_decode($val);

        // now we print out the details....
        printBlock("result", $blocks);
        printBlock("QualAssFormDO.FormStructure", $blocks);
        printBlock("QualAssFormDO.SectionValues", $blocks);
    }

```

```

        printBlock( "QualAssFormDO.Comments", $blocks );
    }

    // output a response block as HTML
    function printBlock( $blockname, $blocks ) {
        echo "\nResponse block: $blockname \n";
        $block = $blocks[$blockname];

        for ( $r = 0; $r < sizeof( $block ); $r++ ) {
            echo "\n";
            for ( $c = 0; $c < sizeof( $block[$r] ); $c++ ) {
                echo( $block[$r][$c] . "\t" );
            }
        }
    }
}
?>

```

As you can see, it is very similar to the other programming languages, and reading into the results is simply a matter of selecting the correct block and then accessing the data cell by row and column. The added complexity is due to the explicit error condition check and result printout.

Example: Accessing QueueMetrics from JavaScript

In order to access the XML-RPC interface from JavaScript, you need to use an adaptor library. We have been able to successfully use a library called JSON-XML-RPC that can be found at <http://code.google.com/p/json-xml-rpc/> in a file called `rpc.js`

Generally speaking, a JavaScript client can make requests only against the same server that is serving the HTML page, therefore you need to install it on the same server as QueueMetrics, creating a separate webapp.

```

<html>
<head>
<title>javascript_client.html</title>
<script src="rpc.js"></script>
</head>
<body>
<h1>QueueMetrics JavaScript XML-RPC example </h1>
<script>
var server = "/DAILY/xmlrpc.do";

function run() {
    try {
        var service = new rpc.ServiceProxy( server, {
            asynchronous:false,
            protocol: "XML-RPC",
            methods: ["QM.stats", "QM.realtime", "QM.auth"]
        } );
        res = service.QM.stats( "q1", "robot", "robot","", "",
            "2005-10-10.10:23:12", "2009-10-10.10:23:10","",
            [ "KoDO.DiscCauses", "CallDistrDO.AnsDistrPerDay" ]
        );
        document.getElementById("RESULT").innerHTML = plotBlocks(res);
    } catch(e) {
        alert("Error raised: " + e);
    }
}

function plotBlocks( hmBlocks ) {
    s = "";
    for (var i in hmBlocks) {
        s += "<h2>Block: " + i + "</h2>";
        s += plotBlock( hmBlocks[i] );
    }
    return s;
}

function plotBlock( arBlock ) {
    s = "";
    for ( r=0; r < arBlock.length; r++ ) {
        s += "<tr>";
        for ( c = 0; c < arBlock[r].length; c++ ) {
            s += "<td>" + arBlock[r][c] + "</td>";

```

```

    }
    s += "</tr>";
  }
  return "<table border=1>" + s + "</table>";
}
</script>

<input type="button" value="Click me!" onclick="run();" >
<div id="RESULT"></div>

</body>
</html>

```

As you can see, the code is actually very similar to the Python one. The only important difference here is that the names of the methods have to be explicitly declared.

Understanding call parameters

There are three methods exported by the XML-RPC interface, and they are used for three different reasons:

- QM.stats: get the main historical stats
- QM.realtime: get the real time stats (available since QM 1.3.5)
- QM.qareport: get the Quality Assessment statistics (available since QM 1.6.0)
- QM.qaformreport: get some Quality Assessment Forms details (available since QM 1.6.0)
- QM.qaformsummary: get Quality Assessment statistics related to the same form (available since QM 1.6.0)
- QM.qaformgrading: fill a QA form with the specified scores and/or comments (available since QM 2.0.0 \todo)
- QM.auth: use QueueMetrics as an auth server for third party software (available since QM 1.3.5)

The method QM.stats

This is the main method exported, its XML-RPC name being QM.stats. It takes a number of arguments, all of which must be supplied and must have the correct data type. They are:

1. *(String)* Queue name or names: 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.
2. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
3. *(String)* Access password: this must be the clear-text password of the given user name
4. *(String)* Logfile - always leave blank.
5. *(String)* Period - always leave blank.
6. *(String)* Start of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
7. *(String)* End of period. Same format as start of period
8. *(String)* Agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity
9. *(List)* A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

This 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 XML format and the resulting client format.

It is also 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 the other live users.

The method QM.realtime

This method is very similar to QM.stats but it is used to retrieve the real time stats. It must be called with the following parameters:

1. *(String)* Queue name or names: 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.
2. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
3. *(String)* Access password: this must be the clear-text password of the given user name
4. *(String)* Logfile - always leave blank.
5. *(String)* Agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity
6. *(List)* A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

The same suggestions that are given for *QM.stats* apply.



Important

Please note that there is a difference between results produced by the XML RPC real-time 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. Being the list of queues, in the XML RPC call, specified by a list of names instead of a list of queue unique identifiers, it's not possible to correctly identify elementary queues from macro queues having the same name. In this situation the agent list will always be calculated as the sum of all agents associated to all elementary queues composing the macro queue, even if the macro queue has directly assigned agents.

The method QM.qareport

This method is very similar to *QM.stats* but it's used to retrieve Quality Assessment statistics. It must be called with the following parameters:

1. *(String)* Queue name or names: 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.
2. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
3. *(String)* Access password: this must be the clear-text password of the given user name
4. *(String)* Start of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
5. *(String)* End of period. Same format as start of period
6. *(String)* Agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity
7. *(String)* The form name for which you need to have information
8. *(String)* The grader type used to filter out graded forms. It could be any of the following values: "unknown", "agent", "grader", "caller". This parameter is optional and could not be present in the function call
9. *(List)* A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

The same suggestions that are given for *QM.stats* apply.

The method QM.qaformreport

This method is very similar to *QM.stats* but it's used to retrieve raw information about Quality Assessment Forms. It must be called with the following parameters:

1. *(String)* Queue name or names: 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.
2. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
3. *(String)* Access password: this must be the clear-text password of the given user name
4. *(String)* Start of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
5. *(String)* End of period. Same format as start of period
6. *(String)* Agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity
7. *(String)* The form name for which you need to have information

8. (*String*) The grader type used to filter out graded forms. It could be any of the following values: "unknown", "agent", "grader", "caller". This parameter is optional and could not be present in the function call
9. (*List*) A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

The same suggestions that are given for *QM.stats* apply.

The method QM.qaformsummary

This method is very similar to *QM.stats* but it's used to retrieve aggregated information about a specific Quality Assessment Form. It must be called with the following parameters:

1. (*String*) Queue name or names: 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.
2. (*String*) Access username: this one must be the user name of an active user holding the key ROBOT.
3. (*String*) Access password: this must be the clear-text password of the given user name
4. (*String*) Start of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
5. (*String*) End of period. Same format as start of period
6. (*String*) Agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity
7. (*String*) The form name for which you need to have information
8. (*String*) The grader type used to filter out graded forms. It could be any of the following values: "unknown", "agent", "grader", "caller". This parameter is optional and could not be present in the function call
9. (*List*) A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

The same suggestions that are given for *QM.stats* apply.

The method QM.qaformgrading

This method lets you fill a QA form through an RPC-XML call. It replies with the same raw information reported by the *QM.qaformreport* method and could replace it if QA parameters are empty when calling.

1. (*String*) Queue name or names: 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.
2. (*String*) Access username: this one must be the user name of an active user holding the key ROBOT.
3. (*String*) Access password: this must be the clear-text password of the given user name
4. (*String*) Start of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
5. (*String*) End of period. 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).
6. (*String*) The form name for which you need to have information
7. (*String*) The unique identifier for the call to be graded
8. (*String*) The grader type used to filter out graded forms. It could be any of the following values: "unknown", "agent", "grader", "caller". This parameter is optional and could not be present in the function call
9. (*Struct*) The list of QA items score supplied as (string,string) pairs. 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
10. (*List*) 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.
11. (*List*) A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

The same suggestions that are given for *QM.stats* apply.

String queues, String user, String pass, String t_from, String t_to, String formName, String agent, HashMap constraints, ArrayList requestedItems)

The method QM.qacallstograde

1. *(String)* Queue name or names: 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.
2. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
3. *(String)* Access password: this must be the clear-text password of the given user name
4. *(String)* Start of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
5. *(String)* End of period. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).
6. *(String)* The form name for which you need to have information
7. *(String)* Agent filter - an agent's name, like "Agent/101" that must be the filter for all the relevant activity
8. *(List)* A list of constraints used by the engine to filter out all relevant forms (see below for the correct syntax)
9. *(List)* A list of the required analysis to be returned to the client. Each analysis name must be supplied as a String.

The constraints are a set of (key, values) 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)

The method QM.auth

This method is used to authenticate a username / password couple 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.

Call parameters:

1. *(String)* User name
2. *(String)* Password

Response

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

The method QM.authenticate

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.

Call parameters

1. *(String)* User name
2. *(String)* Deferred user name, or blank
3. *(String)* Password
4. *(String)* New password, or blank if you do not want it changed.

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 [mailto: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
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 method QM.findAudio

This method is used to retrieve a recorded file name from the QueueMetrics server. 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.

Call parameters:

1. *(String)* User name (*)
2. *(String)* Password (*)
3. *(String)* Server
4. *(String)* Asterisk-ID (*)
5. *(String)* Call start (integer timestamp)
6. *(String)* Agent
7. *(String)* Queue

The username and password of a user with ROBOT access are mandatory.

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

The *Asterisk-Id* is mandatory and is the one retrieved in the Call Details blocks.

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 response may include zero or more files; it is well possible that multiple recordings be present for one call.

A sample PHP file that shows how to access the findAudio interface is supplied with QueueMetrics.

The method QM.insertRecordTag

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

Call parameters:

1. *(String)* User name (*)
2. *(String)* Password (*)
3. *(String)* Server: the server name in a cluster setup or empty for a not cluster setup
4. *(String)* AsteriskId: the asterisk specific call id
5. *(String)* FileName: the recording filename associated to the tag to be inserted
6. *(String)* Time: the time (in seconds) where the tag will be placed. If empty, the tag will be placed at the beginning of the file
7. *(String)* Duration: the tag duration (in seconds). It could be empty.
8. *(String)* Message: the tag message. If empty, no tags will be added. This is useful for retrieve the list of tags associated to a specific call.
9. *(String)* Color: an hex color value in RGB form to be associated a tag. (from 000000 to FFFFFFFF)

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 *AsteriskId* parameters.

The method QM.taskAddNote

This method is used to insert a note task to a particular user. It's possible to specify the validity start and end date.

Call parameters:

1. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
2. *(String)* Access password: this must be the clear-text password of the given user name
3. *(String)* Task to user: this is the login to the user where the task should be addressed

4. *(String)* Start of validity: This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour). It could be empty.
5. *(String)* End of validity: This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour). It could be empty.
6. *(String)* Task Process Field: This is an optional identifier defined as ProcessFamily/ProcessId to be associated to the task. Either ProcessFamily and/or ProcessId could be empty. The field could be empty.
7. *(String)* Task message: This is the message associated to the task
8. *(String)* Task notes: This is the optional note associated to the task

Response There is only the result response block. No other custom blocks will be returned as a result.

The method QM.tskAddTraining

This method is used to insert a training task to a particular user. It's possible to specify the validity start and end date, a title and an optional URL or ID that will be shown in the task detail page.

Call parameters:

1. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
2. *(String)* Access password: this must be the clear-text password of the given user name
3. *(String)* Task to user: this is the login to the user where the task should be addressed
4. *(String)* Start of validity: This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour). It could be empty.
5. *(String)* End of validity: This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour). It could be empty.
6. *(String)* Task Process Field: This is an optional identifier defined as ProcessFamily/ProcessId to be associated to the task. Either ProcessFamily and/or ProcessId could be empty. The field could be empty.
7. *(String)* Task message: This is the message associated to the task
8. *(String)* Task notes: This is the optional note associated to the task
9. *(String)* Task Training Title: This is an optional task training title.
10. *(String)* Task URL: This is an optional URL (in the http://www.mydomain.com/path format) to be associated with the training task. It could be empty.
11. *(String)* Task ID: This is an optional ID associated to the task. If an URL is defined by the previous parameter, this field will be ignored. It could be empty.

Response There is only the result response block. No other custom blocks will be returned as a result.

The method QM.tskAddMeeting

This method is used to insert a meeting task to a particular user. It's possible to specify the validity start and end date, a title, a date and an optional duration that will be shown in the task detail page.

Call parameters:

1. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
2. *(String)* Access password: this one must be the clear-text password of the given user name
3. *(String)* Task to user: this is the login to the user where the task should be addressed
4. *(String)* Start of validity: This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour). It could be empty.
5. *(String)* End of validity: This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour). It could be empty.
6. *(String)* Task Process Field: This is an optional identifier defined as ProcessFamily/ProcessId to be associated to the task. Either ProcessFamily and/or ProcessId could be empty. The field could be empty.
7. *(String)* Task title: This is the title associated to the task
8. *(String)* Task message: This is the message associated to the task

9. *(String)* Task date: This is the meeting date. This must be written in exactly the format yyyy-mm-dd.hh.mm.ss (do not forget the dot between the date and the hour).

10. *(String)* Task duration: This is the optional meeting duration. If no duration is specified (or specified as 0) the meeting will have an undefined duration.

11. *(String)* Task notes: This is the optional note associated to the task

Response There is only the result response block. No other custom blocks will be returned as a result.

The method QM.setActivationKey

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

Call parameters:

1. *(String)* Access username: this one must be the user name of an active user holding the keys ROBOT and KEYUPDATE.
2. *(String)* Access password: this must be the clear-text password of the given user name
3. *(String)* New key: this is the key that must be set. If blank, the current parameters are reported.

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).

The method QM.broadcastMessage

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

Call parameters:

1. *(String)* Access username: this one must be the user name of an active user holding the key ROBOT.
2. *(String)* Access password: this must be the clear-text password of the given user name
3. *(String)* Message text: the text message to be broadcasted
4. *(String)* Queue Name: the optional queue name (it could be empty)
5. *(String)* Location Name: the optional location name (it could be empty)
6. *(String)* Supervisor Login: an optional supervisor login (it could be empty). The message will be broadcasted to people reporting to the supervisor login.
7. *(String)* Agent Login: an optional agent login (it could be empty). The message will be addressed to the specified agent.
8. *(String)* forEveryone: this could be "true" or "1" for messages to be broadcasted to everyone; "false" or "0" otherwise.

The auth server may return four different responses:

- Access allowed
- Access allowed with supplied user data
- Access is forbidden
- Access is fully delegated to QM

The following table explains the relationship between those states:

	Auth OK	Auth KO
User data from RPC	AUTHORITATIVE	FORBIDDEN
User data from QM	SUCCESSFUL	DELEGATED

If auth was SUCCESSFUL, the local QM database is checked for that user name. If such a user is present, the user details, class and key information are loaded from QM. If such a user is not present, the details are taken from the ones supplied via XML-RPC.

If auth was AUTHORITATIVE, the details are taken from the ones supplied via XML-RPC. Then they are copied to the local user database (with a random password) so that although the user cannot login manually, it is possible to decode the user name for all logged operations (e.g. Updating a queue). If a user with the same name is present, credentials are forcibly updated with the authoritative credentials.

If auth was FORBIDDEN, no other check is done and the user is rejected access.

If auth was DELEGATED, the standard QM logon procedure is done.

The whole procedure is totally transparent to the user, so they do not need to know which is the authority that grants or denies access.

XML-RPC call parameters

Method: QMAuth.auth

Table 1. Input parameters

Pos	Name	Type	Comment
0	System ID	String	As set in QM; if not set: blank
1	Username	String	
2	Password	String	

Table 2. Output block

Pos	Name	Type	Comment
0	Status code	String	One of: * A (authoritative) * S (success) * F (forbidden) * D (delegate) If other, behaves as D.
1	Real name	String	
2	Email	String	
3	Class name	String	Must match an existing class
4	User keys	String	

The following values are implied in QM:

- Enabled = yes
- Masterkey = no

The actual user data is only read by QM in case "A"; otherwise it's ignored, whatever is passed.

Tip: As a reference implementation, see the server that ships with QueueMetrics in the mysql-utils/xml-rpc/xmlrpc_auth_server.php file. It also contains an example of querying a LDAP server in PHP.

Configuration properties

A set of two new configuration properties control external auth sources in in QM:

Property	Description
default.authRpcServerUrl	URL set, XML-RPC auth is used. Points to the URL of the auth server.
default.authSystemId	The system-ID for this QM license. Can be currently any user-chosen name for the system.

A. Response block names

Each XML-RPC call will have its own admissible set of block names that can be asked for. Response blocks come in families, also known as Data Objects. Each response block name is made up of the Data Object name dot the method name, as in KoDO.DiscCauses. Remember that response block names are case sensitive.

Response block names for QM.stats

A complete list of possible QueueMetrics blocks is now maintained in the QueueMetrics User Manual, chapter 6 "Report Details". The 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 XML-RPC code. That is the name of the block that has to be retrieved over XML-RPC.



Note

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 as it is on the User Manual.

Response block names for QM.realtime

DataObject: 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

Response block names for QM.qareport

DataObject: QualAssDO

Quality Assessment information, as displayed in the QA report page.

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
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

Method	Description
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

Response block names for QM.qaformreport and QM.qaformgrading

DataObject: 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

Response block names for QM.qaformsummary

DataObject: QualAssDO

Quality Assessment information related to QA Forms.

Method	Description
FormSummary	Aggregated information for the specified form

Response block names for QM.qacallstograde

DataObject: QAGradingDO

Quality Assessment information related to calls to be graded

Method	Description
qagExtendedProposals	Set of calls to be graded and related information

Response block names for QM.auth

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

This block contains the following information:

- UserName: the login name
- Status: OK if authentication was passed or ERR if it was not passed
- 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 check
- NLogons: The number of logons the user has made. Each successful QM.auth call counts as a logon.

Response block names for QM.findAudio

The QM.findAudio call will return one single block named "AudioFiles".

This block contains the following information:

- Column 0: *Filename*. A file name of the audio file
- Column 1: *URL*. The download URL for this audio recording.

If no calls are found, the block has zero rows.

Response block names for QM.insertRecordTag

The QM.insertRecordTag call will return one single block named "TagRecords".

This block contains the following information:

- Column 0: *TagID*. A technical ID associated to each tag
- Column 1: *CallID*. The QueueMetrics Unique ID associated to the call (in the format asteriskId@server [mailto:asteriskId@server])
- Column 2: *Title*. The recording filename the tag is referring
- Column 3: *Color*. A decimal representation of the tag color
- Column 4: *Time*. The tag start time (in seconds)
- Column 5: *Duration*. The tag duration (in seconds) or 0 if no duration was specified
- Column 6: *Message*. The tag message
- Column 7: *Optilock*. Optilock field in the database. Reserved.
- Column 8: *UserID*. The technical user ID for the user who added the tag
- Column 9: *CreationDate*. The add operation timestamp

If no tags are found, the block has zero rows.

Appendix II: A short list of XML-RPC libraries

To access QueueMetrics, you only need a library with Client capabilities. Server capabilities are not needed. The following list is by no means exhaustive of all available implementations:

Perl	The BlackPerl library is available at http://www.blackperl.com/RPC::XML/
Python	The xmlrpclib ships with any modern version of the language.
JavaScript	The JSON-XML-RPC library can be found at: http://code.google.com/p/json-xml-rpc/
JavaScript (2)	The Jsolait library also offers an XML-RPC module: http://jsolait.net/
Java	There are a lot of implementations available for Java, we recommend Redstone's LGPL library - http://xmlrpc.sourceforge.net/
C / C++	See http://xmlrpc-c.sourceforge.net/
C# / .Net	A connector is available at http://www.xml-rpc.net/
VisualBasic	A COM component that will work on most languages on the Windows platform: http://sourceforge.net/projects/comxmlrpc
PHP	The package XML_RPC is a part of the standard Pear library.