Using the MQ 9.1.5 CD REST API in Linux with no security (for Testing environments)

https://www.ibm.com/support/pages/node/6208006

Date last updated: 23-Sep-2020

Angel Rivera – rivera@us.ibm.com
IBM MQ Support

+++ Objective

To provide detailed instructions for using the MQ 9.1 REST API in Linux and in Windows with No Security.
The explicit disabling of security might be suitable for Testing environments, but it is not suitable for Production environments.

Why configuring the MQ REST API without security?
Divide and conquer! For novice users it is easier to do the implementation in 2 stages:

* Stage 1: Setup the MQ Web Server without security and quick test.

Stage 1.a: The setup of the MQ Web Server is provided in the following tutorial:
https://www.ibm.com/support/pages/node/6118000
Configuring MQ 9.1 Web Server in Linux and in Windows with no security
(for Testing the MQ Web Console)

Stage 1.b: Do a quick test of the MQ REST API. This is the focus for this tutorial.

* Stage 2: After Stage 1 is successfully implemented, then the desired security method can be added. This topic is NOT covered in this tutorial.
For more information about configuring security for the MQ Web Server, see:

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.sec.doc/q127930_.htm
IBM MQ 9.1.x / IBM MQ / Securing /
IBM MQ Console and REST API security

The chapters for this tutorial are:
Chapter 1: Overview (background, FAQs, JSON)
Chapter 2: Installing and using “curl”
Chapter 3: Installing and using “jq”
Chapter 4: Examples of the Administrative API (installation, qmgr, queue, mqsc)
Chapter 5: Examples of the Messaging API (put, get)
Chapter 6: Example of Multi-Instance

++ Caveat about reading messages (updated on 23-Sep-2020)

The MQ REST API allows you to do the equivalent of an MQGET that is destructive (reads the message and removes it from the queue).
CAVEAT: There is no way to acknowledge or commit the receipt of the message so this API should only be used for applications where message loss can be tolerated.

++ Quick summary of commands

+ Start the MQ Web Server (which starts the MQ Web Console and MQ REST API)
strmqweb

+ Display the URIs and Ports used by the MQ Web Console and MQ REST API
dspmqweb
MQWB1124I: Server 'mqweb' is running.
URLS:
https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/
http://orizaba1.fyre.ibm.com:9080/ibmmq/rest/
https://orizaba1.fyre.ibm.com:9443/ibmmq/console/
http://orizaba1.fyre.ibm.com:9080/ibmmq/console/

+ Stop the MQ Web Server
endmqweb

+ Quick curl commands to check that the MQ REST API is going fine.

curl -s -k "https://localhost:9443/ibmmq/rest/v2/admin/installation" -X GET

curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr" -X GET

++ Related article: useful for MQ Administrators

https://www.ibm.com/support/pages/node/6208422
MQ Skill Transfer: REST API

To provide a good starting point for MQ Administrators to learn more about the new feature:
REST API

The chapters are:
Chapter 1: Overview (FAQs, history, references)
Chapter 2: Installation (effects on /opt/mqm)
Chapter 3: Setup and configuration (effects on /var/mqm and queue managers)
Chapter 4: Diagnostics review and what doc to gather (runmqras)
Chapter 5: Common errors and how to resolve them

++ Tools

This tutorial uses 2 line command tools for exploiting the MQ REST API:
  curl => connect to the MQ Web Server via HTTP, send a request and displays the result
  jq => parser for JSON

The following GIT project has a shell script (configure_qmgr.sh) which provides a very good example of using these 2 tools

https://github.com/ibm-messaging/mq-cloud-demo
Sample scenario and applications for demonstrating the IBM MQ on Cloud service

https://github.com/ibm-messaging/mq-cloud-demo/blob/master/qm-config/configure_qmgr.sh
Script that uses curl and jq for configuring an MQ queue manager.

++
+++ Chapter 1: Overview (background, FAQs, JSON)
++

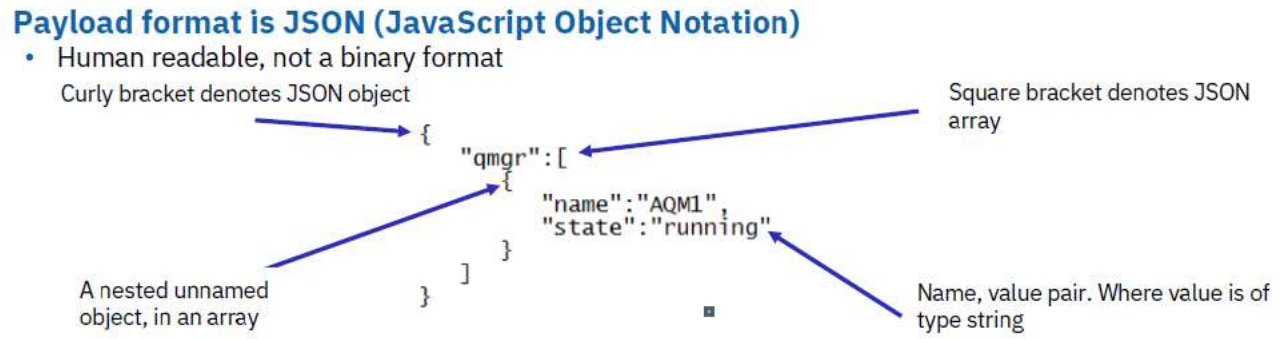
The MQ REST API is available by enabling the MQ Web Server, which is based on WebSphere Liberty Profile (WLP) which is included with the MQ Server.
The MQ Web Server provides an HTTP based administration facilities via the MQ Web Console and the MQ REST API.

The following presentation provides excellent information regarding these features:

https://www.mqtechconference.com/sessions_v2018/MQTC_v2018_MQAdmin_Console_REST.pdf
MQ Administration, the Web Console, & REST API
Sam Goulden, IBM MQ L3 Service,

+ Slide 16: What is REST?
- REpresentational State Transfer
- HTTP is an example of a RESTful architecture
- HTTP defines resources (URL/URIs) and the operations (HTTP verbs) which can use them
  - Originally used for serving web-pages
  - Work really well for APIs too
- Generally light-weight and relatively simple to use, much simpler than SOAP webservices
- Have become incredibly common in recent years
- MQ has taken the approach of following best-practice, and adherence to the various w3c standards when defining its REST API

+ Slide 17: MQ REST API
• An administrative API for managing MQ via REST
• It is much more intuitive to use than PCF and makes it easier to create MQ tooling, that is, a selfservice web-browser based MQ portal using JavaScript
„ No need for an MQ client!
„ Callable from any language which can invoke an HTTPS endpoint
„ Many languages now have built in, or easily added, support for REST
• Payload format is JSON (JavaScript Object Notation)
• Human readable, not a binary format



+ Really good tutorial for JSON

https://shapeshed.com/jq-json/
Last updated Saturday, Nov 16, 2019
JSON on the command line with jq
George Ornbo

+ Main page in the online manual

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.adm.doc/q127600_.htm
IBM MQ 9.1.x / IBM MQ / Administering / Administration using the REST API /
Getting started with the administrative REST API

++ FAQs :

+ How to start the MQ Web Server that is used by the MQ REST API

a) Login as the MQ Administrator.

b) Ensure to use the setmqenv for the proper Installation, such as:
```
. /opt/mqm/bin/setmqenv -n Installation1
```

c) (Optional) Go to the directory that has the xml files for the MQ Web Server. Why? Because the “logs” subdirectory contains *.log files that could be used for problem determination.
```
cd /var/mqm/web/installations/Installation1/servers/mqweb
```

d) Start the MQ Web Server
```
mqm@orizaba1.fyre.ibm.com: /var/mqm/web/installations/Installation1/servers/mqweb
$ strmqweb
Starting server mqweb.
Server mqweb started with process ID 12525.
```

e) Display the status and URLs:
```
mqm@orizaba1.fyre.ibm.com: /var/mqm/web/installations/Installation1/servers/mqweb
$ dspmqweb
MQWB1124I: Server 'mqweb' is running.
URLS:
  https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/
  http://orizaba1.fyre.ibm.com:9080/ibmmq/rest/
  https://orizaba1.fyre.ibm.com:9443/ibmmq/console/
  http://orizaba1.fyre.ibm.com:9080/ibmmq/console/
```

Note:
The following basic URL is the one that that is going to be used in the tutorial:
https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/

Note:
For some URIs, it is necessary to use the proper “versioning” (v1 or v2), otherwise, there could be a failure.

For example, the following does not include the versioning and it fails:

```
mqm@orizaba1.fyre.ibm.com: /home/mqm/
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/admin/qmgr" -X GET
{"error": [{
  "action": "Resubmit the request using a valid URI.",
  "completionCode": 0,
  "explanation": "There is no corresponding REST interface on the provided URI.",
  "message": "MQWB0116E: The URI cannot be invoked as it does not correspond to an
existing REST interface.",
  "msgId": "MQWB0116E",
  "reasonCode": 0,
  "type": "rest"
}]}
```

But by adding the versioning “v1”, it works fine:

```
mqm@orizaba1.fyre.ibm.com: /var/mqm/web/installations/Installation1/servers/mqweb
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v1/admin/qmgr" -X GET
{"qmgr": [
  {
    "name": "QMORI915",
    "state": "endedImmediately"
  },
  {
    "name": "QMDEMO",
    "state": "running"
  },
  {
    "name": "QMORI",
    "state": "running"
  }
]}
```

f) To end the MQ Web Server use:
endmqweb

+ Regarding Multi-Instance queue managers

There is no equivalent to a connectionNameList when accessing the REST API for Multi-Instance queue managers.

- That is, there is no automatic reconnection from CURL or another tool/language that connects to the MQ Web Server to deal with the failover or switchover and move the URL from host-1 to host-2

- It might be possible to use a floating IP address to isolate the REST API for knowing the actual running host.

+ Error handling via JSON

If there is an error during curl, the error will be returned in JSON format.

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue" -X POST -H "Content-Type: application/json" --data "{ "name":"Q2" }"

{"error": [{
"action": "Resubmit the request using the correct format and syntax.",
"completionCode": 0,
"explanation": "The REST API request failed as the data in the request payload could not be parsed.",
"message": "MQWB0107E: Unable to parse the request data due to exception 'Unexpected character ('n' (code 110)): was expecting double-quote to start field name'.",
"msgId": "MQWB0107E",
"reasonCode": 0,
"type": "rest"
}]}

+ Regarding remote queue managers

a) The MQ Web Console works ONLY with local queue managers in the SAME installation as the Console.

b) The MQ REST API can work with remote queue managers, by means of a “gateway” queue manager that is under the same installation of the MQ Web Server.
The remote queue managers need to have connectivity with the gateway queue manager.
LIMITATION: The remote queue managers must be MQ 8.0 or later.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.adm.doc/q131070_.htm
IBM MQ 9.1.x / IBM MQ / Administering / Administration using the REST API / Remote administration using the REST API

c) (Local queue managers) The MQ REST API can work with local queue managers that support the REST API, regardless of their Installation name.
For example, the MQ Web Server could have started with Installation1, but a queue manager from Installation2 can be accessed via the REST API.

LIMITATION: the version of the queue manager needs to be able to support the REST API.
For example, MQ 8.0 queue managers do not support the REST API. If you try to access them, you will get an error message:
MQWB0111E: REST API queue 'SYSTEM.REST.REPLY.QUEUE is not defined

The following is a real example when trying to access MQ 8.0 queue manager “QMORI8”

```
mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k "https://localhost:9443/ibmmq/rest/v2/admin/qmgr/QMORI8" -X GET
{"error": [{
  "action": "Define the required queues on the target queue manager and resubmit the request.",
  "completionCode": 2,
  "explanation": "The REST API request failed as a queue required to handle PCF requests is not defined on the queue manager.",
  "message": "MQWB0111E: REST API queue 'SYSTEM.REST.REPLY.QUEUE' is not defined on queue manager 'QMORI8'.",
  "msgId": "MQWB0111E",
  "reasonCode": 2085,
  "type": "rest"
}]}
```

++
+++ Chapter 2: Installing and using “curl”
++

This tutorial uses 2 line command tools for exploiting the MQ REST API:
  curl => connect to the MQ Web Server via HTTP, send a request and displays the result
  jq => parser for JSON

Just to give you a “taste”, the following examples show how to use the tools from the command line of a Linux server, to get the list and status of the queue managers.
The output is in JSON format.

+ Quick example of “curl” by itself:

mqm@orizaba1.fyre.ibm.com: /var/mqm/web/installations/Installation1/servers/mqweb
$ **curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v1/admin/qmgr" -X GET**
{"qmgr": [
  {
    "name": "QMORI915",
    "state": "endedImmediately"
  },
  {
    "name": "QMDEMO",
    "state": "running"
  },
  {
    "name": "QMORI",
    "state": "running"
  }
]}

++ What is curl?

https://en.wikipedia.org/wiki/CURL
cURL
From Wikipedia, the free encyclopedia

cURL (pronounced 'curl') is a computer software project providing a library (libcurl) and command-line tool (curl) for transferring data using various network protocols.
The name stands for "Client URL".

+ Linux (Installation and configuration)

We are using Virtual Machines that have Linux RHEL 7.6, and “curl” is already installed.

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ **rpm -qa | grep curl**
python-pycurl-7.19.0-19.el7.x86_64
curl-7.29.0-51.el7.x86_64
libcurl-7.29.0-51.el7.x86_64

In case that curl is not installed in your Linux server, you could login as user root and issue:

 # yum install curl

+ Windows (Installation and configuration)

In contrast to our Linux VMs, our Windows VMs do not come with “curl”.
The rest of this section provides the instructions for downloading and installing “curl” in a Windows server.

https://curl.haxx.se/download.html
Curl is a command line tool for transferring data specified with URL syntax.

Note: If you go to the table and download, you will get ONLY the source code.
To download the pre-made executable, you need to interact with the Download Wizard and specify "curl executable":

Select:
Download Wizard
Need help to select what to download? Use the curl Download Wizard!

Select Type of Package
We provide packages of different types. Select one (or select 'show all' to view all types)
-
curl executable - You will get a pre-built 'curl' binary from this link (or in some cases, by using the information that is provided at the page this link takes you). You may or may not get 'libcurl' installed as a shared library/DLL.

curl executable > Win64 > Generic > * > CPU
Select for What CPU
We have packages listed for 2 different CPUs for Win64 .
Show package for Win64 on
x86_64

For Win64 on x86_64 curl executable
curl version: 7.52.1 - SSL enabled SSH enabled
URL: https://bintray.com/artifact/downlo...-win64-mingw.7z
Provided by: Viktor Szakáts
This package is type curl executable You will get a pre-built 'curl' binary from this link (or in some cases, by using the information that is provided at the page this link takes you).
You may or may not get 'libcurl' installed as a shared library/DLL.

Into a directory which is in the PATH and where you store your utilities and batch files, such as:
C:\wintools

After extracting, the full path name of executable:
C:\wintools\curl\curl-7.52.1-win64-mingw\bin\curl.exe
And of DLL:
C:\wintools\curl\curl-7.52.1-win64-mingw\lib

Copy:
C:\wintools\curl\curl-7.52.1-win64-mingw\bin\curl.exe
to be placed in:
C:\wintools\

Note:
Download curl zip
Extract the contents (if you have downloaded the correct version you should find curl.exe)
Place curl.exe in a folder where you keep your software (e.g. D:\software\curl\curl.exe)
To run curl from the command line
a) Right-hand-click on "My Computer" icon
b) Select Properties
c) Click 'Advanced system settings' link
d) Go to tab [Advanced] - 'Environment Variables' button
e) Under System variable select 'Path' and Edit button
f) Add a semicolon followed by the path to where you placed your curl.exe: for example:
set PATH=%PATH%;C:\wintools\curl

+ Testing:
C:\> curl
curl: try 'curl --help' or 'curl --manual' for more information

+ Now you can run from the command line by typing:
curl www.google.com

You should be able to see several html statements, beginning with:

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including webpages,

images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="noodp" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title> <script nonce="wcP9uuNIerSYcFZDvJPJzA==">(function()

… and ending with …

</script>
</body></html>

++ Syntax of curl

Suggestions:

1) For simpler URIs, there is no need to enclose the URI between double quotes.
However, for advanced URIs with search items, you may need to enclose it between double quotes.
A recommendation for keeping things consistent and extendable is to always use the double quotes around the URI.

2) Using the -s (silent) mode to avoid showing the “progress meter” when doing Pipes.

The syntax for curl is documented in:

https://curl.haxx.se/docs/manpage.html

```
The following flags/options are used in this tutorial:

-k, --insecure
(TLS) By default, every SSL connection curl makes is verified to be secure. This option
allows curl to proceed and operate even for server connections otherwise considered
insecure.
Example:
-k http://orizaba1.fyre.ibm.com:9080/ibmmq/rest/v1/admin/qmgr

-X, --request <command>
(HTTP) Specifies a custom request method to use when communicating with the HTTP server.
The specified request method will be used instead of the method otherwise used (which
defaults to GET). Read the HTTP 1.1 specification for details and explanations. Common
additional HTTP requests include PUT and DELETE, but related technologies like WebDAV
offers PROPFIND, COPY, MOVE and more.
Example:
-X GET

-s, --silent
Silent or quiet mode. Don't show progress meter or error messages. Makes Curl mute. It
will still output the data you ask for.
This is the "progress meter" which is shown when using a Pipe |
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   241  100   241    0     0   1747      0 --:--:-- --:--:-- --:--:--  1759

-H, --header <header/@file>
(HTTP) Extra header to include in the request when sending HTTP to a server.
Example:
-H "Content-Type: text/plain;charset=utf-8"

-d @filename
If you start the data with the letter @, the rest should be a file name to read the data
from, or - if you want curl to read the data from stdin.
Posting data from a file named 'foobar' would thus be done with -d, --data @foobar.
When -d, --data is told to read from a file like that, carriage returns and newlines will
```

```
be stripped out.
Example:
-d @mq-curl-command.txt

---data "Data enclosed between quotes"
(HTTP MQTT) Sends the specified data in a POST request to the HTTP server, in the same way
that a browser does when a user has filled in an HTML form and presses the submit button.
This will cause curl to pass the data to the server using the content-type application/x-
www-form-urlencoded.
Example:
--data "This is a test of doing MQ Put via REST API"
```

+ Reference

https://developer.ibm.com/answers/questions/454181/mq-rest-api-example-of-using-curl-in-windows-to-is.html?childToView=454182#answer-454182
MQ REST API, example of using curl in Windows to issue runmqsc commands.

++
+++ Chapter 3: Installing and using “jq”
++

This tutorial uses 2 line command tools for exploiting the MQ REST API:
  curl => connect to the MQ Web Server via HTTP, send a request and displays the result
  jq => parser for JSON

Just to give you a “taste”, the following examples show how to use the tools from the command line of a Linux server, to get the list and status of the queue managers.
The output is in JSON format.

+ Quick example of using both “curl” and “jq”:

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ **curl -s -k "http://orizaba1.fyre.ibm.com:9080/ibmmq/rest/v1/admin/qmgr" -X GET | jq '.qmgr | .[] | .name,.state'**
"QMORI915"
"endedImmediately"
"QMDEMO"
"running"
"QMORI"
"running"

++ What is “jq”?

https://stedolan.github.io/jq/
jq is a lightweight and flexible command-line JSON processor

jq is like sed for JSON data - you can use it to slice and filter and map and transform structured data with the same ease that sed, awk, grep and friends let you play with text.
jq is written in portable C, and it has zero runtime dependencies. You can download a single binary, scp it to a far away machine of the same type, and expect it to work.
jq can mangle the data format that you have into the one that you want with very little effort, and the program to do so is often shorter and simpler than you'd expect.

++ Installation

+ Linux

Even though some articles mentioned that yum could be used, well, the following did not work for me:
  yum install jq

However, the following recommendation worked fine (from forum article):
https://serverfault.com/questions/768026/how-to-install-jq-on-rhel6-5
How to install jq on RHEL6.5

Here are the steps that worked fine:

login as root

cd /usr/local/bin

wget -O jq https://github.com/stedolan/jq/releases/download/jq-1.6/jq-linux64

chmod +x ./jq

+ Windows:

Use web browser and enter the following URL
  https://github.com/stedolan/jq/releases/download/jq-1.6/jq-win64.exe

Download jq_win64.exe into a directory in your PATH, such as:
  C:\wintools

You could rename the downloaded file or copy into jq.exe (because it could be easier to use)
  copy jq_win64.exe jq.exe

++
+++ Chapter 4: Examples of the Administrative API (installation, qmgr, queue, mqsc)
++

The examples in the following steps assume that your REST API URL is the default URL:
https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v1/

If your URL is different than the default, substitute your URL in the following steps.

Remember that in this tutorial we are not using security.

If you are using basic security then append the userid and its password:
-u mqadmin:mqadmin

++ Notice that the output in JSON format does not include a line feed at the end

```
mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v1/admin/qmgr" -X GET
{"qmgr": [{
  "name": "QMORI",
  "state": "running"
}]}mqm@orizaba1.fyre.ibm.com: /home/mqm
```

Notice that the output in JSON format does not include a line feed at the end, thus, the prompt for the next statement is in the same line as the last character of the output. Thus, for illustration purposes, a line feed is added in this tutorial in order to have the next prompt in a different line.

```
mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v1/admin/qmgr" -X GET
{"qmgr": [{
  "name": "QMORI",
  "state": "running"
}]}
mqm@orizaba1.fyre.ibm.com: /home/mqm
```

++ Examples for /admin/installation

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q128350_.htm
IBM MQ 9.1.x / IBM MQ / Reference / Administration reference / Administrative REST API reference / REST API resources /
admin/installation
GET

Resource URL
  https://host:port/ibmmq/rest/v2/admin/installation/{installationName}

The information that is returned is similar to the information that is returned by the dspmqver (display version information) control command.

+ Show the installations:

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/installation" -X GET
{"installation": [{
  "name": "Installation1",
  "platform": "unix",
  "version": "9.1.5.0"
}]}

+ Show the details for Installation1

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/installation/Installation1?attributes=*" -X GET
{"installation": [{
  "extended": {
    "dataPath": "/var/mqm",
    "description": "",
    "installationPath": "/opt/mqm",
    "level": "p915-L200316",
    "maximumCommandLevel": 915,
    "operatingSystem": "Linux 3.10.0-957.12.1.el7.x86_64",
    "primary": false
  },
  "name": "Installation1",
  "platform": "unix",
  "version": "9.1.5.0"
}]}

+ Show the details for the installationPath for Installation1

```
mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k
"https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/installation/Installation1?attributes=extended.installationPath" -X GET
{"installation": [{
  "extended": {"installationPath": "/opt/mqm"},
  "name": "Installation1",
  "platform": "unix",
  "version": "9.1.5.0"
}]}
```

+ Using jq to filter the extended values

```
mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k
https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/installation/Installation1?attributes=* -X GET | jq '.installation | .[] | .extended'
{
  "dataPath": "/var/mqm",
  "description": "",
  "installationPath": "/opt/mqm",
  "level": "p915-L200316",
  "maximumCommandLevel": 915,
  "operatingSystem": "Linux 3.10.0-957.12.1.el7.x86_64",
  "primary": false
}
```

+ Using jq to filter the value for the installationPath

```
mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k
https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/installation/Installation1?attributes=* -X GET | jq '.installation | .[] | .extended.installationPath'
"/opt/mqm"
```

++ Examples for /admin/qmgr

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q128360_.htm
IBM MQ 9.1.x / IBM MQ / Reference / Administration reference / Administrative REST API reference / REST API resources /
/admin/qmgr
GET

Resource URL
https://host:port/ibmmq/rest/v2/admin/qmgr/{qmgrName}

The information that is returned is similar to the information that is returned by the dspmq (display queue managers) control command, the DISPLAY QMSTATUS MQSC command, and the Inquire Queue Manager Status PCF command.

Notes about the queue manager name:
- The queue manager name is case-sensitive.
- If the queue manager name includes a forward slash, a period, or a percent sign, these characters must be URL encoded:
A forward slash (/) must be encoded as %2F.
A percent sign (%) must be encoded as %25.
A period (.) must be encoded as %2E.

+ Show the queue managers

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr" -X GET


```
{"qmgr": [
  {
    "name": "QMORI915",
    "state": "endedImmediately"
  },
  {
    "name": "QMDEMO",
    "state": "running"
  },
  {
    "name": "QMORI",
    "state": "running"
  }
]}
```

+ Show only one queue manager (QMORI)

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ **curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI" -X GET**
{"qmgr": [{
  "name": "QMORI",
  "state": "running"
}]}

+ Show extended information for queue manager QMORI

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ **curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?attributes=extended" -X GET**
{"qmgr": [{
  "extended": {
    "installationName": "Installation1",
    "isDefaultQmgr": false,
    "permitStandby": "notPermitted"
  },
  "name": "QMORI",
  "state": "running"
}]}

+ Show the extended attribute permitStandby for all the queue managers

mqm@orizaba1.fyre.ibm.com: /home/mqm
$ **curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr?attributes=extended.permitStandby" -X GET**
{"qmgr": [
  {
    "extended": {"permitStandby": "notApplicable"},
    "name": "QMORI915",
    "state": "endedImmediately"
  },
  {
    "extended": {"permitStandby": "notPermitted"},
    "name": "QMDEMO",
    "state": "running"
  },
  {
    "extended": {"permitStandby": "notPermitted"},
    "name": "QMORI",

```
    "state": "running"
  }
]}
```

+ Show the extended attribute permitStandby for only the queue manager QMORI

```
$ curl -s -k
"https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?attributes=extended.permitStandby" -X GET
{"qmgr": [{
  "extended": {"permitStandby": "notPermitted"},
  "name": "QMORI",
  "state": "running"
}]}
```

+ Use jq to display the queue manager QMORI (notice that “qmgr” is not shown)

```
$ curl -s -k
"https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?attributes=extended.permitStandby" -X GET | jq '.qmgr'
[
  {
    "extended": {
      "permitStandby": "notPermitted"
    },
    "name": "QMORI",
    "state": "running"
  }
]
```

+ (Fine tuning) Use jq to simplity the output above and eliminate the outer square brackets

```
$ curl -s -k
"https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?attributes=extended.permitStandby" -X GET | jq '.qmgr | .[] '
{
  "extended": {
    "permitStandby": "notPermitted"
  },
  "name": "QMORI",
  "state": "running"
}
```

+ (Fine tuning) Use jq to show only the extended attributes

```
$ curl -s -k
```

"https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?attributes=extended.permitStandby" -X GET | jq '.qmgr | .[] | .extended'

```
{
  "permitStandby": "notPermitted"
}
```


+ (Fine tuning) Use jq to get the value for permitStandby

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?attributes=extended.permitStandby" -X GET | jq '.qmgr | .[] | .extended.permitStandby'

```
"notPermitted"
```


+ Show status of queue manager QMORI: status

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?status" -X GET

```
{"qmgr": [{
  "name": "QMORI",
  "state": "running"
}]}
```


+ Show status of queue manager QMORI: status=*

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/qmgr/QMORI?status=*" -X GET

```
{"qmgr": [{
  "name": "QMORI",
  "state": "running",
  "status": {
    "channelInitiatorState": "running",
    "connectionCount": 25,
    "ldapConnectionState": "disconnected",
    "publishSubscribeState": "running",
    "started": "2020-05-08T15:09:42.000Z"
  }
}]}
```

++ Examples for /admin/qmgr/{qmgrName}/queue (this is v1)

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q129010_.htm
IBM MQ 9.1.x / IBM MQ / Reference / Administration reference / Administrative REST API reference / REST API resources /
/admin/qmgr/{qmgrName}/queue

This resource URL is available only in version 1 of the REST API. To create queues using version 2 of the REST API, use the resource: /admin/action/qmgr/{qmgrName}/mqsc

+ POST
[V9.1.0 Jul 2018] Use the HTTP POST method with the queue resource to create a queue on a specified queue manager.
This REST API command is similar to the Change, Copy, and Create Queue PCF command, and the DEFINE queues MQSC commands.

The manual says:
The following JSON payload is sent:
```
{
   "name": "localQueue"
}
```

But if you construct the CURL data as-is (well, in a single line), it will fail due to parsing errors:

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue" -X POST -H "Content-Type: application/json" --data "{ "name":"Q2" }"
```
{"error": [{
  "action": "Resubmit the request using the correct format and syntax.",
  "completionCode": 0,
  "explanation": "The REST API request failed as the data in the request payload could not be parsed.",
  "message": "MQWB0107E: Unable to parse the request data due to exception 'Unexpected character ('n' (code 110)): was expecting double-quote to start field name'.",
  "msgId": "MQWB0107E",
  "reasonCode": 0,
  "type": "rest"
}]}
```

The reason is that the double quotes " in the argument must be ESCAPED with a back slash \

Let's escape the double quotes and this time the command runs fine.

$ curl -s -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/ -X POST -H "Content-Type: application/json" --data "{ \"name\":\"Q2\" }"

Note:
You can create a text file with the original (not escaped double quotes) such as:
File name: name-of-queue.txt
Contents: { "name":"Q2" }
Then you can use the file with the -d parameter and use an @ character before the filename:

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue" -X POST -H "Content-Type: application/json" **-d @name-of-queue.txt**

+ DELETE
Use the HTTP DELETE method with the queue resource to delete a specified queue on a specified queue manager.

$ curl -s -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q3 -X DELETE

Similar example, but you want to also delete the messages from the queue (purge):

$ curl -s -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q3?purge -X DELETE

+ GET
Similar to the DISPLAY QUEUE and DISPLAY QSTATUS MQSC commands.

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue" -X GET


```
{"queue": [
  {
    "name": "SYSTEM.ADMIN.STATISTICS.QUEUE",
    "type": "local"
  },
  {
    "name": "SYSTEM.DEFAULT.INITIATION.QUEUE",
    "type": "local"
  },
```


Etc…

+ Show the status for a queue

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?status=*" -X GET


```
{"queue": [{
  "name": "Q1",
  "status": {
    "currentDepth": 1,
    "lastGet": "",
    "lastPut": "",
    "mediaRecoveryLogExtent": "",
    "monitoringRate": "off",
    "oldestMessageAge": -1,
    "onQueueTime": {
      "longSamplePeriod": -1,
      "shortSamplePeriod": -1
    },
    "openInputCount": 0,
    "openOutputCount": 0,
    "uncommittedMessages": 0
  },
  "type": "local"
}]}
```


+ Get the value for currentDepth
(You can use the above output to help you refine the argument for jq)

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?status=*" -X GET | jq '.queue | .[] | .status.currentDepth'

1

+ For the next queries, in another command prompt the following command was issued, in order to have an active handle:
amqsput Q1 QMORI

$ curl -s -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?applicationHandle=*" -X GET


```
{"queue": [{
  "applicationHandle": [{
    "asynchronousConsumerState": "none",
    "channelName": "SYSTEM.DEF.SVRCONN",
    "connectionName": "127.0.0.1",
    "description": "IBM MQ Channel",
    "openOptions": [
```

```
      "MQOO_OUTPUT",
      "MQOO_FAIL_IF_QUIESCING"
    ],
    "processId": 20184,
    "qmgrTransactionId": "0000000000000000",
    "state": "inactive",
    "tag": "amqsputc",
    "threadId": 0,
    "type": "userApplication",
    "unitOfWorkId":
"00000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000",
    "unitOfWorkType": "qmgr",
    "userId": "mqm"
  }],
  "name": "Q1",
  "type": "local"
}]}
```


```
$ curl -s -k
"https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?attributes=*&stat
us=*&applicationHandle=*"
```


```
{"queue": [{
  "applicationDefaults": {
    "clusterBind": "onOpen",
    "messagePersistence": "nonPersistent",
    "messagePriority": 0,
    "messagePropertyControl": "compatible",
    "putResponse": "synchronous",
    "readAhead": "no",
    "sharedInput": true
  },
  "applicationHandle": [{
    "asynchronousConsumerState": "none",
    "channelName": "SYSTEM.DEF.SVRCONN",
    "connectionName": "127.0.0.1",
    "description": "IBM MQ Channel",
    "openOptions": [
      "MQOO_OUTPUT",
      "MQOO_FAIL_IF_QUIESCING"
    ],
```

"processId": 20184,
"qmgrTransactionId": "0000000000000000",
"state": "inactive",
"tag": "amqsputc",
"threadId": 0,
"type": "userApplication",
"unitOfWorkId":
"00000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000",
"unitOfWorkType": "qmgr",
"userId": "mqm"
}],
"cluster": {
"name": "",
"namelist": "",
"transmissionQueueForChannelName": "",
"workloadPriority": 0,
"workloadQueueUse": "asQmgr",
"workloadRank": 0
},
"dataCollection": {
"accounting": "asQmgr",
"monitoring": "asQmgr",
"statistics": "asQmgr"
},
"events": {
"depth": {
"fullEnabled": true,
"highEnabled": false,
"highPercentage": 80,
"lowEnabled": false,
"lowPercentage": 20
},
"serviceInterval": {
"duration": 999999999,
"highEnabled": false,
"okEnabled": false
}
},
"extended": {
"allowSharedInput": true,

```
  "backoutRequeueQueueName": "",
  "backoutThreshold": 0,
  "custom": "",
  "enableMediaImageOperations": "asQmgr",
  "supportDistributionLists": false
},
"general": {
  "description": "",
  "inhibitGet": false,
  "inhibitPut": false,
  "isTransmissionQueue": false
},
"name": "Q1",
"status": {
  "currentDepth": 1,
  "lastGet": "",
  "lastPut": "",
  "mediaRecoveryLogExtent": "",
  "monitoringRate": "off",
  "oldestMessageAge": -1,
  "onQueueTime": {
    "longSamplePeriod": -1,
    "shortSamplePeriod": -1
  },
  "openInputCount": 0,
  "openOutputCount": 1,
  "uncommittedMessages": 0
},
"storage": {
  "maximumDepth": 5000,
  "maximumMessageLength": 4194304,
  "messageDeliverySequence": "priority",
  "nonPersistentMessageClass": "normal"
},
"timestamps": {
  "altered": "2020-05-12T16:42:56.000Z",
  "created": "2020-05-12T16:42:31.000Z"
},
"trigger": {
  "data": "",
  "depth": 1,
  "enabled": false,
  "initiationQueueName": "",
  "messagePriority": 0,
  "processName": "",
  "type": "first"
```

```
  },
  "type": "local"
}]}
```

+ Using comparison operator “greaterThan:Number”

First example has greaterThan 3 (and there are no hits)

$ **curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue?attributes=*&status=*&filter=status.openInputCount:greaterThan:3" -X GET**

{"queue": []}

Second example has greaterThan 1 (and there is 1 hit)

$ **curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue?attributes=*&status=*&filter=status.openInputCount:greaterThan:1" -X GET**


```
{"queue": [{
  "applicationDefaults": {
    "clusterBind": "onOpen",
    "messagePersistence": "persistent",
    "messagePriority": 0,
    "messagePropertyControl": "compatible",
    "putResponse": "synchronous",
    "readAhead": "no",
    "sharedInput": true
  },
  "cluster": {
    "name": "",
    "namelist": "",
    "transmissionQueueForChannelName": "",
    "workloadPriority": 0,
    "workloadQueueUse": "asQmgr",
    "workloadRank": 0
  },
  "dataCollection": {
    "accounting": "asQmgr",
    "monitoring": "asQmgr",
    "statistics": "asQmgr"
  },
  "events": {
    "depth": {
      "fullEnabled": true,
      "highEnabled": false,
      "highPercentage": 80,
      "lowEnabled": false,
      "lowPercentage": 20
    },
    "serviceInterval": {
```

```
    "duration": 999999999,
    "highEnabled": false,
    "okEnabled": false
  }
},
"extended": {
  "allowSharedInput": true,
  "backoutRequeueQueueName": "",
  "backoutThreshold": 0,
  "custom": "",
  "enableMediaImageOperations": "asQmgr",
  "supportDistributionLists": false
},
"general": {
  "description": "Control queue for queued Pub/Sub interface",
  "inhibitGet": false,
  "inhibitPut": false,
  "isTransmissionQueue": false
},
"name": "SYSTEM.BROKER.CONTROL.QUEUE",
"status": {
  "currentDepth": 0,
  "lastGet": "",
  "lastPut": "",
  "mediaRecoveryLogExtent": "",
  "monitoringRate": "off",
  "oldestMessageAge": -1,
  "onQueueTime": {
    "longSamplePeriod": -1,
    "shortSamplePeriod": -1
  },
  "openInputCount": 3,
  "openOutputCount": 0,
  "uncommittedMessages": 0
},
"storage": {
  "maximumDepth": 5000,
  "maximumMessageLength": 4194304,
  "messageDeliverySequence": "fifo",
  "nonPersistentMessageClass": "normal"
},
"timestamps": {
  "altered": "2020-05-08T15:09:41.000Z",
  "created": "2020-05-08T15:09:41.000Z"
},
"trigger": {
```

```
    "data": "",
    "depth": 1,
    "enabled": false,
    "initiationQueueName": "",
    "messagePriority": 0,
    "processName": "",
    "type": "first"
  },
  "type": "local"
}]}
```

++ Examples for the general feature of “mqsc” (v2)

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q129385_.htm
IBM MQ 9.1.x / IBM MQ / Reference / Administration reference / Administrative REST API reference / REST API resources /
/admin/action/qmgr/{qmgrName}/mqsc

In this example, a DISPLAY command is issued to show the attributes for an existing channel.
You MUST escape the double quotes with a back slash.

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"DISPLAY CHANNEL(SYSTEM.DEF.SVRCONN)\" } }"


```
{
  "commandResponse": [{
    "completionCode": 0,
    "reasonCode": 0,
    "text": ["AMQ8414I: Display Channel details.   CHANNEL(SYSTEM.DEF.SVRCONN)
CHLTYPE(SVRCONN)   ALTDATE(2020-05-08)                ALTTIME(08.09.40)   CERTLABL( )
COMPHDR(NONE)   COMPMSG(NONE)                    DESCR( )   DISCINT(0)
HBINT(300)   KAINT(AUTO)                      MAXINST(999999999)   MAXINSTC(999999999)
MAXMSGL(4194304)   MCAUSER( )                        MONCHL(QMGR)   RCVDATA( )
RCVEXIT( )   SCYDATA( )                     SCYEXIT( )   SENDDATA( )
SENDEXIT( )   SHARECNV(10)                     SSLCAUTH(REQUIRED)   SSLCIPH( )
SSLPEER( )   TRPTYPE(TCP)                    "]
  }],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```


In this example, a DISPLAY command is issued to find out if a channel already exists (in this case, it does NOT exist).

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"DISPLAY CHANNEL(NEWSVRCONN)\" } }"


```
{
  "commandResponse": [{
    "completionCode": 2,
    "reasonCode": 2085,
    "text": ["AMQ8147E: IBM MQ object NEWSVRCONN not found."]
  }],
```

```
  "overallCompletionCode": 2,
  "overallReasonCode": 3008
}
```

+ Show how to query for authority records for SYSTEM.CHANNEL.SYNCQ and there is 1 record:

.

```
$ curl -s -k "https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"display authrec PROFILE(SYSTEM.CHANNEL.SYNCQ)\"  } }"
{
  "commandResponse": [
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": ["AMQ8864I: Display authority record details.
PROFILE(SYSTEM.CHANNEL.SYNCQ)           ENTITY(AngelRivera@AzureAD)
ENTTYPE(PRINCIPAL)                      OBJTYPE(QUEUE)
AUTHLIST(BROWSE,CHG,CLR,DLT,DSP,GET,INQ,PUT,PASSALL,PASSID,SET,SETALL,SETID)"]
    },
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": ["AMQ8864I: Display authority record details.
PROFILE(SYSTEM.CHANNEL.SYNCQ)           ENTITY(mqm@ANGELITO)    ENTTYPE(GROUP)
OBJTYPE(QUEUE)
AUTHLIST(BROWSE,CHG,CLR,DLT,DSP,GET,INQ,PUT,PASSALL,PASSID,SET,SETALL,SETID)"]
    }
  ],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```

+ Define queue local Q5

```
$ curl -s -k
"https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"DEFINE QLOCAL(Q5)\"  } }"
{
  "commandResponse": [{
    "completionCode": 0,
    "reasonCode": 0,
    "text": ["AMQ8006I: IBM MQ queue created."]
  }],
  "overallCompletionCode": 0,
```

```
  "overallReasonCode": 0
}
```


+ Display the newly created queue local Q5

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"DISPLAY QLOCAL(Q5)\" } }"


```
{
  "commandResponse": [{
    "completionCode": 0,
    "reasonCode": 0,
    "text": ["AMQ8409I: Display Queue details.   QUEUE(Q5)
TYPE(QLOCAL)   ACCTQ(QMGR)                      ALTDATE(2020-05-12)   ALTTIME(13.08.48)
BOQNAME( )   BOTHRESH(0)                      CLUSNL( )   CLUSTER( )
CLCHNAME( )   CLWLPRTY(0)                      CLWLRANK(0)   CLWLUSEQ(QMGR)
CRDATE(2020-05-12)   CRTIME(13.08.48)                  CURDEPTH(0)   CUSTOM( )
DEFBIND(OPEN)   DEFPRTY(0)                     DEFPSIST(NO)   DEFPRESP(SYNC)
DEFREADA(NO)   DEFSOPT(SHARED)                    DEFTYPE(PREDEFINED)   DESCR( )
DISTL(NO)   GET(ENABLED)                     HARDENBO   IMGRCOVQ(QMGR)
INITQ( )   IPPROCS(0)                     MAXDEPTH(5000)   MAXMSGL(4194304)
MAXFSIZE(DEFAULT)   MONQ(QMGR)                       MSGDLVSQ(PRIORITY)   NOTRIGGER
NPMCLASS(NORMAL)   OPPROCS(0)                      PROCESS( )   PUT(ENABLED)
PROPCTL(COMPAT)   QDEPTHHI(80)                     QDEPTHLO(20)   QDPHIEV(DISABLED)
QDPLOEV(DISABLED)   QDPMAXEV(ENABLED)                  QSVCIEV(NONE)
QSVCINT(999999999)                RETINTVL(999999999)   SCOPE(QMGR)
SHARE   STATQ(QMGR)                       TRIGDATA( )   TRIGDPTH(1)
TRIGMPRI(0)   TRIGTYPE(FIRST)                    USAGE(NORMAL)"]
  }],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```


+ Display the CUDEPTH for Q5

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v1/admin/qmgr/QMFOR910/queue/Q5?status=status.currentDepth" -X GET


```
{"queue": [{
  "name": "Q5",
  "status": {"currentDepth": 0},
  "type": "local"
}]}
```

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"DISPLAY QLOCAL(Q5) CURDEPTH\" } }"


```
{
  "commandResponse": [{
    "completionCode": 0,
    "reasonCode": 0,
    "text": ["AMQ8409I: Display Queue details.   QUEUE(Q5)
TYPE(QLOCAL)   CURDEPTH(0)                      "]
  }],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```


+ Example that shows a query that has 2 records.
In this example, the queue Q2 is being used by amqsput and amqsget.

$ curl -s -k "https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" --data "{ \"type\":\"runCommand\", \"parameters\": { \"command\": \"DISPLAY QSTATUS(Q2) TYPE(HANDLE) ALL\" } }"


```
{
  "commandResponse": [
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": ["AMQ8450I: Display queue status details.   QUEUE(Q2)
TYPE(HANDLE)   APPLDESC( )                      APPLTAG(amqsget)   APPLTYPE(USER)
BROWSE(NO)   CHANNEL( )                        CONNAME( )   ASTATE(NONE)
HSTATE(ACTIVE)   INPUT(SHARED)                      INQUIRE(NO)   OUTPUT(NO)
PID(2684)    QMURID(0.0)                       SET(NO)   TID(1)
URID(XA_FORMATID[] XA_GTRID[] XA_BQUAL[])   URTYPE(QMGR)
USERID(mqm)"]
    },
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": ["AMQ8450I: Display queue status details.   QUEUE(Q2)
TYPE(HANDLE)   APPLDESC( )                      APPLTAG(amqsput)   APPLTYPE(USER)
BROWSE(NO)   CHANNEL( )                        CONNAME( )   ASTATE(NONE)
HSTATE(INACTIVE)   INPUT(NO)                        INQUIRE(NO)   OUTPUT(YES)
PID(2318)    QMURID(0.0)                       SET(NO)   TID(1)
URID(XA_FORMATID[] XA_GTRID[] XA_BQUAL[])   URTYPE(QMGR)
USERID(mqm)"]
    }
  ],
```

```
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```


++ Using a text file with the runmqsc command (and curl reads from that file)

Another way to provide the runmqsc command is by using a file.
For example, the following text file was created:
   mq-curl-command.txt
The contents is a single line:

```
{ "type":"runCommand", "parameters": { "command": "DISPLAY LISTENER(*) PORT" } }
```


Here are the invocation and the results:

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/admin/action/qmgr/QMORI/mqsc" -X POST -H "Content-Type: application/json" -d @mq-curl-command.txt


```
{
  "commandResponse": [
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": ["AMQ8630I: Display listener information details.
LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)    PORT(0)"]
    },
    {
      "completionCode": 0,
      "reasonCode": 0,
      "text": ["AMQ8630I: Display listener information details.
LISTENER(SYSTEM.LISTENER.TCP.1)          PORT(1416)"]
    }
  ],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```

++
+++ Chapter 5: Examples of the Messaging API (put, get)
++

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.dev.doc/q130740_.htm
IBM MQ 9.1.x / IBM MQ / Reference / Developing applications reference / Messaging REST API reference / REST API resources /
/messaging/qmgr/{qmgrName}/queue/{queueName}/message

++ Putting a message

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/messaging/qmgr/QMORI/queue/Q1/message" -X POST -H "Content-Type: text/plain;charset=utf-8" --data "This is a test of doing MQ Put via REST API"

The put worked fine (no errors were displayed).

+ Get the CURDEPTH for the queue Q1

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?status=*" -X GET | jq '.queue | .[] | .status.currentDepth'
1

+ Let's use the MQ sample "amqsbcg" to do a non-destructive browse of the message:

$ amqsbcg Q1 QMORI
```
MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 1208
  Format : 'MQSTR   '
  Priority : 4  Persistence : 0
  MsgId : X'414D5120514D4F5249202020202020203676B55E0281B224'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'QMORI                                         '
  ** Identity Context
  UserIdentifier : '            '
  AccountingToken :
   X'0000000000000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : 'UNAUTHENTICATED                '
  ** Origin Context
```

```
  PutApplType    : '28'
  PutApplName    : 'IBM MQ REST API             '
  PutDate  : '20200512'    PutTime  : '20304983'
  ApplOriginData : 'REST'

  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'

****   Message      ****
 length - 43 of 43 bytes
00000000:  5468 6973 2069 7320 6120 7465 7374 206F          'This is a test o'
00000010:  6620 646F 696E 6720 4D51 2050 7574 2076          'f doing MQ Put v'
00000020:  6961 2052 4553 5420 4150 49                      'ia REST API     '
```

++ Browsing ONE message (non destructive), using -X GET

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/messaging/qmgr/QMORI/queue/Q1/message" -X GET -H "Content-Type: text/plain;charset=utf-8"
This is a test of doing MQ Put via REST API

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?status=*" -X GET | jq '.queue | .[] | .status.currentDepth'
1

++ Getting ONE message (destructive), using -X DELETE

If there are 2 messages in the queue, then the following command will ONLY get 1 message, leaving another message in the queue.
That is, it is a destructive read.
CAVEAT: There is no way to acknowledge or commit the receipt of the message so this API should only be used for applications where message loss can be tolerated.

If you repeat the command, then you will get the next message, and so on.

$ curl -s -k "https://orizaba1.fyre.ibm.com:9443/ibmmq/rest/v2/messaging/qmgr/QMORI/queue/Q1/message" -X DELETE -H "Content-Type: text/plain;charset=utf-8"
This is a test of doing MQ Put via REST API

$ curl -s -k "https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QMORI/queue/Q1?status=*" -X GET | jq '.queue | .[] | .status.currentDepth'
0

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ Chapter 6: Examples of Multi-instance queue managers
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+ Multi-instance in Linux:

Using 2 hosts:
  box1.fyre.ibm.com
  noon1.fyre.ibm.com

Multi-instance queue manager name:
  QMMI1

+ Scenario 1: Active in Host-1 box1 and Standby in Host-2 noon1.

Host-1: box1 (Active)
.
mqm@box1.fyre.ibm.com: /home/mqm
$ dspmq -x -m QMMI1
QMNAME(QMMI1)                                             STATUS(Running)
    INSTANCE(box1.fyre.ibm.com) MODE(Active)
    INSTANCE(noon1.fyre.ibm.com) MODE(Standby)
.
mqm@box1.fyre.ibm.com: /home/mqm
$ curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr -X GET -u mqadmin:mqadmin
{"qmgr": [
  {
    "name": "QMMI1",
    "state": "running"
  },
  {
    "name": "QM91",
    "state": "running"
  },
  {
    "name": "QM3L",
    "state": "running"
  },
  {
    "name": "QM4L",
    "state": "running"
  }
]}
```

```
.
.
Host-2: noon1 (Standby)
.
mqm@noon1.fyre.ibm.com: /home/mqm
$ dspmq -x -m QMMI1
QMNAME(QMMI1)                                 STATUS(Running as standby)
   INSTANCE(box1.fyre.ibm.com) MODE(Active)
   INSTANCE(noon1.fyre.ibm.com) MODE(Standby)
.
mqm@noon1.fyre.ibm.com: /home/mqm
$ curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr -X GET -u mqadmin:mqadmin
{"qmgr": [
  {
    "name": "QMMI1",
    "state": "runningAsStandBy"
  },
  {
    "name": "QMLNX2",
    "state": "endedImmediately"
  },
  {
    "name": "QM91",
    "state": "running"
  }
]}
.
.
+ Scenario 2: Switchover. None in Host-1 box1 and Active in Host-2 noon1.
.
Host-1: box1 (none)
.
Do a switchover:
.
mqm@box1.fyre.ibm.com: /home/mqm
$ endmqm -is QMMI1
IBM MQ queue manager 'QMMI1' ending.
IBM MQ queue manager 'QMMI1' ended, permitting switchover to a standby instance.

mqm@box1.fyre.ibm.com: /home/mqm
$ dspmq -x -m QMMI1
QMNAME(QMMI1)                                 STATUS(Running elsewhere)
   INSTANCE(noon1.fyre.ibm.com) MODE(Active)
.
.
Note:
```

When the Active was running in this host, the output of curl showed:

```
   "name": "QMMI1",
   "state": "running"
```

But after the instance was terminated and the switchover occurred, then the status is shown now as:

```
   "name": "QMMI1",
   "state": "runningElsewhere"
```

.

```
mqm@box1.fyre.ibm.com: /home/mqm
$) curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr -X GET -u mqadmin:mqadmin
{"qmgr": [
  {
    "name": "QMMI1",
    "state": "runningElsewhere"
  },
  {
    "name": "QM91",
    "state": "running"
  },
  {
    "name": "QM3L",
    "state": "running"
  },
  {
    "name": "QM4L",
    "state": "running"
  }
]}
```

.

Host-2: noon1 (Active)

.

After the switchover notice that the old Standby has become the Active:

.

```
mqm@noon1.fyre.ibm.com: /home/mqm
$ dspmq -x -m QMMI1
QMNAME(QMMI1)                                   STATUS(Running)
   INSTANCE(noon1.fyre.ibm.com) MODE(Active)
```

.

Note:

When the Standby was running in this host, the output of curl showed:

```
   "name": "QMMI1",
   "state": "runningAsStandBy"
```

But after the switchover occurred, then the status is shown now as:

```
   "name": "QMMI1",
   "state": "running"
```

```
mqm@noon1.fyre.ibm.com: /home/mqm
$ curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr -X GET -u mqadmin:mqadmin
{"qmgr": [
  {
    "name": "QMMI1",
    "state": "running"
  },
  {
    "name": "QMLNX2",
    "state": "endedImmediately"
  },
  {
    "name": "QM91",
    "state": "running"
  }
]}

+++ end
```