Independent Submission
Request for Comments: 8522
Category: Informational
ISSN: 2070-1721

M. Stubbig
Independent

February 2019

# Looking Glass Command Set

## Abstract

This document introduces a command set standard to the web-based "Network Looking Glass" software. Its purpose is to provide application programmers uniform access to the Looking Glass service and to analyze a standardized response.

The interface is supposed to provide the same level of information as web-based interfaces, but in a computer-readable format.

## Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8522.

Table of Contents



## 1. Introduction

Many Internet service providers (ISPs) and Internet exchange points (IXPs) offer a complimentary web-based service to their customers and the general public that gives insights to the backbone routing table, BGP neighbor information, or offered routes. This service is known as a "Network Looking Glass". Because they utilize a web-based interface, it is hard to automate access to the services and make that automation transferable between different service implementations.

This document describes a common command set to provide application programmers uniform access to Looking Glass services.

The commands are intended to provide the same level of information as available via web-based interfaces, but to do so in a computer-readable format. The intention is that multiple implementers of Looking Glass services can provide access through these commands so that an application can make use of the different implementations.

The command set is split into the following categories: mandatory to support, optional, and additional. The commands are extensible for new features and for value-add by implementations.

The Looking Glass command set is described as a language-independent concept. Consequently, any programming language that satisfies the commands listed in the following sections is acceptable.

This work is not the output of the IETF and is presented in the hope that Looking Glass implementers will offer a common programmable interface.

### 1.1. Background

The requirement of a uniform access to a Looking Glass service becomes important when multiple Looking Glasses are part of a monitoring system. Implementing a web client and HTTP parser for every kind of web-based Looking Glass is a time-consuming workaround. However, the Looking Glass command set is a much more viable, compatible, and scalable solution.

### 1.2. Syntax Notation

This specification uses the JavaScript Object Notation (JSON) of [RFC8259] arranged as JSend-compliant (Appendix A) responses.

### 1.3. Examples

All URLs in this documentation use the reserved sample domain of "example.net" as defined in Section 6.5 of [RFC6761].

The URLs further use the fixed [RFC5785] prefix of ".well-known/looking-glass" to prevent a collision in the domain's namespace.

IPv4 addresses use the documentation block of 192.0.2.0/24 [RFC5737] and IPv6 addresses reside in the reserved prefix of 2001:DB8::/32 [RFC3849]. BGP Autonomous System (AS) numbers are chosen from the private AS range defined in [RFC6996].

The examples skip some required parameters for reasons of simplicity.

## 2. Operation

A client issues a query using the HTTP GET method to request specific resources from the server. The resource is a URI and can be informational or a command execution. The client must present all necessary parameters for the server to execute the command on the selected router. Every call is stateless and independent of the previous one.

The path component of the resource URI must use the prefix of ".well-known/looking-glass" (see Section 5.1) to stay namespace neutral.

The "call" is a request from the client that specifies a predefined operation ("function") that the server will execute on a selected router. The "command" is a task executed on the router and initiated by the server on behalf of the client. The type and scope of all commands are defined and limited by the server. The client must not be able to execute random commands on the targeting router. There must not be any direct communication between the client and the router.

After the execution of the command on the selected router has finished, the server replies to the client if the operation has either succeeded, failed, or timed out. The response is sent to the client in JSON format. The communication protocol used between the server and router is not specified by this document; any method (e.g., Telnet, SSH, NETCONF, serial console) is acceptable.

All parameters and their values are case insensitive.

### 2.1. Method Parameters

Method parameters are mandatory components of the URI and are placed in the "path" section in terms of [RFC7320]. Basically, the method parameters specify the call and determine which command the client wants to be executed on the selected router.

## 2.2. Query Parameters

Query parameters are optional components of the URI and are placed in the "query" section in terms of [RFC7320]. Generally, the query parameters are additional instructions for the requested command.

protocol
: Restrict the command and method parameters to use the specified protocol and version. Protocol is selected as "Address Family Identifier" [IANA-AFN] [RFC4760] and optionally as "Subsequent Address Family Identifier" [IANA-SAFI] separated by a comma.
Default value is 1,1 (IP version 4, unicast).
JSON datatype is String.
Examples:

* protocol=2,1 (IP version 6, unicast)

* protocol=26 (MPLS, no SAFI used)

router
: Run the command on the router identified by its name. This is not necessarily the router's hostname as long as the Looking Glass software recognizes it.
Default value is the first router in the list of available routers.
JSON datatype is String.
Example: router=rbgn06.example.net

routerindex
: Run the command on this router identified by its position in the list of available routers.
Default value is "0".
JSON datatype is Number.
Example: routerindex=8

random
: Append a random string to prevent the client (or an intermediate proxy) from caching the response. The server must ignore its value.
No default value.
JSON datatype is String.
Example: random=517A93B50

vrf
: Run the command from the selected routing table. This parameter is valid only on routers that support "Virtual Routing and Forwarding" (VRF).
No default value.
JSON datatype is String.
Example: vrf=mgmt

runtime
: Stop executing the command after the runtime limit (in seconds) is exceeded. A value of 0 disables the limit.
Default value is "30".
JSON datatype is Number.
Example: runtime=60

format
: Request the server to provide the output (if any) in the selected format. Specify multiple formats separated by a comma in descending order of preference. See Section 3.3.2 for more details.
Default value is "text/plain" (raw/unformatted output).
JSON datatype is String.
Example: format=application/yang,text/plain

## 2.3. Response

The HTTP response header contains an appropriate HTTP status code as defined in [RFC7231] with the Content-Type set to "application/json".

The HTTP body contains details and error descriptions. The response text must comply with the JSON syntax specification JSend, which is briefly explained in Appendix A. Consequently, every response must contain a "status" field of either "success", "fail", or "error" as explained in the following sections.

### 2.3.1. Success

A successful response must set the "status" field to "success". It must also contain a "data" object including the following information:

performed_at
: Combined date and time in UTC ISO 8601 [iso8601] indicating when the operation finished. This information must be present.

runtime
: Amount of seconds (wallclock) used to run the command. This information must be present.

router
: Name of the router that executed the command. This information may be present.

output
: Output of the command in a format that was requested by the client; it otherwise defaults to raw output as it appeared on the router's command-line interface (CLI). It might even be blank if the command did not produce any output. This information should be present.

format
: Selected output format by the server. The client might request multiple formats so that the "Looking Glass" server has to choose the best option and tell the client which format was selected. This information should be present (defaults to "text/plain" if missing).

Adding more information to the response is permitted and must be placed inside the "data" object.

The HTTP status code should be 200.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "status" : "success",
  "data" : {
    "router" : "route-server.lookingglass.example.net"
    "performed_at" : "2014-10-15T17:15:34Z",
    "runtime" : 2.63,
    "output" : [
      "full raw output from the observing router..."
    ],
    "format" : "text/plain"
  }
}
```

### 2.3.2. Fail

A status of "fail" indicates that the selected command was executed on the router but failed to succeed. The response message must set the "status" field to "fail" and must contain the "data" object with command-specific content listed in Section 2.3.1.

The HTTP status code should be 200.

Example:

```
HTTP/2.0 200 OK
{
  "status" : "fail",
  "data" : {
    "performed_at" : "2014-10-18T20:04:37Z",
    "runtime" : 10.37,
    "output" : [
      "Sending 5, 100-byte ICMP Echos to 192.0.2.5",
      ".....",
      "Success rate is 0 percent (0/5)"
    ],
    "format" : "text/plain",
    "router" : "route-server.lookingglass.example.net"
  }
}
```

### 2.3.3. Error

The status "error" represents either that the command timed out or that an error occurred in processing the request. The response message must set the "status" field to "error" and must contain the "message" key, which keeps a meaningful message, explaining what went wrong.

The response may contain the "data" key with required values listed in Section 2.3.1. It may also include a "code" field that carries a numeric code corresponding to the error.

The HTTP status code should be 400 in case of a client-side error, 500 in case of a server-side error, or 502 for errors occurring on the target router. Code 504 should be used when a command timed out.

Example:

```
HTTP/1.1 400 Bad Request
{
  "status" : "error",
  "message" : "Unrecognized host or address."
}
```

## 3. Functions

The Looking Glass command set provides functions that are either mandatory to support or optional to implement. The same principle applies to the web-based Looking Glass.

It is not possible for any function to modify the server's state. Therefore, all HTTP methods are GET operations.

Variables are templated and expanded in accordance with [RFC6570].

### 3.1. Diagnostic Commands

#### 3.1.1. Ping

Send echo messages to validate the reachability of a remote host and measure round-trip time. The host can be a name or address.

Implementation of the ping command is mandatory.

Syntax: https://example.net/.well-known/looking-glass/v1/ping/{host}

Example query:

```
GET /.well-known/looking-glass/v1/ping/2001:DB8::35?protocol=2,1
Host: example.net
```

Example response:

```
HTTP/1.1 200 OK
{
  "status" : "success",
  "data" : {
    "min" : 40,
    "avg" : 41,
    "max" : 44,
    "rate" : 100,
    "output" : [
      "Sending 5, 100-byte ICMP Echos to 2001:DB8::35",
      "!!!!!",
      "Success rate is 100 percent (5/5)"
    ],
    "format" : "text/plain",
    "performed_at" : "2014-10-04T14:40:58Z",
    "runtime" : 0.77,
    "router" : "c2951.lab.lg.example.net"
  }
}
```

#### 3.1.2. Traceroute

Trace the path from the executing router to the destination host and list all intermediate hops. The host can be a name or address.

Implementation of the traceroute command is optional.

Syntax: https://example.net/.well-known/looking-glass/v1/traceroute/{host}

Example query:

```
GET /.well-known/looking-glass/v1/traceroute/192.0.2.8?routerindex=5
Host: example.net
```

Example response:

```
HTTP/1.1 200 OK
{
  "status": "success",
  "data": {
    "output": [
      "Tracing the route to 192.0.2.8",
      "",
      "  1 198.51.100.77 28 msec 28 msec 20 msec",
      "  2 203.0.113.130 52 msec 40 msec 40 msec",
      "  3 192.0.2.8 72 msec 76 msec 68 msec"
    ],
    "format": "text/plain",
    "performed_at": "2018-06-10T12:09:31Z",
    "runtime": 4.21,
    "router": "c7206.lab.lg.example.net"
  }
}
```

### 3.2. Informational Commands

#### 3.2.1. show route

Retrieve information about a specific subnet from the routing table.

Implementation of the "show route" command is mandatory.

Syntax: https://example.net/.well-known/looking-glass/v1/show/route/{addr}

Example query:

```
GET /.well-known/looking-glass/v1/show/      [multiline]
        route/2001:DB8::/48?protocol=2,1
Host: example.net
```

Example response:

```
HTTP/1.1 200 OK
{
  "status": "success",
  "data": {
    "output": [
      "S   2001:DB8::/48 [1/0]",
      "     via FE80::C007:CFF:FED9:17, FastEthernet0/0"
    ],
    "format": "text/plain",
    "performed_at": "2018-06-11T17:13:39Z",
    "runtime": 1.39,
    "router": "c2951.lab.lg.example.net"
  }
}
```

### 3.2.2. show bgp

Display a matching record from the BGP routing table. This should include networks, next hop, and may include metric, local preference, path list, weight, etc.

Implementation of the "show bgp" command is optional.

Syntax: https://example.net/.well-known/looking-glass/v1/show/bgp/{addr}

Example query:

```
GET /.well-known/looking-glass/v1/show/bgp/192.0.2.0/24
Host: example.net
```

Example response:

```
HTTP/1.1 200 OK
{
  "status": "success",
  "data": {
    "output": [
      "BGP routing table entry for 192.0.2.0/24, version 2",
      "Paths: (2 available, best #2, table default)",
      "  Advertised to update-groups:",
      "     1",
      "  Refresh Epoch 1",
      "  Local",
      "    192.0.2.226 from 192.0.2.226 (192.0.2.226)",
      "      Origin IGP, metric 0, localpref 100, valid, internal",
      "[...]"
    ],
    "format": "text/plain",
    "performed_at": "2018-06-11T21:47:17Z",
    "runtime": 2.03,
    "router": "c2951.lab.lg.example.net"
  }
}
```

### 3.2.3. show bgp summary

Print a summary of BGP neighbor status. This may include the neighbor BGP ID, autonomous system number, duration of peering, number of received prefixes, etc.

Implementation of the "show bgp summary" command is optional.

Syntax: https://example.net/.well-known/looking-glass/v1/show/bgp/summary

Example:

```
GET /.well-known/looking-glass/v1/show/bgp/summary?     [multiline]
       protocol=2&routerindex=3
Host: example.net
```

Example response:

```
HTTP/1.1 200 OK
{
  "status": "success",
  "data": {
    "output": [
      "BGP router identifier 192.0.2.18, local AS number 64501",
      "BGP table version is 85298, main routing table version 85298",
      "50440 network entries using 867568 bytes of memory",
      "[...]",
      "Neighbor        V       AS MsgRcvd MsgSent   TblVer  Up/Down",
      "2001:DB8:91::24 4    64500  481098  919095   85298   41w5d"
    ],
    "format": "text/plain",
    "performed_at": "2018-06-11T21:59:21Z",
    "runtime": 1.91,
    "router": "c2951.lab.lg.example.net"
  }
}
```

### 3.2.4. show bgp neighbors

Provide detailed information on BGP neighbor connections. Available details may include neighbor BGP ID, advertised networks, learned networks, autonomous system number, capabilities, protocol, statistics, etc.

Implementation of the "show bgp neighbors" command is optional.

Syntax: https://example.net/.well-known/looking-glass/v1/show/bgp/neighbors/{addr}

Example query:

```
GET /.well-known/looking-glass/v1/show/bgp/neighbors/192.0.2.226
Host: example.net
```

Example response:

```
HTTP/1.1 200 OK
{
  "status": "success",
  "data": {
    "output": [
      "BGP neighbor is 192.0.2.226, remote AS 64500, internal link",
      "  BGP version 4, remote router ID 198.51.100.31",
      "  BGP state = Established, up for 01:24:06",
      "[...]"
    ],
    "format": "text/plain",
    "performed_at": "2018-06-11T21:41:17Z",
    "runtime": 1.87,
    "router": "c2951.lab.lg.example.net"
  }
}
```

### 3.3. Organizational Commands

The following organizational commands must be included in the implementation.

#### 3.3.1. router list

Provides a full list of routers that are available for command execution. This list includes the router ID and its name. It is equivalent to the common "router" HTML drop-down form element and contains the same information.

Syntax: https://example.net/.well-known/looking-glass/v1/routers

Example response:

```
{
  "status" : "success",
  "data" : {
    "routers" : [
      "route-server.lookingglass.example.net",
      "customer-edge.lookingglass.example.net",
      "provider-edge.lookingglass.example.net"
    ],
    "performed_at" : "2014-10-19T20:07:01Z",
    "runtime" : 0.73
  }
}
```

### 3.3.2. router details

Lists additional information about the selected router specified by its router index. The response must contain the router's hostname and router index. The response may contain more details like output format, country code, city, administrative contact, vendor, and model.

Available output formats are specified by Internet media type as of [RFC6838] and listed in [IANA-MT]. If the routers support multiple formats, they are separated by a comma.

The router might provide output formats that are not yet registered or listed in [IANA-MT]. For example, output in NETCONF format could use "text/x.netconf". [RFC6838] provides a tree for unregistered subtypes.

A missing output format defaults to "text/plain", which is a copy of the raw command-line output.

Syntax: https://example.net/.well-known/looking-glass/v1/routers/{number}

Example query:

```
GET /.well-known/looking-glass/v1/routers/1
Host: example.net
```

Example response:

```
{
  "status" : "success",
  "data" : {
    "id" : 1,
    "name" : "customer-edge.lookingglass.example.net",
    "format" : "text/plain,text/x.netconf",
    "country" : "de",
    "autonomous_system" : 64512
  }
}
```

### 3.3.3. commands

Provides a full list of commands that are available for execution. The list includes mandatory to support, optional, and additional (Section 3.4) commands. It is equivalent to the "command" HTML drop-down or radio-button form element and contains the same information.

The list is formatted as a "commands" array containing one object per command. This object contains informative strings about the current command: href, arguments, description, and command.

Syntax: https://example.net/.well-known/looking-glass/v1/cmd

Example response:

```
{
  "status" : "success",
  "data" : {
    "commands" : [
      {
        "href" : "https://example.net/.well-known/     [multiline]
                          looking-glass/v1/show/route",
        "arguments" : "{addr}",
        "description" : "Print records from IP routing table",
        "command" : "show route"
      },
      {
        "href" : "https://example.net/.well-known/     [multiline]
                          looking-glass/v1/traceroute",
        "arguments" : "{addr}",
        "description" : "Trace route to destination host",
        "command" : "traceroute"
      }
    ]
  }
}
```

### 3.4. Extensible Commands

The list of commands discussed in Section 3.3.3 may be expanded as long as the principles of this document are observed.

For example, a Looking Glass provider may not be offering BGP-related commands because of an OSPF-based network.

The sample command might be:

```
GET /.well-known/looking-glass/v1/show/ospf/database
Host: example.net
```

## 4. Miscellaneous

The network traffic sent by a "Looking Glass" is not appropriate when measuring Service Level Agreements or validating Quality of Service settings.

If a monitoring system uses the Looking Glass command set for reachability checks, it should not rely on the HTTP status codes but on the "status" message field inside the HTTP body.

## 5. IANA Considerations

### 5.1. Well-Known URIs Registry

This specification registers a Well-Known URI [RFC5785]:

URI Suffix: looking-glass

Change Controller: M. Stubbig

Reference : This document, Section 2

## 6. Security Considerations

The use of HTTPS is required to ensure a high level of security, privacy, and confidentiality during transit.

### 6.1. Abuse Potential

The main goal of the Looking Glass command set is the automated usage of the Looking Glass service. This allows the scripting of API calls, which could be used as a Distributed Denial of Service (DDoS) attack. It is recommended that implementers of the Looking Glass API take steps to mitigate the above described abuse. The strategy can include blocking or rate-limiting by client IP address or target IP network.

### 6.2. Authentication

Authentication is not a requirement because the current Looking Glass web services are usable without authentication. Requests to the proposed API service may be authenticated by any method. The decision is up to the implementer’s security requirements.

### 6.3. Minimal Information

Some of the described commands provide a detailed insight into the provider’s network. It is therefore up to the implementer’s security policy to dismiss commands that are marked as "optional" or to restrict commands that are marked as "mandatory".

# 7. References

## 7.1. Normative References

[IANA-AFN] IANA, "Address Family Numbers", <https://www.iana.org/assignments/address-family-numbers/>.

[IANA-MT] IANA, "Media Types", <https://www.iana.org/assignments/media-types/>.

[IANA-SAFI] IANA, "Subsequent Address Family Identifiers (SAFI) Parameters", <https://www.iana.org/assignments/safi-namespace/>.

[JSend] OmniTI Labs, "JSend", 2014, <https://labs.omniti.com/labs/jsend>.

[RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, DOI 10.17487/RFC4760, January 2007, <https://www.rfc-editor.org/info/rfc4760>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

### 7.2. Informative References

[iso8601] International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", December 2004.

[RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/info/rfc3849>.

[RFC5737] Arkko, J., Cotton, M., and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation", RFC 5737, DOI 10.17487/RFC5737, January 2010, <https://www.rfc-editor.org/info/rfc5737>.

[RFC6761] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", RFC 6761, DOI 10.17487/RFC6761, February 2013, <https://www.rfc-editor.org/info/rfc6761>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6996] Mitchell, J., "Autonomous System (AS) Reservation for Private Use", BCP 6, RFC 6996, DOI 10.17487/RFC6996, July 2013, <https://www.rfc-editor.org/info/rfc6996>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <https://www.rfc-editor.org/info/rfc7320>.

## Appendix A. JSend

According to [JSend]:

> JSend is a specification that lays down some rules for how JSON responses from web servers should be formatted. JSend focuses on application-level (as opposed to protocol- or transport-level) messaging which makes it ideal for use in REST-style applications and APIs.

A basic JSend-compliant response must contain a "status" key and should contain "data", "message", and "code" keys dependent on the status value. The following table lists the required and optional keys.

| Type | Required keys | Optional keys |
|---|---|---|
| success | status, data | |
| fail | status, data | |
| error | status, message | code, data |

Table 1: Type and Keys in JSend Response

## Author's Address

Markus Stubbig
Independent
Germany

Email: stubbig.ietf@gmail.com