A better REST with MQ V9.1.3

gwydiontudur
Published on 30/07/2019

You're probably familiar with the MQ REST API that first appeared in MQ V9.0.1. and has been incrementally developed ever since. It allows you to easily administer MQ from new environments without the need for an MQ client.

The REST API currently has two distinct ways of administering MQ.

Firstly, there's a per-object REST API, where you call the appropriate HTTP verb on a URL that represents the object being administered. For example, a GET request on the `/admin/qmgr/{qmgrName}/queue` resource displays a list of queues on a queue manager.

The disadvantage of this is that the per-object REST API doesn't have complete coverage of all the objects and operations that you would want to perform as an MQ administrator.

Secondly, the REST API also allows you to send MQSC commands to the queue manager, in order to give you access to the whole suite of MQSC commands. Until now, this has been in the form of a plain text MQSC command surrounded by some JSON, with the command response also being returned in plain text. The command is issued with a HTTP POST request to the `/admin/action/qmgr/{qmgrName}/mqsc` URL.

Being able to issue any MQSC command through the REST API is very useful, but the disadvantage here of course is that you need to parse the command output yourself.

MQ V9.1.3 – the best of both worlds?

The improved REST API in V9.1.3 gives you the best of both worlds. You can now issue MQSC commands in JSON format through the REST API, and get the command response returned in JSON too, so you don't need to parse the MQSC command yourself.

Let's look at how you would construct an MQSC command in JSON. We'll use the command `DEFINE QLOCAL('Q1') DESCR('A test queue')` as an example. To run this command using the REST API in MQ V9.1.3 you send the following JSON in the body of the POST request to the `admin/action/qmgr/{qmgrName}/mqsc` URL:

```
{
  "type": "runCommandJSON",
  "command": "define",
  "qualifier": "qlocal",
  "name": "Q1",
  "parameters": {
    "descr": "A test queue"
  }
}
```

Notice how the MQSC command has been translated into JSON.

- The primary command keyword, DEFINE in this case, is specified in the “command” JSON attribute.
- The secondary command keyword, QLOCAL in this case, is specified in the “qualifier” JSON attribute.
- The primary argument, which is usually the name of an object that the command acts on, and in this case the name of the queue, Q1, is specified in the “name” JSON attribute.
- Any additional parameters in the MQSC command are specified in the “parameters” JSON object, with the name(value) pairs in the MQSC command becoming JSON attributes.

Every MQSC command has a primary and secondary command keyword, and so the “command” and “qualifier” JSON attributes are required. The “name” attribute and “parameters” object are optional as these only need to be specified on some MQSC commands.

The final thing to note here is the new command type of “runCommandJSON”. This is what specifies that the MQSC command is supplied in JSON format. The plain-text MQSC command function is still available on the same URL when type is specified as “runCommand”.

When this command is executed, we get the following response indicating that the queue has been defined successfully.

```
{
  "commandResponse": [
    {
      "completionCode": 0,
      "message": [
        "AMQ8006I: IBM MQ queue created."
      ],
      "reasonCode": 0
    }
  ],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```

Another example

Having used the REST API to define a queue, you can now issue a MQSC DISPLAY command to display the queue. This is how you can construct an MQSC command in JSON to display the queue that was just defined.

```
{
  "type": "runCommandJSON",
  "command": "display",
  "qualifier": "queue",
  "name": "Q1",
  "responseParameters": [
    "descr", "usage", "trigger"
  ]
}
```

Note the “responseParameters” array. This is an optional array of attributes that lists the names of the attributes we want to be returned by the DISPLAY command.

When this command is executed, we get back the following response.

```
{
  "commandResponse": [
    {
      "completionCode": 0,
      "parameters": {
        "descr": "My queue",
        "notrigger": "YES",
        "queue": "Q1",
        "type": "QLOCAL",
        "usage": "NORMAL"
      },
      "reasonCode": 0
    }
  ],
  "overallCompletionCode": 0,
  "overallReasonCode": 0
}
```

This shows you the real benefit of the improved REST API in V9.1.3. The queue definition is returned in JSON, so your application can easily parse this and does not need to parse the MQSC output.

JSON data types

Most MQSC parameters accept a value that is a string or an integer. These values are treated by the REST API as JSON strings or integers, as you would expect.
There are some MQSC parameters that are a bit more complicated however.

Some parameters accept a list of values. These are treated by the REST API as JSON arrays. For example, you could include this list of exit names on a channel definition:

```
"sendexit": ["exit1","exit2"]
```

The full list of parameters that are arrays is documented in the Knowledge Center.

There are also some MQSC parameters that don’t have a value. For example, a queue can be defined with either the TRIGGER or NOTRIGGER parameter. The example queue definition above shows you how this is handled by the REST API. Any parameter that is present, but does not have a value, is specified in the REST API as a JSON string with the value “YES”. In this example, the queue has been defined, by default, with NOTRIGGER.

Additional information

Armed with the information and examples in this article, you should now be able to try out the improved REST API for yourself, so please have a go and let us know what you think!

The JSON format MQSC REST API is documented in the Knowledge Center.

If you’re new to the MQ REST API, the Getting Started with the administrative REST API topic in the Knowledge Center is a useful guide to setting up the REST API and using it for the first time.