ICANN Monitoring System API (MoSAPI)

Document Version 3.0.0
2022-10-27

Document Revision History

Version	Publishing date	Description of the change	Projected date to implement the version of the specification in production
2.5	2017/11/28	First version released to the public.	In production
2.6	2018/02/12	ADDITION - Default rate-limit and expiration date values were added in section 3.1. ADDITION - Maximum length definitions for name and description were added in section 6.1.1. ADDITION - Result code 2016 (section 8) was added to the result code table. CHANGE - Result code 2007 message (section 8) was changed to cover the case of equal values in the endTime and startTime.	In production
2.7	2018/03/06	MoSAPI released as a production service.	In production
2.8	2018/10/02	ADDITION - UP-inconclusive-no-data and UP-inconclusive-no-probes were added to sections 5.1 and 5.8. CHANGE - Error codes changed in sections 5.8.1 and 5.8.2. CHANGE - Editorial updates.	Never released to production, replaced with version 2.8.1
2.8.1	2018/11/26	CHANGE - Minor adjustments to the list of error codes in sections 5.8.1 and 5.8.2.	Never released to production, replaced with version 2.8.2
2.8.2	2019/02/21	CHANGE - Error codes changed in sections 5.8.1.	2019/02/21
2.9	2019/03/25	ADDITION - Section 9 and 10 were added.	2019/04/30
2.10	2019/05/30	ADDITION – Section 11. CHANGE – Editorial updates.	2019/05/30
2.11	2020/01/31	ADDITION – Section 12. CHANGE – Editorial updates.	Never released to production, replaced with version 2.11.1
2.11.1	2020/02/26	ADDITION – Additional details in Section 12. CHANGE – Rate-limit in Section 3.1. ADDITION – Section 1.4.	Never released to production, replaced with version 2.11.2
2.11.2	2020/06/29	ADDITION – Additional details in Section 12. CHANGE – Response message in Section 4. CHANGE – Technical Requirements in Section 1.4.	July 2020
3.0.0	2022/10/12	CHANGE – Section 2 now describes the <base_url> for both /v1 and /v2 APIs. CHANGE – Section 3 now describes session management for both /v1 and /v2 APIs. CHANGE – Sections 5.1 through 5.7 show /v2 API examples. CHANGE – Sections 5.8 shows new /v2 JSON items and /v2 API examples.	October 2022

		ADDITION – Section 5.9 describes the new Soon to Be Revoked flag. CHANGE – Section 6 has been deprecated. CHANGE – Section 7 has been deprecated. CHANGE – Section 8 removes endpoints that have been deprecated. CHANGE – Section 11 has been deprecated. CHANGE – The example URLs in Section 9 have been changed to new /v1 URLs. CHANGE – Section 12 has been updated.	

1. Introduction

This document describes the REST API endpoints provided by ICANN to contracted parties and Country Code Top-Level Domain (ccTLD) registries in order to retrieve monitoring and other information that is intended to be available only to the specific registry or registrar.

1.1. Date and Time

All the fields that represent dates in this document must contain timestamps indicating the date and time in Coordinated Universal Time (UTC).

1.2. Credentials

The MoSAPI uses the same credentials (e.g., username, password, list of IP address blocks - IPv4 and/or IPv6) as the Registration Reporting Interface (RRI). These credentials are managed through the NSP portal provided by ICANN to the contracted parties or through email in the case of ccTLD registries.

The MoSAPI supports both IPv4 and IPv6 transport.

The MoSAPI requires the use of Hypertext Transfer Protocol Secure (HTTPS) to provide confidentiality, server authentication, and integrity in the communication channel.

1.3. Glossary

In the following section, the concepts used in the MoSAPI are explained:

- **Service**: a service that may be monitored by the MoSAPI. The potential monitored services are: dns, rdds, epp and dnssec.
- **Test Cycle**: series of tests executed to verify the state of a monitored Service. For Domain Name System (DNS), the Service is considered to be up at a particular moment, if at least two of the delegated name servers registered in the DNS have successful results from tests to each of their public-DNS registered IP addresses in the root zone for the name server. For the Registration Data Directory Services Requirements (RDDS) Services (i.e. Whois TCP/43 and web-Whois) to be considered up at a particular moment, the Services must have successful results from tests to the randomly chosen public-DNS registered IP address to which whois.nic.<TLD> resolves. If 51% or more of the testing probe nodes see a monitored Service as unavailable at a given time, the Service will be considered unavailable. For RDDS, if any of the RDDS Services (i.e., Whois TCP/43 and web-Whois) is considered unavailable, the RDDS will be considered unavailable. The minimum number of active testing probe nodes to consider the results of a test cycle as valid at any given time is 20 for DNS and 10 for RDDS; otherwise, the test cycle results will be discarded, and the Service will be considered up.
- **Test**: for DNS it means one non-recursive DNS query sent to a particular IP address via UDP or TCP; if DNSSEC is offered in the queried DNS zone, for a query to be considered answered, the signatures must be positively verified against a corresponding DS record published in the parent zone. For RDDS it means one query sent to a particular IP address. The answer to the query must contain the corresponding information from the Registry System, otherwise the query will be considered unanswered. A query with a RTT higher than X milliseconds will also be considered unanswered. For DNS (UDP) X=2,500 ms, DNS (TCP) X=7,500 ms for RDDS X=10,000 ms.
- **RTT (Round Trip Time)**: for DNS/UDP, the sequence of two packets, the UDP DNS query and the corresponding UDP DNS response. For DNS/TCP, the sequence of packets from the start of the TCP connection to its end. For Whois TCP/43, the sequence of packets from the start of the TCP connection

to its end, including the reception of the Whois TCP/43 response. For web-Whois, the sequence of packets from the start of the TCP connection to its end, including the reception of a HTTP response; if the Registry Operator implements HTTP URL redirection (e.g., HTTP 302), only the last HTTP transaction is measured.

- **Emergency Threshold**: downtime threshold that if reached by any of the monitored Services may cause the TLD's Services emergency transition to an interim Registry Operator. To reach an Emergency Threshold a Service must accumulate X hours of total downtime during the last 7 days (i.e., rolling week). For DNS X=4, for RDDS X=24.

- **Incident**: an Incident is the collection of measurements from the moment an Alarm is generated until the moment that the Alarm is cleared. An Incident can have 2 distinct states:

  · Active: measurements corresponding to a current downtime.
  · Resolved: measurements corresponding to past downtime.

  The measurements of Incidents that occurred in the last 7 days (i.e., rolling week, from: the current date and time -7days, to: the current date and time) are considered for the Service's Emergency Threshold calculations.

- **Alarm**: an Alarm signals that a Service has been detected as being down because X consecutive test cycles with Y minutes between them failed. An Alarm is cleared when the Service is detected as being up because X consecutive test cycles with Y minutes between them have been successful. For DNS, X=3 and Y=1. For RDDS, X=2 and Y=5. An alarmed Service triggers the creation of an Incident; if the Alarm is cleared then the Incident will be marked as resolved.

- **False Positive**: a flag set to an Incident indicating that an Incident was found by a manual process to be a false positive. When an Incident is marked as a False Positive the measurements of the Incident are not used for the Emergency Threshold calculations.

1.4. Technical requirements

- Clients shall send the HTTP Cookies provided by the server on every HTTP request when `<base_url>/login` was used to create a session.

2. Common elements used in this specification

In the following section, common elements used in this specification are explained:

- **<base_url>:** the base URL of the MoSAPI is https://mosapi.icann.org/ry/<tld>/<version>, for example: https://mosapi.icann.org/ry/example/v1/monitoring/state

  Where:

  - <tld> must be substituted by the TLD being queried. In case of an IDN TLD, the A-label must be used.
  - <version> must be substituted by the version number of the specification supported by the server. For this specification the possible values are ‘v1’ and ‘v2’. When using the “login” and “logout” APIs, <version> is omitted.

- **<service>** must be substituted by the Service being queried. The possible values of Service, as described in Section 1 of gTLD Base Registry Agreement - Specification 10, are: dns, dnssec, rdds, and epp.

  Note: EPP is not currently being monitored.

The <base_url> of https://mosapi.icann.org/mosapi/<version>/<tld> will be deprecated and future use of it is discouraged. The change in the <base_url> facilitates the use of the /login and /logout session-management APIs with both ‘v1’ and ‘v2’ APIs. See Section 3 regarding the use of /login and /logout with both ‘v1’ and ‘v2’ APIs.

Additionally, ‘v1’ APIs will be deprecated in the future, therefore use of the ‘v2’ APIs is encouraged.

3. Session handling

The MoSAPI provides two endpoints for session handling, the authentication mechanism is HTTP Basic Access Authentication as specified in RFC 2617.

Authentication credentials for the API endpoints are provided by ICANN per TLD. The credentials must only be used when creating a session using the <base_url>/login API endpoint described in this section. Note that the <base_url> does not include the version component when used with /login or /logout (see Section 2).

3.1. Creating a session

```
<base_url>/login
```

Possible results:

- HTTP/401, the <base_url>/login API endpoint provides a HTTP/401 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Invalid credentials" when the authentication credentials are invalid.

- HTTP/403, the <base_url>/login API endpoint provides a HTTP/403 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Your IP address is not allowed to connect for this TLD" if the credentials are valid but the connecting IP address is not allowed for the specified <tld>.

- HTTP/429, the <base_url>/login API endpoint provides a HTTP/429 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "You reached the limit of login requests per minute" for the specified <tld>.

  Note: Only connections originating from IP addresses allowed for the <tld> counts towards the limit. Connections originating from IP addresses not included in the allowlist are dropped without further validation. Currently, the rate-limit allows for one login request every 300s per <tld>. Developers are encouraged to re-use the session to minimize the number of login requests.

- HTTP/200, when a valid request is received, the <base_url>/login API endpoint provides an HTTP/200 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Login successful". The HTTP header Set-Cookie is set with the cookie attributes "id=<sessionID>; expires=<date>; path=**<base_url>**; secure; httpOnly".

  - The <sessionID> value is a 160-bit random value encoded in Base16.
  - The <date> value is the expiration date of the session.

Example using curl (https://curl.haxx.se/) for a login request:

```
curl --cookie-jar cookies.txt --user user:passwd https://mosapi.icann.org/ry/example/login
```

After the login, the session cookie will be valid for the path /ry/example and all sub-paths. Following a successful login, either a ‘v1’ or ‘v2’ API endpoint maybe issued. For example:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v1/monitoring/state
```

or

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/state
```

Note: Every time the <base_url>/login API endpoint successfully validates the credentials and origin IP address, a new session is created. Only one concurrent session is permitted per account. A session is only terminated after its expiration date (by default, the value of expiration date is 15 minutes after the session is created), by using the <base_url>/logout API endpoint, or if the session is the oldest and a new session is being created that would be above the limit of permitted concurrent sessions.

3.2. Closing a session

```
<base_url>/logout
```

In order to destroy a session, the client must set the HTTP header Cookie with the value "id=<sessionID>", where <sessionID> must be a 160-bit random value provided in the HTTP server response of a successful /login request. If multiple cookies are provided, the first cookie is used for destroying the session.

Possible results:

- HTTP/401, the <base_url>/logout API endpoint provides a HTTP/401 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Invalid session ID" when the specified <sessionID> is invalid.

- HTTP/403, the <base_url>/logout API endpoint provides a HTTP/403 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Your IP address is not allowed to connect for this TLD" if the specified <sessionID> is valid but the connecting IP address is not allowed for the specified <tld>.

- HTTP/200, when a valid request is received, the <base_url>/logout API endpoint provides a HTTP/200 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Logout successful". The HTTP header Set-Cookie is set with the values "id=; expires=<date>; path=**<base_url>**; secure; httpOnly".

    - The <date> value is set to the Unix epoch date and time.
    - The <version> value must be 'v1'.
    - The <tld> value is the TLD being queried.

Example using CURL for a logout request:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/logout
```

4. API endpoint authentication

When sending a request to the MoSAPI, the client must set the HTTP header Cookie with the value "id=<sessionID>", where <sessionID> must be the 160-bit random value provided in the last HTTP server response of a successful "login" request. If multiple cookies are provided, the first cookie is used for validating the session.

The following responses may be provided by the API endpoints shown below:

- HTTP/401, the API endpoint provides a HTTP/401 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "The client could not be authenticated using any of the available methods: TLS-Client-Authentication or Session Cookie" when the specified <sessionID> is invalid.

- HTTP/403, the API endpoint provides a HTTP/403 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Your IP address is not allowed to connect for this TLD" if the specified <sessionID> is valid but the connecting IP address is not allowed for the specified <tld>.

5. gTLD Base Registry Agreement - Specification 10 monitoring

Registries may access the monitoring information collected by the SLA Monitoring system using the GET HTTP verb in the MoSAPI endpoints described below. The monitoring information will be refreshed at least every 2 minutes.

5.1. Monitoring the state of a TLD

```
<base_url>/monitoring/state
```

Possible results:

- HTTP/200, when a valid request is received, the <base_url>/monitoring/state API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "tld", a JSON string that contains the monitored TLD.

  - "status", a JSON string that contains the status of the TLD as seen from the monitoring system. The "status" field may contain one of the following values:

    · Up: all of the monitored Services are up.
    · Down: one or more of the monitored Services are down.
    · Up-inconclusive: the SLA monitoring system is under maintenance, therefore all the monitored Services of the TLD are considered to be up by default. Note: if the status is "Up-inconclusive", all Services in the "testedServices" array will have the status with a value of "disabled".

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "testedServices", a JSON array that contains detailed information for each potential monitored service (i.e., DNS, RDDS, EPP, DNSSEC). Each <service> object contains the following fields:

    - "status", a JSON string that contains the status of the Service as seen from the monitoring system. The "status" field can contain one of the following values:

      · Up: the monitored Service is up.
      · Down: the monitored Service is down.
      · Disabled: the Service is not being monitored.
      · UP-inconclusive-no-data: indicates that there are enough probe nodes online, but not enough raw data points were received to make a determination.
      · UP-inconclusive-no-probes: indicates that there are not enough probe nodes online to make a determination.
      · UP-inconclusive-reconfig: indicates that the system is undergoing a reconfiguration for the particular TLD and service.

    - "emergencyThreshold", a JSON number that contains the current percentage of the Emergency Threshold of the Service. Note: the value "0" specifies that the are no Incidents affecting the Emergency Threshold of the Service.

- "incidents", a JSON array that contains "incident" objects. The "incident" object contains:
  - "incidentID", a JSON string that contains the Incident identifier (i.e., <incidentID>). The Incident identifier (i.e., <incidentID>) is a concatenation of the Unix time stamp of the start date and time of the Incident, followed by a full stop (".", ASCII value 0x002E), followed by the monitoring system identifier.
  - "startTime", a JSON number that contains the Unix time stamp of the start date and time of the Incident.
  - "falsePositive", a JSON boolean value that contains true or false with the False Positive status of the Incident.
  - "state", a JSON string that contains the current state (i.e., Active or Resolved) of the Incident.
  - "endTime", a JSON number that contains the Unix time stamp of the end date and time of the Incident; if the Incident state is active the "endTime" field will contain a null value.

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the state of a TLD:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/state
```

Example of a JSON response for a TLD state request:


```
{
    "tld": "example",
    "lastUpdateApiDatabase": 1496923082,
    "status": "Down",
    "testedServices": {
            "DNS": {
                    "status": "Down",
                    "emergencyThreshold": 10.0000,
                    "incidents": [{
                            "incidentID": "1495811850.1700",
                            "endTime": null,
                            "startTime": 1495811850,
                            "falsePositive": false,
                            "state": "Active"
                    }]
            },
            "DNSSEC": {
                    "status": "Down",
                    "emergencyThreshold": 10.0000,
                    "incidents": [{
                            "incidentID": "1495811790.1694",
                            "endTime": null,
                            "startTime": 1495811790,
                            "falsePositive": false,
                            "state": "Active"
                    }]
            },
            "EPP": {
                    "status": "Disabled"
            },
            "RDDS": {
                    "status": "Disabled"
            }
    },
    "version": 2
}
```

5.2. Monitoring the Alarm status of a Service

```
<base_url>/monitoring/<service>/alarmed
```

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/alarmed API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/alarmed API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "alarmed", a JSON string that contains one of the following values:

    · Yes: an Alarm exists for the Service.
    · No: an Alarm does not exist for the Service.
    · Disabled: the Service is not being monitored.

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the Alarm status of a Service:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/dns/alarmed
```

Example of a JSON response for a Service in Alarm status:

```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "alarmed": "Yes"
}
```

5.3. Monitoring the availability of a Service

```
<base_url>/monitoring/<service>/downtime
```

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/downtime API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/downtime API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "downtime", a JSON number that contains the number of minutes of downtime of the Service during a rolling week period.

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the availability of a Service:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/dns/downtime
```

Example of a JSON response for a Service availability request:

```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "downtime": 935
}
```

5.4. Query a list of Incidents for a Service

```
<base_url>/monitoring/<service>/incidents?startDate=<startDate>&endDate=<endDate
>&falsePositive=<falsePositive>
```

Where:

- Optional <startDate> to be substituted by the Unix time stamp of the 'after' date and time to filter by. The filter will match Incidents that started after the provided date and time.

- Optional <endDate> to be substituted by the Unix time stamp of the 'before' date and time to filter by. The filter will match Incidents that started before the provided date and time.

- Optional <falsePositive> to be substituted by “true” or “false” in order to filter the Incidents marked as False Positive. If its value equals “true”, only Incidents marked as False Positive will be returned. If its value equals “false”, only Incidents not marked as False Positive will be returned. If <falsePositive> is not defined, all Incidents will be returned.

Note: The <base_url>/monitoring/<service>/incidents supports a maximum of 31 days difference between <startDate> and <endDate>. If only <startDate> is provided, the API endpoint will return results that are within 31 days after the date and time provided. If only <endDate> is provided, the API endpoint will return results that are within 31 days before the date and time provided. If neither <startDate> nor <endDate> are provided, the API endpoint will return results that are within 31 days before the current date and time. If <endDate> is in the future, the value of <endDate> will be the current date and time.

Possible results:

- HTTP/400, see section 8.

- HTTP/404, the <base_url>/monitoring/<service>/incidents API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/incidents API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "incidents", JSON array, see definition in section 5.1.

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request a list of Incidents of a Service:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/incidents?startDate=1422492400&endDate=1422493000
```

Example of a JSON response showing a list of Incidents:


```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "incidents": [
    {
      "incidentID": "1422492450.699",
      "startTime": 1422492450,
      "falsePositive": false,
      "state": "Active",
      "endTime": null
    },
    {
      "incidentID": "1422492850.3434",
      "startTime": 1422492850,
      "falsePositive": true,
      "state": "Resolved",
      "endTime": 1422492950
    }
  ]
}
```

5.5. Monitoring the state of a particular Incident

```
<base_url>/monitoring/<service>/incidents/<incidentID>/state
```

Where:

- <incidentID> must be substituted by the Incident id assigned by the monitoring system.

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/incidents/<incidentID>/state API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <incidentID> does not exist or if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/incidents/<incidentID>/state API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "incidents", JSON array, see definition in section 5.1.

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the state of an Incident:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/incidents/1422492450.699/state
```

Example of a JSON response for an Incident state request:

```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "incidents": [
    {
      "incidentID": "1422492450.699",
      "startTime": 1422492450,
      "falsePositive": false,
      "state": "Active",
      "endTime": null
    }
  ]
}
```

5.6. Monitoring the False Positive flag of an Incident

```
<base_url>/monitoring/<service>/incidents/<incidentID>/falsePositive
```

Where:

- <incidentID> must be substituted by the Incident id assigned by the monitoring system.

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/incidents/<incidentID>/falsePositive API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <incidentID> does not exist or if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/incidents/<incidentID>/falsePositive API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "falsePositive", a JSON boolean value that contains true or false with the False Positive status of the Incident. The default value is false.

  - "updateTime", a JSON number that contains the Unix time stamp of the date and time the False Positive status was updated; if the False Positive status has never been updated the "updateTime" field will contain a null value.

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the False Positive flag of an Incident:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/incidents/1422492930.699/falsePositive
```

Example of a JSON response for an Incident flagged as False Positive:

```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "falsePositive": true,
  "updateTime": 1422494780
}
```

Note: The False Positive flag is the only thing that may change after an Incident is resolved. The user MAY be notified if an Incident is marked as a false positive by an offline mechanism.

5.7. Querying the list of measurements for an Incident

```
<base_url>/monitoring/<service>/incidents/<incidentID>
```

Where:

- <incidentID> must be substituted by the Incident id assigned by the monitoring system.

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/incidents/<incidentID> API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <incidentID> does not exist or if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/incidents/<incidentID> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:

  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.

  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.

  - "measurements", a JSON array that contains a list of <measurementID> values assigned by the monitoring system. A <measurementID> is a concatenation of the Unix time stamp of the date and time when the measurement was computed, followed by a full stop (".", ASCII value 0x002E), followed by a random value, followed by a full stop (".", ASCII value 0x002E), followed by the string "json" (ASCII value, 0x006A + 0x0073 + 0x006F + 0x006E).

Differences between v1 and v2:

There are no differences between /v1 and /v2 versions of this API endpoint. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the list of measurements of an Incident:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/incidents/1422492930.699
```

Example of a JSON response showing a list of measurements identifiers:

```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "measurements": [
    "1422492930.699.json",
    "1422492990.699.json",
    "1422493050.699.json",
    "1422493110.699.json"
  ]
}
```

5.8. Querying the details of a particular measurement

```
<base_url>/monitoring/<service>/incidents/<incidentID>/<measurementID>
```

Where:

- <incidentID> must be substituted by the Incident id assigned by the monitoring system.
- <measurementID> must be substituted by the measurement id assigned by the monitoring system.

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/incidents/<incidentID>/<measurementID> API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <incidentID> does not exist, the specified <measurementID> does not exist or if the specified <service> is not being monitored.
- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/incidents/<incidentID>/<measurementID> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:
  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.
  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.
  - "tld", a JSON string that contains the monitored TLD.
  - "service", a JSON string that contains the Service being queried. The possible values of Service, as described in Section 1 of Specification 10, are: dns, dnssec, rdds, and epp.
  - "cycleCalculationDateTime", a JSON number that contains the date and time the test cycle results were computed.
  - “minNameServersUp”, a JSON number indicating the minimum number of nameservers required to be up for the cycle to be considered up. This item is only returned from the /v2 API endpoint and the “dns” and “dnssec” services.
  - “nameServerAvailability”, a JSON object only returned from the /v2 API endpoint and the “dns” and “dnssec” service containing the following members:
    - “nameServerStatus”, a JSON array containing JSON objects, each describing the status of a specific nameserver. This item is only returned in the /v2 API endpoint and the services “dns” and “dnssec”. Each object has the following members:
      - “status”, a JSON string that contains the status of the nameserver. It will have the value “Up” if the nameserver is up, and “Down” if the nameserver is down.
      - “target”: a JSON string holding the fully-qualified domain name of the nameserver.
    - “probes”, a JSON array containing JSON objects, each describing a probe used in the measurement. This item is only returned in the /v2 API endpoint. Each object has the members:
      - “city”, a JSON string containing the name of the city where the probe is located.
      - “testData”, a JSON array with the following JSON objects:

- “target”, a JSON string containing the name of the fully-qualified domain name of the host being measured.
- “status”, a JSON string the contains the status of the target being measured. It will have the value “Up” if the target is up, and “Down” if the target is down.

- "status", a JSON string that contains the status of the Service after computing the test cycle results. The "status" field can contain one of the following values:
  - Up: the monitored Service is up.
  - Down: the monitored Service is down.
  - UP-inconclusive-no-data: indicates that there are enough probe nodes online, but not enough raw data points were received to make a determination.
  - UP-inconclusive-no-probes: indicates that there are not enough probe nodes online to make a determination.
  - UP-inconclusive-reconfig: indicates that the system is undergoing a reconfiguration for the particular TLD and service.

- "testedInterface", a JSON array that contains information about the interface being tested. The "testedInterface" fields contains the following fields:
  - "interface", a JSON string that contains the tested interface.
  - "probes", a JSON array that contains detailed monitoring information per probe node. The "probes" field contains the following fields:
    - "city", a JSON string with the location of the probe node.
    - “testedName”, a JSON string with the value of the domain name used in a DNS or RDDS query. This string is only returned with the /v2 API endpoint.
    - “transport”, a JSON string denoting the IP transport protocol used. Values are either “udp” or “tcp”. This string is only returned with the /v2 API endpoint.
    - "status", a JSON string that contains the status of the interface as seen from the probe node. The "status" field can contain one of the following values:
      - Up: the monitored Service is up.
      - Down: the monitored Service is down.
      - Offline: the probe node is offline. Note: the probe node is not considered part of the probe node universe when calculating the rolling week thresholds.
      - No result: results from this probe node were not received by the central server when the calculations were executed. Note: the service is considered to be up for rolling week threshold calculations.
    - "testData", a JSON array that contains monitoring information. The "testData" field contains the following fields:
      - "target", a JSON string that in the case of the DNS Service contains the name server being tested, in the case of RDDS, this field contains "null".
      - "status", a JSON string that in the case of the DNS Service contains the status of the name server being tested. In the case of RDDS this field contains the status of the IP address being tested (available in the "metrics" element, see below). The "status" field contains the following fields:
        - Up: the test was considered successful.
        - Down: the test was not considered successful.

+ "metrics", a JSON array with monitoring details of particular tests. The "metrics" field contains the following fields:

  - "testDateTime", a JSON number that contains the date and time the result was computed. If the "result" field contains "no data", the "testDateTime" field will contain a null value.

  - "targetIP", a JSON string with the IP Address being tested.

  - "rtt", a JSON number that contains the milliseconds needed for the query to be resolved. If the "result" field contains an error code or "no data", the "rtt" field will contain a null value.

  - "result", a JSON string that contains the value "ok" if the query response was valid, "no data" if no data was received from the probe node, or an error code if the result is not valid. The information regarding the error codes may be found in section 5.8.1 and 5.8.2. Note: in case of "no data" the query response is assumed to be valid for rolling week threshold calculations

    Note: previous versions of MoSAPI used a JSON number when the result was an error code instead of a JSON string.

  - “nsid”, a JSON string the NSID of the responding nameserver (see RFC 5001). This string is only present in measurements of DNS and DNSSEC and is only returned from the /v2 API endpoint.

Note: the JSON object for the measurement details provides the status of the test cycle computed from the results of all probe nodes.

Differences between v1 and v2:

Between /v1 and /v2 versions of this API endpoint, no items were removed from the results. In the items listed above, those added to /v2 are noted. It is recommended to use /v2 as /v1 API endpoints will be deprecated in the future.

Example using CURL to request the details of a measurement:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/incidents/1666623720.732355/1666623960.732355.json
```

5.8.1. DNS/DNSSEC Monitoring error codes

The following table lists the error codes for DNS/DNSSEC monitoring:

Result Code	Obsolete	Internal Error	Interface	Description
-1	N	Y	DNS UDP / DNS TCP	Internal error.
-2	N	Y	DNS UDP	Expecting NOERROR RCODE but got unexpected RCODE from local resolver.
-3	N	Y	DNS TCP	Expecting NOERROR RCODE but got unexpected RCODE from local resolver.
-200	N	N	DNS UDP	No reply from the authoritative name server.
-201	Y	N	DNS UDP / DNS TCP	Invalid reply from Name Server.
-204	Y	N	DNS UDP / DNS TCP	DNSSEC error.
-206	Y	N	DNS UDP / DNS TCP	Keyset is not valid.
-207	N	N	DNS UDP	Expecting DNS class IN but got class CHAOS in the DNS response.
-208	N	N	DNS UDP	Expecting DNS class IN but got class HESIOD in the DNS response.
-209	N	N	DNS UDP	Expecting DNS class IN but got something different from class IN, CHAOS or HESIOD in the DNS response.
-210	N	N	DNS UDP	Header section incomplete in the DNS response.
-211	N	N	DNS UDP	Question section incomplete in the DNS response.
-212	N	N	DNS UDP	Answer section incomplete in the DNS response.
-213	N	N	DNS UDP	Authority section incomplete in the DNS response.
-214	N	N	DNS UDP	Additional section incomplete in the DNS response.
-215	N	N	DNS UDP	Malformed DNS response.
-250	N	N	DNS UDP	Querying for a non-existent domain - the AA flag is off (was expecting on) in the DNS response.
-251	N	N	DNS UDP	Querying for a non-existent domain - Domain name being queried not present in question section of the DNS response.
-253	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got FORMERR on the DNS response.
-254	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got SERVFAIL on the DNS response.

-255	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NOTIMP on the DNS response.
-256	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got REFUSED on the DNS response.
-257	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got YXDOMAIN on the DNS response.
-258	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got YXRRSET on the DNS response.
-259	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NXRRSET on the DNS response.
-260	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NOTAUTH on the DNS response.
-261	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NOTZONE on the DNS response.
-270	N	N	DNS UDP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got unexpected (i.e., 11-15) on the DNS response.
-400	N	N	DNS UDP	Timeout when waiting for a response from the TLD authoritative servers as reported by the local DNS resolver.
-401	N	N	DNS UDP	The TLD is configured as DNSSEC-enabled, but no DNSKEY was found in the apex.
-402	N	N	DNS UDP	DNSSEC error in the chain of trust from the root to the TLD apex.
-403	N	N	DNS UDP	The TLD was not found in the root.
-405	N	N	DNS UDP	Unknown cryptographic algorithm found in a DNSSEC signature.
-406	N	N	DNS UDP	Unsupported cryptographic algorithm found in a DNSSEC signature.
-407	N	N	DNS UDP	No RRSIGs were found, and the TLD is expected to be signed.
-408	N	N	DNS UDP	Querying for a non-existent domain - No NSEC/NSEC3 RRs were found in the authority section.
-410	N	N	DNS UDP	No signature covering the RRSET was found.
-414	N	N	DNS UDP	An RRSIG was found, and it is not signed by a DNSKEY from the KEYSET.
-415	N	N	DNS UDP	Bogus DNSSEC signature was found.
-416	N	N	DNS UDP	An expired DNSSEC signature was found.
-417	N	N	DNS UDP	A DNSSEC signature with an inception date in the future was found.
-418	N	N	DNS UDP	A DNSSEC signature with expiration date earlier than inception date was found.
-422	N	N	DNS UDP	A resource record (RR) not covered by the given NSEC/NSEC3 RRs was found. Note: the condition is only evaluated if RCODE=NXDOMAIN.

-425	N	N	DNS UDP	Malformed RRSIG with too few RDATA fields was found.
-427	N	N	DNS UDP	Malformed DNSSEC response.
-600	N	N	DNS TCP	Connection to the name server was successful, but the connection timed out.
-601	N	N	DNS TCP	Error when opening a connection to the name server.
-607	N	N	DNS TCP	Expecting DNS class IN but got CHAOS in the DNS response.
-608	N	N	DNS TCP	Expecting DNS class IN but got HESIOD in the DNS response.
-609	N	N	DNS TCP	Expecting DNS class IN but got something different from [IN, CHAOS or HESIOD] in the DNS response.
-610	N	N	DNS TCP	Header section incomplete in the DNS response.
-611	N	N	DNS TCP	Question section incomplete in the DNS response.
-612	N	N	DNS TCP	Answer section incomplete in the DNS response.
-613	N	N	DNS TCP	Authority section incomplete in the DNS response.
-614	N	N	DNS TCP	Additional section incomplete in the DNS response.
-615	N	N	DNS TCP	Malformed DNS response.
-650	N	N	DNS TCP	Querying for a non-existent domain - the AA flag is off (expecting on) in the DNS response.
-651	N	N	DNS TCP	Querying for a non-existent domain - Domain name being queried not present in question section of the DNS response.
-653	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got FORMERR on the DNS response.
-654	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got SERVFAIL on the DNS response.
-655	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NOTIMP on the DNS response.
-656	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got REFUSED on the DNS response.
-657	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got YXDOMAIN on the DNS response.
-658	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got YXRRSET on the DNS response.
-659	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NXRRSET on the DNS response.

-660	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NOTAUTH on the DNS response.
-661	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got NOTZONE on the DNS response.
-670	N	N	DNS TCP	Querying for a non-existent domain - Expecting NXDOMAIN/NOERROR RCODE but got unexpected (i.e., 11-15) on the DNS response.
-800	N	N	DNS TCP	Timeout when waiting for a response from the TLD authoritative servers as reported by the local DNS resolver.
-801	N	N	DNS TCP	The TLD is configured as DNSSEC-enabled, but no DNSKEY was found in the apex.
-802	N	N	DNS TCP	DNSSEC error in the chain of trust from the root zone to the TLD apex.
-803	N	N	DNS TCP	The TLD was not found in the root.
-805	N	N	DNS TCP	Unknown cryptographic algorithm found in a DNSSEC signature.
-806	N	N	DNS TCP	Unsupported cryptographic algorithm found in a DNSSEC signature.
-807	N	N	DNS TCP	No RRSIGs where found, and the TLD is expected to be signed.
-808	N	N	DNS TCP	Querying for a non-existent domain - No NSEC/NSEC3 RRs were found in the authority section
-810	N	N	DNS TCP	No signature covering the RRSET was found.
-814	N	N	DNS TCP	An RRSIG was found, and it is not signed by a DNSKEY from the KEYSET.
-815	N	N	DNS TCP	Bogus DNSSEC signature was found.
-816	N	N	DNS TCP	An expired DNSSEC signature was found.
-817	N	N	DNS TCP	A DNSSEC signature with an inception date in the future was found.
-818	N	N	DNS TCP	A DNSSEC signature with expiration date earlier than inception date was found.
-822	N	N	DNS TCP	A RR not covered by the given NSEC/NSEC3 RRs was found. Note: the condition is only evaluated if RCODE=NXDOMAIN.
-825	N	N	DNS TCP	Malformed RRSIG with too few RDATA fields was found.
-827	N	N	DNS TCP	Malformed DNSSEC response.

Note: error codes marked as Obsolete are listed for documentation purposes.

Note: a test with an error code marked as Internal Error will be considered to be UP.

5.8.2. RDDS Monitoring error codes

The following table lists the error codes for RDDS monitoring:

Result Code	Obsolete	Internal Error	Interface	Description
-1	N	Y	Whois-43 / Web-whois	Internal Error
-2	N	Y	Whois-43 / Web-whois	RDDS service could not be tested due to lack of IPv4/6 transport in the probe node.
-3	N	Y	Whois-43	Expecting NOERROR RCODE but got unexpected code when resolving the WHOIS-43 hostname using the local DNS resolver.
-4	N	Y	Web-whois	Expecting NOERROR RCODE but got unexpected code when resolving web-whois hostname using the local DNS resolver.
-200	Y	N	Whois-43	Connection timed out while trying to get a response from the server.
-201	N	N	Whois-43	Syntax error while parsing the WHOIS-43 response.
-204	Y	N	Web-whois	Connection timed out while trying to get a response from the server.
-205	Y	N	Whois-43 / Web-whois	Error when trying to resolve the Whois server hostname (e.g., whois.nic.example).
-206	N	N	Web-whois	An HTTP status code was not found in the HTTP message.
-207	Y	N	Web-whois	No HTTP/200 status code in response (after following redirects).
-222	N	N	Whois-43	Timeout when waiting for a response from the TLD authoritative servers as reported by the local DNS resolver.
-224	N	N	Whois-43	DNSSEC error when trying to resolve the hostname for the WHOIS-43 server.
-225	N	N	Whois-43	The hostname for the WHOIS-43 server was not found in the DNS.
-227	N	N	Whois-43	Connection to WHOIS-43 server was successful, but the connection timed out.
-228	N	N	Whois-43	Connection to WHOIS-43 server was unsuccessful.
-229	N	N	Whois-43	Empty response received from WHOIS-43 server.
-250	N	N	Web-whois	Timeout when waiting for a response from the TLD authoritative servers as reported by the local DNS resolver.
-252	N	N	Web-whois	DNSSEC error when trying to resolve the hostname for the web-whois server.
-253	N	N	Web-whois	The hostname for the web-whois server was not found in the DNS.
-255	N	N	Web-whois	Connection to the web-whois server was successful, but the connection timed out.

-256	N	N	Web-whois	Error when opening a connection to web-whois server.
-257	N	N	Web-whois	Malformed HTTP message.
-258	N	N	Web-whois	Malformed HTTP message or TLS general error.
-259	N	N	Web-whois	The maximum number of HTTP redirects (301, 302 and 303) were followed, and a 200 / HTTP status code was not found.
-300	N	N	Web-whois	Expecting HTTP status code 200 but got 100.
-301	N	N	Web-whois	Expecting HTTP status code 200 but got 101.
-302	N	N	Web-whois	Expecting HTTP status code 200 but got 102.
-303	N	N	Web-whois	Expecting HTTP status code 200 but got 103.
-304	N	N	Web-whois	Expecting HTTP status code 200 but got 201.
-305	N	N	Web-whois	Expecting HTTP status code 200 but got 202.
-306	N	N	Web-whois	Expecting HTTP status code 200 but got 203.
-307	N	N	Web-whois	Expecting HTTP status code 200 but got 204.
-308	N	N	Web-whois	Expecting HTTP status code 200 but got 205.
-309	N	N	Web-whois	Expecting HTTP status code 200 but got 206.
-310	N	N	Web-whois	Expecting HTTP status code 200 but got 207.
-311	N	N	Web-whois	Expecting HTTP status code 200 but got 208.
-312	N	N	Web-whois	Expecting HTTP status code 200 but got 226.
-313	N	N	Web-whois	Expecting HTTP status code 200 but got 300.
-317	N	N	Web-whois	Expecting HTTP status code 200 but got 304.
-318	N	N	Web-whois	Expecting HTTP status code 200 but got 305.
-319	N	N	Web-whois	Expecting HTTP status code 200 but got 306.
-320	N	N	Web-whois	Expecting HTTP status code 200 but got 307.
-321	N	N	Web-whois	Expecting HTTP status code 200 but got 308.
-322	N	N	Web-whois	Expecting HTTP status code 200 but got 400.
-323	N	N	Web-whois	Expecting HTTP status code 200 but got 401.
-324	N	N	Web-whois	Expecting HTTP status code 200 but got 402.
-325	N	N	Web-whois	Expecting HTTP status code 200 but got 403.
-326	N	N	Web-whois	Expecting HTTP status code 200 but got 404.
-327	N	N	Web-whois	Expecting HTTP status code 200 but got 405.
-328	N	N	Web-whois	Expecting HTTP status code 200 but got 406.
-329	N	N	Web-whois	Expecting HTTP status code 200 but got 407.
-330	N	N	Web-whois	Expecting HTTP status code 200 but got 408.
-331	N	N	Web-whois	Expecting HTTP status code 200 but got 409.
-332	N	N	Web-whois	Expecting HTTP status code 200 but got 410.
-333	N	N	Web-whois	Expecting HTTP status code 200 but got 411.
-334	N	N	Web-whois	Expecting HTTP status code 200 but got 412.
-335	N	N	Web-whois	Expecting HTTP status code 200 but got 413.
-336	N	N	Web-whois	Expecting HTTP status code 200 but got 414.
-337	N	N	Web-whois	Expecting HTTP status code 200 but got 415.
-338	N	N	Web-whois	Expecting HTTP status code 200 but got 416.
-339	N	N	Web-whois	Expecting HTTP status code 200 but got 417.
-340	N	N	Web-whois	Expecting HTTP status code 200 but got 421.

-341	N	N	Web-whois	Expecting HTTP status code 200 but got 422.
-342	N	N	Web-whois	Expecting HTTP status code 200 but got 423.
-343	N	N	Web-whois	Expecting HTTP status code 200 but got 424.
-344	N	N	Web-whois	Expecting HTTP status code 200 but got 426.
-345	N	N	Web-whois	Expecting HTTP status code 200 but got 428.
-346	N	N	Web-whois	Expecting HTTP status code 200 but got 429.
-347	N	N	Web-whois	Expecting HTTP status code 200 but got 431.
-348	N	N	Web-whois	Expecting HTTP status code 200 but got 451.
-349	N	N	Web-whois	Expecting HTTP status code 200 but got 500.
-350	N	N	Web-whois	Expecting HTTP status code 200 but got 501.
-351	N	N	Web-whois	Expecting HTTP status code 200 but got 502.
-352	N	N	Web-whois	Expecting HTTP status code 200 but got 503.
-353	N	N	Web-whois	Expecting HTTP status code 200 but got 504.
-354	N	N	Web-whois	Expecting HTTP status code 200 but got 505.
-355	N	N	Web-whois	Expecting HTTP status code 200 but got 506.
-356	N	N	Web-whois	Expecting HTTP status code 200 but got 507.
-357	N	N	Web-whois	Expecting HTTP status code 200 but got 508.
-358	N	N	Web-whois	Expecting HTTP status code 200 but got 510.
-359	N	N	Web-whois	Expecting HTTP status code 200 but got 511.
-360	N	N	Web-whois	Expecting HTTP status code 200 but got an unexpected status code.

Note: the DNS resolvers used in the system validate DNSSEC.

Note: error codes marked as Obsolete are listed for documentation purposes.

Note: a test with an error code marked as an Internal Error will be considered to be UP.

5.9 Querying the Soon to Be Revoked Flag

```
<base_url>/monitoring/soonToBeRevoked
```

Possible results:

- HTTP/200, when a valid request is received, the <base_url>/monitoring/soonToBeRevoked API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body:
  - "version", a JSON number that contains the version number of the JSON object. The number will be 1 for objects returned from /v1 APIs and 2 for objects returned from /v2 APIs.
  - "lastUpdateApiDatabase", a JSON number that contains the Unix time stamp of the date and time that the monitoring information provided in the MoSAPI was last updated from the monitoring system central database.
  - "enabled", a JSON string with either the value “Yes”, meaning the Soon to Be Revoked flag is set, or “No”, meaning the Soon to be Revoked flag is not set.

Differences between v1 and v2:

This API endpoint is not present in /v1 and is only available in /v2.

Example using CURL to request the availability of a Service:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/soonToBeRevoked
```

Example of a JSON response for a soon to be revoked request:

```
{
  "version": 2,
  "lastUpdateApiDatabase": 1422492450,
  "enabled": "No"
}
```

6. Maintenance window support

In previous versions of MoSAPI, registry operators could use a schedule object to manage maintenance windows. This feature has been moved to RRI and is now fully deprecated in MoSAPI.

Information regarding RRI may be found here: https://www.icann.org/rri

The public deprecation notice for this feature may be found here: https://www.icann.org/en/system/files/files/mosapi-recent-changes-05aug22-en.pdf

7. Probe node network

In previous versions of MoSAPI, registry operators could obtain a list of MoSAPI probe nodes. This feature has been moved to RRI and is now fully deprecated in MoSAPI.

Information regarding RRI may be found here: https://www.icann.org/rri

The public deprecation notice for this feature may be found here:
https://www.icann.org/en/system/files/files/mosapi-recent-changes-05aug22-en.pdf

8. HTTP/400 extended error codes

The API endpoints provides a HTTP/400 if the input does not comply with the business rules, or the syntax of the input is invalid. The API endpoint sets the HTTP header Content-type to "application/json; charset=utf-8". A JSON object with the fields listed below is provided in the HTTP Entity-body:

- "resultCode", a JSON number that contains the result code.
- "message", a JSON string that contains the standard error message defined in the table below.
- "description", a JSON string that may be used to provide additional error diagnostic information.

Example of a JSON object that contains extended error codes:

```
{
  "resultCode": "2015",
  "description": "The value of falsePositive (test) is invalid",
  "message": "The value of falsePositive is invalid"
}
```

The following table contains the extended error codes for the HTTP/400 status:

Result Code	API endpoints	HTTP Verb			Message
		PUT	DELETE	GET	
2011	<base_url>/monitoring/<service>/incidents			•	The difference between endDate and startDate is more than 31 days.
2012	<base_url>/monitoring/<service>/incidents			•	The endDate is before the startDate.
2013	<base_url>/monitoring/<service>/incidents			•	The startDate syntax is incorrect.
2014	<base_url>/monitoring/<service>/incidents			•	The endDate syntax is incorrect.
2015	<base_url>/monitoring/<service>/incidents			•	The value of falsePositive is invalid.

9. Domain Abuse Activity Reporting (DAAR)

ICANN's Domain Abuse Activity Reporting (DAAR) is a system for studying and reporting on domain name registration and security threat (domain abuse) behavior across top-level domain (TLD) registries and registrars. More information about the DAAR can be found at https://www.icann.org/octo-ssr/daar.

MoSAPI provides registries with access to DAAR-measurements for their TLDs using the GET and HEAD HTTP verbs in the MoSAPI endpoints described below. At the moment, DAAR data in MoSAPI is only available to gTLD registries.

9.1. Getting the latest DAAR report available for the TLD

```
<base_url>/daar/report/latest
```

Possible results:

- HTTP/404, the <base_url>/daar/report/latest API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body (only when using the HTTP/GET verb) with the string "Not available" if no report exists for the TLD.

- HTTP/200, when a valid request is received, the <base_url>/daar/report/latest API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  The header "Last-Modified" is set to the date and time when the report was generated.

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body (only when using the HTTP/GET verb):

  - "version", a JSON number that contains the version number of the JSON object intended for future upgrades of the specification; for this version the value will always be "1".

  - "tld", a JSON string that contains the monitored TLD.

  - "daarReportDate", a JSON string that contains the date of the report in the format <YYYY>-<MM>-<DD>. Where:

    - <YYYY>: year
    - <MM>: zero-padded month
    - <DD>: zero-padded day.

  - "daarReportData", measurements of the DAAR reporting including the following elements (JSON strings): domains in zone, number of unique abuse domains, number of spam domains, number of phish domains, number of bot c&c domains and number of malware domains.

Example using CURL to request latest DAAR report:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v1/daar/report/latest
```

Example of a JSON response for the latest DAAR report:

```
{
	"version": 1,
	"tld": "example",
	"daarReportDate": "2018-12-12",
	"daarReportData": {
		"domainsInZone": 27957,
		"uniqueAbuseDomains": 14,
```

```
            "spamDomains": 10,
            "phishDomains": 3,
            "botnetCcDomains": 0,
            "malwareDomains": 2
        }
}
```

9.2. Querying for a DAAR report for a date

```
<base_url>/daar/report/<YYYY>-<MM>-<DD>
```

Possible results:

- HTTP/404, the <base_url>/daar/report/<YYYY>-<MM>-<DD> API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body (only when using the HTTP/GET verb) with the string "Not available" if a report for the specified date does not exist.

- HTTP/200, when a valid request is received, the <base_url>/daar/report/<YYYY>-<MM>-<DD> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the DAAR report (see section 9.1) is provided in the HTTP Entity-body (only when using the HTTP/GET verb).

  The header "Last-Modified" is set to the date and time when the report was generated.

Example using CURL to request the details of a measurement:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v1/daar/report/2019-01-01
```

Example of JSON response for the DAAR report for a date:

See example in section 9.1.

9.3. Querying for DAAR reports available

```
<base_url>/daar/reports?startDate=<startDate>&endDate=<endDate>
```

Where:

- Optional <startDate> to be substituted by <YYYY>-<MM>-<DD> to match reports after the provided date and time. If <startDate> is omitted, the oldest available report will match.
- Optional <endDate> to be substituted by <YYYY>-<MM>-<DD> to match reports before the provided date and time. If <endDate> is omitted, it will be substituted with the current date.

  Note: if both <startDate> and <endDate> are omitted, all available reports will match.

Possible results:

- HTTP/400, see section 8, only the following error codes apply: 2012, 2013 and 2014.
- HTTP/200, when a valid request is received, the <base_url>/daar/reports?startDate=<startDate>&endDate=<endDate> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".

  If a valid request is received, a JSON object with the fields listed below is provided in the HTTP Entity-body (only when using the HTTP/GET verb):

  - "version", a JSON number that contains the version number of the JSON object intended for future upgrades of the specification; for this version the value will always be "1".
  - "tld", a JSON string that contains the monitored TLD.
  - "daarReports", a JSON array with all the reports available within the period.

    The array contains JSON objects with the following elements:

    - "daarReportDate", see section 9.1 for definition.
    - "daarReportGenerationDate", date and time that the report was generated in the format specified in RFC 3339.

Example using CURL to request the details of a measurement:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v1/daar/reports
```

Example of a JSON response for a query of reports available:


```
{
	"version": 1,
	"tld": "example",
	"daarReports": [{
			"daarReportDate": "2018-12-12",
			"daarReportGenerationDate": "2018-12-13T23:20:50.52Z"
		},
		{
			"daarReportDate": "2018-12-13",
			"daarReportGenerationDate": "2018-12-13T23:20:51.52Z"
		}

	]
}
```


Example of a JSON response when no reports are available for the queried period:


```
{
	"version": 1,
	"tld": "example",
	"daarReports": []
}
```

10. Recent Measurements

MoSAPI provides registries with access to measurements files for cycles marked as up or down using the GET or HEAD HTTP verbs in the MoSAPI endpoints described below. This functionality allows registries to obtain raw data regarding the tests performed by the monitoring system regardless of the cycle being part of an incident.

10.1. Querying years for which reports are available

```
<base_url>/monitoring/<service>/measurements
```

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/measurements API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/measurements API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the years for which reports are available is provided in the HTTP Entity-body (only when using the HTTP/GET verb).

Example using CURL to request years for which reports are available:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/dns/measurements
```

Example of JSON response of the years for which reports are available:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "years": ["2018", "2017", "2016"]
}
```

Example of a JSON response when no years are available for the monitored service:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "years": []
}
```

10.2. Querying months for which reports are available

```
<base_url>/monitoring/<service>/measurements/<YYYY>
```

Where:

- `<YYYY>: year`

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/measurements API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored or the <YYYY> does not exist in the source.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/measurements/<YYYY> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the months for which reports are available is provided in the HTTP Entity-body (only when using the HTTP/GET verb).

Example using CURL to request years for which reports are available:

```
curl --cookie cookies.txt https://mosapi.icann.org/ry/example/v2/monitoring/dns/measurements/<YYYY>
```

Example of JSON response of the months for which reports are available:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "months": ["06", "05", "04", "03", "02", "01"]
}
```

Example of a JSON response when no months are available for the monitored service and the <YYYY>:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "months ": []
}
```

10.3. Querying days for which reports are available

```
<base_url>/monitoring/<service>/measurements/<YYYY>/<MM>
```

Where:

- `<YYYY>: year`
- `<MM>: zero-padded month`

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/measurements API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored or the <YYYY>/<MM> does not exist in the source.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/measurements/<YYYY>/<MM> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the days for which reports are available is provided in the HTTP Entity-body (only when using the HTTP/GET verb).

Example using CURL to request years for which reports are available:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/measurements/<YYYY>/<MM>
```

Example of JSON response of the days for which reports are available:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "days": ["03", "02", "01"]
}
```

Example of a JSON response when no days are available for the monitored service and the <YYYY>/<MM>:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "days ": []
}
```

10.4. Querying for available measurements

```
<base_url>/monitoring/<service>/measurements/<YYYY>/<MM>/<DD>
```

Where:

- <YYYY>: year
- <MM>: zero-padded month
- <DD>: zero-padded day

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/measurements API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored or the <YYYY>/<MM>/<DD> does not exist in the source.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/measurements/<YYYY>/<MM>/<DD> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8". If a valid request is received, a JSON object with the available measurements is provided in the HTTP Entity-body (only when using the HTTP/GET verb).

Example using CURL to request years for which reports are available:

```
curl --cookie cookies.txt
https://mosapi.icann.org/ry/example/v2/monitoring/dns/measurements/<YYYY>/<MM>/<DD>
```

Example of JSON response of the days for which measurements are available:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "measurements": ["1422492930.json", "1422492990.json", "1422493050.json",
"1422493110.json"]
}
```

Example of a JSON response when no measurements are available for the monitored service and the <YYYY>/<MM>/<DD>:

```
{
      "version": 2,
      "lastUpdateApiDatabase": 1422492450,
      "measurements ": []
}
```

10.5. Querying the details of a particular measurement

```
<base_url>/monitoring/<service>/measurements/<YYYY>/<MM>/<DD>/<measurementID>
```

Where:

- `<YYYY>: year`
- `<MM>: zero-padded month`
- `<DD>: zero-padded day`
- `<measurementID> must be substituted by the measurement id assigned by the monitoring system.`

Possible results:

- HTTP/404, the <base_url>/monitoring/<service>/measurements/<YYYY>/<MM>/<DD>/<measurementID> API endpoint provides a HTTP/404 status code, sets the HTTP header Content-type to "text/plain; charset=utf-8", and provides a text response in the HTTP Entity-body with the string "Not available" if the specified <service> is not being monitored or the <measurementID> does not exist in the source.

- HTTP/200, when a valid request is received, the <base_url>/monitoring/<service>/measurements/<YYYY>/<MM>/<DD>/<measurementID> API endpoint provides a HTTP/200 status code and sets the HTTP header Content-type to "application/json; charset=utf-8".
  If a valid request is received, a JSON object with the fields listed in section 5.8 is provided in the HTTP Entity-body.

  The Content-Encoding entity header is set to "gzip" indicating that the entity-body is compressed using the Lempel-Ziv coding (LZ77), with a 32-bit CRC.

- HTTP/406, when a valid request is received, the <base_url>/monitoring/<service>/measurements/<YYYY>/<MM>/<DD>/<measurementID> API endpoint provides a HTTP/406 if the client does not include a HTTP header Accept-Encoding with value set to "gzip".

11.List of ICANN-accredited Registrars' RDAP Base URLs

In previous versions of MoSAPI, registry operators could query for MoSAPI for a list of RDAP Base URLs of the ICANN-accredited Registrars. This feature is now fully deprecated in MoSAPI, and the list is available from IANA.

The IANA list may be found here: https://www.iana.org/assignments/registrar-ids/registrar-ids.xhtml

The public deprecation notice for this feature may be found here: https://www.icann.org/en/system/files/files/mosapi-recent-changes-05aug22-en.pdf

12.Alternative methods of authentication

In addition to HTTP Basic Access Authentication, and state management using HTTP Cookies described in section 3 and 4, TLS Client Authentication is supported by MoSAPI as an alternative method of authentication and state management.

12.1. Overview

Nowadays it is common for a Registry Operator to subcontract the operation of Registry Services to one or more Registry Service Providers (RSP).

Before the implementation of TLS Client Authentication, the only mechanism of authentication was Basic Authentication with one set of credentials per TLD; therefore, an RSP wanting access to MoSAPI would have been forced to share the credentials with the Registry Operator. Thus, if credentials need to be changed, coordination between the RSP and Registry Operator is required.

RSPs have requested the ability to have separated credentials and additional authentication methods to access MoSAPI. Additionally, RSPs have requested a solution that allows them to change the credentials without going through the Registry Operator to execute such change.

RSPs have also expressed interest in using TLS Client Authentication, based on the experience of some RSPs using TLS Client Authentication in EPP.

12.2. TLS Client Authentication

As the name implies, this alternative method uses TLS with client authentication, meaning that MoSAPI will authenticate the client using X.509 certificates in HTTPS. TLSA DNS resource records (see RFC6698) are used to provide a mechanism to store the client certificates to be authenticated and authorized for a given TLD.

In order to setup TLS Client Authentication, the Registry Operator (logged with an account allowed to make changes for a TLD) needs to provide three pieces of information in the NSP portal:

1. Domain name for TLS client access (e.g., rsp1.nic.example)
2. Authorized roles related to SLAM (more than one may be chosen):
    a. MoSAPI - TLD SLAM Data (see, section 5 and 10)
    b. MoSAPI - TLD DAAR Data (see, section 9)

    Note: other non-MoSAPI related roles may be available in NSP.

MoSAPI uses the domain name for TLS access to find the TLSA RRs to be used to validate the client during the TLS handshake. A batch process, at least every hour, will query the DNS (validating with DNSSEC) to get the universe of TLSA RRs per owner name.

The following combinations of TLSA Certificate Usages Registry, TLSA Selectors and TLSA Matching Types are supported:

TLSA Certificate Usages Registry	TLSA Selectors	TLSA Matching Types
3	1	1
		2

The following public key algorithms are supported on the X.509 certificates used for TLS client authentication:

- RSA encryption with a key size of 4096 or higher.
- Elliptic Curve public key

The following signature algorithms are supported on the X.509 certificates used for TLS client authentication:

- `sha256WithRSAEncryption`
- `sha384WithRSAEncryption`
- `sha512WithRSAEncryption`
- `ecdsa-with-SHA256`
- `ecdsa-with-SHA384`
- `ecdsa-with-SHA512`

12.3. Authentication and Authorization with TLS Client Authentication

MoSAPI will perform the following steps when authenticating and authorizing clients using TLS Client Authentication:

1. A client connects to MoSAPI and provides its certificate during the TLS handshake.
2. MoSAPI verifies that the client certificate validates with any TLSA RR of the universe of TLSA RRs. If there is a match, the TLS connection is considered TLS-client authenticated. Otherwise, the TLS session is considered to be non-TLS-client authenticated; the client will still have the option to authenticate using HTTP Basic Access Authentication as before.
3. The client requests access to the SLAM monitoring data of a given TLD.
4. **If the TLS connection is considered TLS-client authenticated**, using the set of authorized TLSA RRs for the TLD, MoSAPI validates there is a match for the client certificate. In other words, MoSAPI validates that the client certificate is authorized for the TLD based on the roles permitted in NSP. HTTP/401 is returned to the client if this step fails.
   If the TLS connection is considered non-TLS-client authenticated, HTTP cookies will be used to validate the session as described in section 3 and 4 of this specification.

   If client requests access to the monitoring data of another TLD, step 4 is repeated.

Every time an HTTPS connection is established, an internal new session is created. Only 4 concurrent sessions are permitted per client certificate. A session is only terminated after its expiration date (by default, the value of expiration date is 4 hours after the session is created), or if the session is the oldest and a new session is being created, that would be above the limit of permitted concurrent sessions. When a session is terminated, the server closes the HTTPS connection.

The error codes HTTP/401 described in section 3.1 apply to this method of authentication.