Workgroup:	Network Working Group
Internet-Draft:	draft-wullink-restful-epp-01
Published:	15 February 2024
Intended Status:	Standards Track
Expires:	18 August 2024

Authors:	M. Wullink	M. Davids
	SIDN Labs	*SIDN Labs*

Extensible Provisioning Protocol (EPP) RESTful Transport

Abstract

This document describes RESTful EPP (REPP), a data format agnostic, REST based Application Programming Interface (API) for the Extensible Provisioning Protocol [RFC5730]. REPP enables the development of a stateless and scalable EPP service.

This document includes a mapping of [RFC5730] [XML] EPP commands to a RESTful HTTP based interface. Existing semantics as defined in [RFC5731], [RFC5732] and [RFC5733] are retained and reused in RESTful EPP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 18 August 2024.

Copyright Notice

Table of Contents

1. Introduction

This document describes an Application Programming Interface (API) for the Extensible Provisioning Protocol (EPP) protocol described in [RFC5730]. The API leverages the HTTP protocol [RFC2616] and the principles of [REST]. Conforming to the REST constraints is generally referred to as being "RESTful". Hence the API is dubbed: "'RESTful EPP" or "REPP" for short.

REPP includes a mapping of [RFC5730] EPP commands to REST resources based on Uniform Resource Locators (URLs) defined in [RFC1738]. REPP uses a stateless architecture. It aims to provide a solution that is more suitable for complex, high availability environments.

Section 2.1 of [RFC5730] describes how EPP can be layered over multiple transport protocols. Currently, EPP transport over TCP [RFC5734] is the only widely deployed transport mapping for EPP. Section 2.1 furthermore requires that newly defined transport mappings preserve the stateful nature of EPP. This document updates this requirement to also allow stateless for EPP transport.

The stateless nature of REPP requires that no client or application state is maintained on the server. Each client request to the server must contain all the information necessary for the server to process the request.

REPP is data format agnostic, the client uses server-driven content negotiation. Allowing the client to select from a set of representation media types supported by the server, such as XML, JSON [RFC8259] or [YAML].

2. Terminology

In this document the following terminology is used.

REST - Representational State Transfer ([REST]). An architectural style.

RESTful - A RESTful web service is a web service or API implemented using HTTP and the principles of [REST].

EPP RFCs - This is a reference to the EPP version 1.0 specifications [RFC5730], [RFC5731], [RFC5732] and [RFC5733].

Stateful EPP - The definition according to Section 2 of [RFC5730].

RESTful EPP or REPP - The RESTful transport for EPP described in this document.

URL - A Uniform Resource Locator as defined in [RFC3986].

Resource - An object having a type, data, and possible relationship to other resources, identified by a URL.

Command Mapping - A mapping of [RFC5730] EPP commands to RESTful EPP URL resources.

REPP client - An HTTP user agent performing an REPP request

REPP server - An HTTP server responsible for processing requests and returning results in any supported media type.

3. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT","SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

XML is case sensitive. Unless stated otherwise, XML specifications and examples provided in this document MUST be interpreted in the character case presented to develop a conforming implementation.

The examples in this document assume that request and response messages are properly formatted XML documents.

In examples, lines starting with "C:" represent data sent by a REPP client and lines starting with "S:" represent data returned by a REPP server. Indentation and white space in examples are provided only to illustrate element relationships and are not REQUIRED features of the protocol.

All example requests assume a REPP server using HTTP version 2 is listening on the standard HTTPS port on host reppp.example.nl. An authorization token has been provided by an out of band process and MUST be used by the client to authenticate each request.

4. Design Considerations

RESTful transport for EPP (REPP) is designed to improve the ease of design, development, deployment, and management of an EPP service. This section lists the main design criteria.

- Ease of use, provide a clear, clean, easy to use and self-explanatory interface that can easily be integrated into existing software systems. Based on these principles a [REST] architectural style was chosen, where a client interacts with a REPP server via HTTP.
- Scalability, HTTP allows the use of well know mechanisms for creating scalable systems, such as load balancing. Load balancing at the level of request messages is more efficient compared to load balancing based on TCP sessions. When using EPP over TCP, the TCP session can be used to transmit multiple request messages and these are then all processed by a single EPP server and not load balanced across a pool of available servers. During normal registry operations, the bulk of EPP requests can be expected to be of the informational type, load balancing and possibly separating these to dedicated compute resources may also improve registry services and provide better performance for the transform request types.
- Stateless, [RFC5730] REQUIRES a stateful session between a client and server. A REPP server MUST be stateless and MUST NOT keep client session or any other application state. Each client request needs to provide all the information necessary for the server to successfully process the request.

- Security, allow for the use of authentication and authorization solutions available for HTTP based applications. HTTP provides an Authorization header Section 14.8 of [RFC2616].
- Content negotiation, A server may choose to include support for multiple media types. The client must be able to signal to the server what media type the server should expect for the request content and to use for the response content. This document only describes the use of [XML] but the use of other media types such as JSON [RFC7159] should also be possible.
- Compatibility with existing EPP semantics defined in the EPP RFCs.
- Simplicity, when the semantics of a resource URL and HTTP method match an EPP command and request message, the use of a request message should be optional.
- Performance, reducing the number of required request and response messages, improves the performance and network bandwidth requirements for both client and server. Fewer messages have to be created, marshalled, and transmitted.

5. EPP Extension Framework

Section 2 of [RFC3735] describes how the EPP extension framework can be used to extend EPP functionality by adding new features at the protocol, object and command-response level. This section describes the impact of REPP on each of the extension levels:

- Protocol Extension: Section 9 describes a protocol extension resource for use with existing and future protocol extensions. REPP does not define a new Protocol extension. All existing and future Protocol extension level EPP extensions MAY be used.
- Object extension: REPP does not define any new object level extensions. All existing and future object level EPP extensions MAY be used.
- Command-Response extension: Section 9 describes a Command-Response extension resource for each object mapping and can be used for existing and future command extensions. REPP does not define a new Command-Response extension. All existing and future Command-Response extension level EPP extensions MAY be used.

6. Resource Naming Convention

A REPP resource can be a single unique object identifier e.g. a domain name or consist out of a collection of objects. A collection of objects available for registry operations MUST be identified by: /{context-root}/{version}/{collection}

- {context-root} is the base URL which MUST be specified, the {context-root} MAY be an empty, zero length string.
- {version} is a path segment which identifies the version of the REPP implementation. This is the equivalent of the Version element in the EPP RFCs. The version used in the REPP URL MUST match the version used in EPP Greeting message.
- {collection} MUST be substituted by "domains", "hosts" or "contacts" or other supported objects, referring to either [RFC5731], [RFC5732] or [RFC5733].

A trailing slash MAY be added to each request. Implementations MUST consider requests which only differ with respect to this trailing slash as identical.

A specific EPP object instance MUST be identified by {context-root}/ {version}/ {collection}/{id} where {id} is a unique object identifier described in EPP RFCs.

An example domain name resource, for domain name example.nl, would look like this:

`/repp/v1/domains/example.nl`

The path segment after a collection path segment MUST be used to identify an object instance, the path segment after an object instance MUST be used to identify attributes or related collections of the object instance.

Resource URLs used by REPP contain embedded object identifiers. By using an object identifier in the resource URL, the object identifier in the request messages becomes superfluous. However, since the goal of REPP is to maintain compatibility with existing EPP object mapping schemas, this redundancy is accepted as a tradeoff. Removing the object identifier from the request message would require updating the object mapping schemas in the EPP RFCs.

The server MUST return HTTP status code 412 when the object identifier, for example domain:name, host:name or contact:id, in the EPP request message does not match the {id} object identifier embedded in the URL.

7. Session Management

One of the main design considerations for REPP is to enable scalable EPP services, for this reason the REPP uses a stateless architecture and does not create and maintain client sessions. The Session concept is an anti-pattern in the context of a stateless service, the server MUST NOT maintain any state information relating to the client or EPP transaction.

Session management as described in [RFC5730] requires a stateful server architecture for maintaining client and application state over multiple client request and is therefore no longer supported.

A REPP request MUST contain all information required for the server to be able to successfully process the request. The client MUST include authentication credentials for each request. This MAY be done by using any of the available HTTP authentication mechanisms, such as those described in [RFC2617].

A REPP server MUST listen for HTTP connection requests on the standard TCP port assigned in [RFC2616]. After a connection has been established, the server MUST NOT return a Greeting message. The server MAY close open TCP connections when these violate server policies, for instance connections having a long inactivity period or a long connection lifetime.

8. REST

REPP uses the REST architectural style, each HTTP method is assigned a distinct behavior, Section 8.1 provides an overview of the behavior assigned to each method. REPP requests are expressed by a URL referring to a resource, a HTTP method, HTTP headers and an optional message body containing the EPP request message.

A REPP HTTP message body MUST contain at most a single EPP request or response. HTTP requests MUST be processed independently of each other and in the same order as received by the server. A client MAY choose to send a new request, using an existing connection, before the response for the previous request has been received (pipelining). A server using HTTP/2 [RFC7540] or HTTP/3 [RFC9114] contains built-in support for stream multiplexing and MAY choose to support pipelining using this mechanism. The response MAY be returned out of order back to the client, because some requests require more processing time by the server.

HTTP/1 does not use persistent connections by default, the client MAY use the "Connection" header to request for the server not to close the existing connection, so it can be re-used for future requests. The server MAY choose not to honor this request.

8.1. Method Definition

REPP commands MUST be executed by using an HTTP method on a resource identified by an URL. The server MUST support the following methods.

- GET: Request a representation of an object resource or a collection of resources
- PUT: Update an existing object resource
- PATCH: Partially update an existing object resource
- POST: Create a new object resource
- DELETE: Delete an existing object resource
- HEAD: Check for the existence of an object resource
- OPTIONS: Request a greeting

8.2. Content negotiation

The server MAY choose to support multiple data format for EPP object representations, such as XML and JSON. The client and server MUST support server-driven content negotiation and related HTTP headers for content negotiation, as described in Section 12.2 of [RFC2616].

The client MUST use the following HTTP headers:

- Content-Type: Used to indicate the media type for the content in the message body
- Accept: Used to indicate the media type the server MUST use for the representation of objects, this MAY be a list of types and related weight factors, as described in Section 14.1 of [RFC2616]

The client MUST synchronize the value for the Content-Type and Accept headers, for example a client MUST NOT send an XML formatted request message to the server, while at the same time requesting a JSON formatted response message. The server MUST use the Content-Type HTTP header to indicate the media type used for the representation in the response message body. The server MUST return HTTP status code 406 (Not Acceptable) or 415 (Unsupported Media Type) when the client requests an unsupported media type.

8.3. Request

In contrast to EPP over TCP [RFC5734], a REPP request does not always require a EPP request message. The information conveyed by the HTTP method, URL, and request headers may be sufficient for the server to be able to successfully processes a request for most commands. However, the client MUST include the request message in the HTTP request body when the server uses an EPP extension that requires additional XML elements or attributes to be present in the request message. All REPP HTTP headers listed below use the "REPP-" prefix, following the recommendations in [RFC6648].

- REPP-Cltrid: The client transaction identifier is the equivalent of the clTRID element defined in [RFC5730] and MUST be used accordingly, when the HTTP message body does not contain an EPP request that includes a cltrid.
- REPP-Svcs: The namespace used by the client in the EPP request message, this is equivalent to the "svcs" element in the Login command defined in Section 2.9.1.1 of [RFC5730]. The client MUST use this header if the media type of the request or response message body content requires the server to know what namespaces to use. Such as is the case for XML-based request and response messages. The header value MAY contain multiple comma separated namespaces.
- REPP-Svcs-Ext: The extension namespace used by the client in the EPP request message, this is equivalent to the "svcExtension" element in the Login command defined in Section 2.9.1.1 of [RFC5730]
- REPP-AuthInfo: The client MAY use this header for sending basic token-based authorization information, as described in Section 2.6 of [RFC5731] and Section 2.8 of [RFC5733]. If the authorization is linked to a contact object then the client MUST also include the REPP-Roid header.
- REPP-Roid: If the authorization info, is linked to a database object, the client MAY use this header for the Repository Object IDentifier (ROID), as described in Section 4.2 of [RFC5730].
- Accept-Language: This header is equivalent to the "lang" element of the EPP Login command. The server MUST support the use of HTTP Accept-Language header by clients. The client MAY issue a Hello request to discover the languages supported by the server. Multiple servers in a load-balanced environment SHOULD reply with consistent "lang" elements in the Greeting response. The value of the Accept-Language header MUST match 1 of the languages from the Greeting. When the server receives a request using an unsupported language, the server MUST respond using the default language configured for the server, as required in Section 2.9.1.1 of [RFC5730]
- Connection: If the server uses HTTP/1.1 or lower, the CLIENT MAY choose to use this header to request the server to keep op the TCT-connection. The client MUST not use this header when the server uses HTTP/2 Section 8.2.2 of [RFC9113] or HTTP/3 Section 4.2 of [RFC9113]
- Accept-Encoding: The client MAY choose to use the Accept-Encoding HTTP header to request the server to use compression for the response message body.

8.4. Response

The server HTTP response contains a status code, headers, and MAY contain an EPP response message in the message body. HTTP headers are used to transmit additional data to the client and MAY be used to send EPP process related data to the client. HTTP headers used by REPP MUST use the "REPP-" prefix, the following response headers have been defined for REPP.

- REPP-Svtrid: This header is the equivalent of the "svTRID" element defined in [RFC5730] and MUST be used accordingly when the REPP response does not contain an EPP response in the HTTP message body. If an HTTP message body with the EPP XML equivalent "svTRID" exists, both values MUST be consistent.
- REPP-Cltrid: This header is the equivalent of the "clTRID" element defined in [RFC5730] and MUST be used accordingly when the REPP response does not contain an EPP response in the HTTP message body. If the contents of the HTTP message body contains a "clTRID" value, then both values MUST be consistent.
- REPP-Eppcode: This header is the equivalent of the EPP result code defined in [RFC5730] and MUST be used accordingly. This header MUST be added to all responses, except for the Greeting, and MAY be used by the client for easy access to the EPP result code, without having to parse the content of the HTTP response message body.
- REPP-Check-Avail: An alternative for the "avail" attribute of the object:name element in an Object Check response and MUST be used accordingly. The server does not return a HTTP message body in response to a REPP Object Check request.
- REPP-Check-Reason: An optional alternative for the "object:reason" element in an Object Check response and MUST be used accordingly.
- REPP-Queue-Size: Return the number of unacknowledged messages in the client message queue. The server MAY include this header in all REPP responses.
- Cache-Control: The client MUST never cache results, the server MUST always return the value "No-Store" for this header, as described in Section 5.2.1.5 of [RFC7234].
- Content-Language: The server MUST include this header in every response that contains an EPP message in the message body.
- Content-Encoding: The server MAY choose to compress the responses message body, using an algorithm selected from the list of algorithms provided by the client using the Accept-Encoding request header.

REPP does not always return an EPP response message in the HTTP message body. The Object Check request for example may return an empty HTTP response body. When the server does not return an EPP message, it MUST return at least the REPP-Svtrid, REPP-Cltrid and REPP-Eppcode headers.

8.5. Error Handling

Restful EPP and HTTP protocol are both an application layer protocol, having their own status- and result codes. The endpoints described in Section 9 MUST return HTTP status code 200 (OK) for successful requests when the EPP result code indicates a positive completion (1xxx) of the EPP command.

When an EPP command results in a negative completion result code (2xxx), the server MUST return the HTTP status code 422 (Unprocessable Content). A more detailed explanation of the EPP error MUST be included in the message body of the HTTP response, as described in [RFC9110], but only when this is permitted for the used HTTP method. Errors related to the HTTP protocol MUST result in the use of an appropriate HTTP status code by the HTTP server. An error or problem while processing one request MUST NOT result in the failure of other independent requests using the same connection.

The client MUST be able to use the best practices for RESTful applications and use the HTTP status code to determine if the EPP request was successfully processed. The client MAY use the well defined HTTP status code and REPP-Eppcode HTTP header for error handling logic, without having to parse the EPP result code in the message body.

For example, a client sending an Object Transfer request for an Object already linked to an active transfer process, will result in an EPP result code 2106, the HTTP response contains a status code 422 and he value for the REPP-Eppcode HTTP header is set to 2106. The client MAY use the HTTP status code for checking if an EPP command failed and only parse the result message when additional information from the response message is required for handling the error.

9. Command Mapping

EPP commands are mapped to RESTful EPP requests using four elements.

1. Resource defined by a URL
2. HTTP method to be used on the resource
3. EPP request message
4. EPP response message

Table 1 lists a mapping for each EPP command to a REPP request, the subsequent sections provide details for each request. Resource URLs in the table are assumed to be using the prefix: "/{context-root}/{version}/". Some REPP endpoints do not require a request and/or response message, as is indicated by the table columns "Request" and "response".

- {c}: An abbreviation for {collection}: this MUST be substituted with "domains", "hosts", "contacts" or any other collection of objects.
- {i}: An abbreviation for an object id, this MUST be substituted with the value of a domain name, hostname, contact-id or a message-id or any other defined object.
- Optional: A request message is only required when the server uses an EPP extension, which requires the use of XML elements and/or atributes that are not mapped to REPP.

Command	Method	Resource	Request	Response
Hello	OPTIONS	/	No	Yes
Login	N/A	N/A	N/A	N/A
Logout	N/A	N/A	N/A	N/A
Check	HEAD	/{c}/{i}	Optional	No

Command	Method	Resource	Request	Response
Info	GET	/{c}/{i}	Optional	Yes
Poll Request	GET	/messages	No	Yes
Poll Ack	DELETE	/messages/{i}	No	Yes
Create	POST	/{c}	Yes	Yes
Delete	DELETE	/{c}/{i}	Optional	Yes
Renew	POST	/{c}/{i}/renewals	Optional	Yes
Transfer Request	POST	/{c}/{i}/transfers	Optional	Yes
Transfer Query	GET	/{c}/{i}/transfers/latest	Optional	Yes
Transfer Cancel	DELETE	/{c}/{i}/transfers/latest	Optional	Yes
Transfer Approve	PUT	/{c}/{i}/transfers/latest	Optional	Yes
Transfer Reject	DELETE	/{c}/{i}/transfers/latest	Optional	Yes
Update	PATCH	/{c}/{i}	Yes	Yes
Extension [1]	*	/{c}/{i}/extension/*	*	*
Extension [2]	*	/extension/*	*	*

Table 1: Mapping of EPP Command to REPP Request

[1] This mapping is used for Object extensions based on the extension mechanism as defined in [RFC5730, section 2.7.2] [2] This mapping is used for Protocol extensions based on the extension mechanism as defined in [RFC5730, seciton 2.7.1]

When there is a mismatch between a resource identifier in the HTTP message body and the resource identifier in the URL used for a request, then the server MUST return HTTP status code 400 (Bad Request). The examples, in the sections below, assume the server does not use any EPP extensions and therefore the client does not add any request message to the HTTP message body.

9.1. Hello

- Request: OPTIONS /
- Request message: None
- Response message: Greeting response

Due to the stateless nature of REPP, the server does not respond by sending a Greeting message when a connection is created, as described in Section 2 of [RFC5730]. The client MUST request a Greeting by using the Hello request as described in Section 2.3 of [RFC5730]. The server MUST respond by returning a Greeting response, as defined in Section 2.4 of [RFC5730].

The version value used in the Hello response MUST match the version value used for the `{version}` path segment in the URL used for the Hello request.

Example request:

```
C: OPTIONS /repp/v1/ HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: Connection: keep-alive
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 799
S: Content-Type: application/epp+xml
S: Content-Language: en
S:
S: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
S: <repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:   <greeting>
S:     <svcMenu>
S:       <version>1.0</version>
S:       <!-- The rest of the response is omitted here -->
S:     <svcMenu>
S:   </greeting>
S: </repp>
```

9.2. Login

The Login command defined in Section 2.9.1.1 of [RFC5730] is used to establish a session between client and server, this is part of the stateful nature of the EPP protocol. REPP is stateless and MUST NOT maintain any client state and does not include a Login command. The client MUST include all information in a REPP request, required for the server to be able to properly process the request. This includes request attributes defined as for Login command in Section 2.9.1.1 of [RFC5730].

The request attributes from the Login command, used for configuring the client session, are moved to the HTTP layer.

- `clID`: Replaced by HTTP authentication
- `pw`:: Replaced by HTTP authentication
- `newPW`: Replaced by out of band process
- `version`: Replaced by the `{version}` path parameter in the request URL.
- `lang`: Replaced by the `Accept-Language` HTTP header.

- svcs: Replaced by the REPP-Svcs HTTP header.
- svcExtension: Replaced by the REPP-Svcs-Ext HTTP header.

The server MUST check the namespaces used in the REPP-Svcs HTTP header. An unsupported namespace MUST result in the appropriate EPP result code.

9.3. Logout

Due to the stateless nature of REPP, the session concept is no longer used and therefore the Logout command MUST NOT be implemented by the server.

9.4. Query Resources

A REPP client MAY use the HTTP GET method for executing a query command only when no request data has to be added to the HTTP message body. Sending content using an HTTP GET request is discouraged in [RFC9110], there exists no generally defined semantics for content received in a GET request. When an EPP object requires additional authInfo information, as described in [RFC5731] and [RFC5733], the client MUST use the HTTP POST method and add the query command content to the HTTP message body.

9.4.1. Check

- Request: HEAD /{collection}/{id}
- Request message: Optional
- Response message: None

The HTTP HEAD method MUST be used for object existence check. The response MUST contain the REPP-Check-Avail header and MAY contain the REPP-Check-Reason header. The value of the REPP-Check-Avail header MUST be "0" or "1" as described in Section 2.9.2.1 of [RFC5730], depending on whether the object can be provisioned or not.

The Check endpoint MUST be limited to checking only a single object-id per request. This may seem a limitation compared to the Check command defined in [RFC5730] where a Check message may contain multiple object-ids. The REPP Check request can be load balanced more efficiently when only a single object-id has to be checked.

Example request for a domain name:

```
C: HEAD /repp/v1/domains/example.nl HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: REPP-Cltrid: ABC-12345
S: REPP-Svtrid: XYZ-12345
S: REPP-Check-Avail: 0
S: REPP-Check-Reason: In use
S: REPP-result-code: 1000
s: Content-Length: 0
```

9.4.2. Info

The Object Info request MUST use the HTTP GET method on a resource identifying an object instance, using an empty message body. If the object has authorization information attached and the authorization then the client MUST include the REPP-AuthInfo HTTP header. If the authorization is linked to a database object the client MUST include the REPP-Roid header.

Example request for an object not using authorization information.

- Request: GET /{collection}/{id}
- Request message: Optional
- Response message: Info response

```
C: GET /repp/v1/domains/example.nl HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example request using REPP-AuthInfo header for an object that has attached authorization information.

- Request: GET /{collection}/{id}
- Request message: Optional
- Response message: Info response

```
C: GET /repp/v1/domains/example.nl HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: REPP-AuthInfo: secret-token
C: REPP-Roid: REG-XYZ-12345
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example Info response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 424
S: Content-Type: application/epp+xml
S: Content-Language: en
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <domain:infData xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        < !-- The rest of the response is omitted here -- >
S:      </domain:infData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.4.2.1. Object Filtering

The server MUST support the use of the `filter` and `val` query parameters for the purpose of limiting the number of objects in a response.

- `filter`: The attribute or field name to apply the filter on
- `val`: The value used for filtering

The Domain Name Mapping Section 3.1.2 describes an optional "hosts" attribute for the Domain Info command. This attribute may be used for filtering hosts returned in the Info response, and is mapped to the filter and val query parameters. If the filtering query parameters are absent from the request URL, the server MUST use the default filter value described in the corresponding EPP RFCs.

URLs used for filtering based on hosts attribute for Domain Info request:

- default: GET /domains/{id}
- all: GET /domains/{id}?filter=hosts&val=all
- del: GET /domains/{id}?filter=hosts&val=del
- sub: GET /domains/{id}?filter=hosts&val=sub
- none: GET /domains/{id}?filter=hosts&val=none

Example Domain Info request, the response should only include delegated hosts:

```
C: GET /repp/v1/domains/example.nl?filter=hosts&val=del HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

9.4.3. Poll

9.4.3.1. Poll Request

- Request: GET /messages
- Request message: None
- Response message: Poll response

The client MUST use the HTTP GET method on the messages resource collection to request the message at the head of the queue. The "op=req" semantics from Section 2.9.2.3 are assigned to the HTTP GET method.

Example request:

```
C: GET /repp/v1/messages HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 312
S: Content-Type: application/epp+xml
S: Content-Language: en
S: REPP-Eppcode: 1301
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1301">
S:      <msg>Command completed successfully; ack to dequeue</msg>
S:    </result>
S:    <msgQ count="5" id="12345">
S:      <qDate>2000-06-08T22:00:00.0Z</qDate>
S:      <msg>Transfer requested.</msg>
S:    </msgQ>
S:    <resData>
S:       <!-- The rest of the response is omitted here -->
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.4.3.2. Poll Ack

- Request: DELETE /messages/{id}
- Request message: None
- Response message: Poll Ack response

The client MUST use the HTTP DELETE method to acknowledge receipt of a message from the queue. The "op=ack" semantics from Section 2.9.2.3 are assigned to the HTTP DELETE method. The "msgID" attribute of a received EPP Poll message MUST be included in the message resource URL, using the {id} path element. The server MUST use REPP headers to return the EPP result code and the number of messages left in the queue. The server MUST NOT add content to the HTTP message body of a successful response, the server may add content to the message body of an error response.

Example request:

```
C: DELETE /repp/v1/messages/12345 HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: REPP-Eppcode: 1000
S: REPP-Queue-Size: 0
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: Content-Length: 145
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <msgQ count="0" id="12345"/>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.4.4. Transfer Query

The Transfer Query request MUST use the special "latest" sub-resource to refer to the latest object transfer. A latest transfer object may not exist, when no transfer has been initiated for the specified object. The client MUST use the HTTP GET method and MUST NOT add content to the HTTP message body.

- Request: GET {collection}/{id}/transfers/latest
- Request message: Optional
- Response message: Transfer Query response

Example domain name Transfer Query request without authorization information required:

```
C: GET /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

If the requested object has associated authorization information that is not linked to another database object, then the HTTP GET method MUST be used and the authorization information MUST be included using the REPP-AuthInfo header.

Example domain name Transfer Query request using REPP-AuthInfo header:

```
C: GET /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: REPP-AuthInfo: secret-token
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

If the requested object has associated authorization information linked to another database object, then the HTTP GET method MUST be used and both the REPP-AuthInfo and the REPP-Roid header MUST be included.

Example domain name Transfer Query request and authorization using REPP-AuthInfo and the REPP-Roid header:

```
C: GET /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-AuthInfo: secret-token
C: REPP-Roid: REG-XYZ-12345
C: Content-Length: 0
C:
```

Example Transfer Query response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 230
S: Content-Type: application/epp+xml
S: Content-Language: en
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <!-- The rest of the response is omitted here -->
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5. Transform Resources

9.5.1. Create

- Request: POST /{collection}
- Request message: Object Create request
- Response message: Object Create response

The client MUST use the HTTP POST method to create a new object resource. If the EPP request results in a newly created object, then the server MUST return HTTP status code 200 (OK). The server MUST add the "Location" header to the response, the value of this header MUST be the URL for the newly created resource.

Example Domain Create request:

```
C: POST /repp/v1/domains HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Content-Type: application/epp+xml
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: Content-Length: 220
C:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
C:  <request>
C:    <body>
C:      <domain:create
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:        <!-- The rest of the request is omitted here -->
C:      </domain:create>
C:    </body>
C:    <clTRID>ABC-12345</clTRID>
C:  </request>
C:</repp>
```

Example Domain Create response:

```
S: HTTP/2 200
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: Content-Length: 642
S: Content-Type: application/epp+xml
S: Location: https://repp.example.nl/repp/v1/domains/example.nl
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0"
S:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:   <response>
S:      <result code="1000">
S:        <msg>Command completed successfully</msg>
S:      </result>
S:      <resData>
S:        <domain:creData>
S:          <!-- The rest of the response is omitted here -->
S:        </domain:creData>
S:      </resData>
S:      <trID>
S:        <clTRID>ABC-12345</clTRID>
S:        <svTRID>XYZ-12345</svTRID>
S:      </trID>
S:   </response>
S:</repp>
```

9.5.2. Delete

- Request: DELETE /{collection}/{id}
- Request message: Optional
- Response message: Status

The client MUST the HTTP DELETE method and a resource identifying a unique object instance. The server MUST return HTTP status code 200 (OK) if the resource was deleted successfully.

Example Domain Delete request:

```
C: DELETE /repp/v1/domains/example.nl HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
```

Example Domain Delete response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5.3. Renew

- Request: POST /{collection}/{id}/renewals
- Request message: Optional
- Response message: Renew response

The Renew command is mapped to a nested collection, named "renewals". Not all EPP object types include support for the renew command. The current-date query parameter MAY be used for date on which the current validity period ends, as described in Section 3.2.3 of [RFC5731]. The new period MAY be added to the request using the unit and value request parameters. The response MUST include the Location header for the renewed object.

Example Domain Renew request:

```
C: POST /repp/v1/domains/example.nl/renewals?current-date=2024-01-01 HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Content-Type: application/epp+xml
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: Content-Length: 0
C:
```

Example Domain Renew request, using 1 year period:

```
C: POST /repp/v1/domains/example.nl/renewals?current-date=2024-01-01?
unit=y&value=1 HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Content-Type: application/epp+xml
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: Content-Length: 0
C:
```

Example Renew response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: Content-Length: 205
S: Location: https://repp.example.nl/repp/v1/domains/example.nl
S: Content-Type: application/epp+xml
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <!-- The rest of the response is omitted here -->
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5.4. Transfer

Transferring an object from one sponsoring client to another client is specified in [RFC5731] and [RFC5733]. The Transfer command is mapped to a nested resource, named "transfers". The semantics of the HTTP DELETE method are determined by the role of the client executing the DELETE method. For the current sponsoring client of the object, the DELETE method is defined as "reject transfer". For the new sponsoring client the DELETE method is defined as "cancel transfer".

9.5.4.1. Request

- Request: POST /{collection}/{id}/transfers
- Request message: Optional
- Response message: Status

To start a new object transfer process, the client MUST use the HTTP POST method for a unique resource to create a new transfer resource object, not all EPP objects support the Transfer command as described in Section 3.2.4 of [RFC5730], Section 3.2.4 of [RFC5731] and Section 3.2.4 of [RFC5733].

If the transfer request is successful, then the response MUST include the Location header for the object being transferred.

Example request not using object authorization:

```
C: POST /repp/v1/domains/example.nl/transfers HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example request using object authorization:

```
C: POST /repp/v1/domains/example.nl/transfers HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: REPP-Cltrid: ABC-12345
C: REPP-AuthInfo: secret-token
C: Accept-Language: en
C: Content-Length: 0
```

Example request using 1 year renewal period, using the unit and value query parameters:

```
C: POST /repp/v1/domains/example.nl/transfers?unit=y&value=1 HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example Transfer response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: Content-Length: 328
S: Content-Type: application/epp+xml
S: Location: https://repp.example.nl/repp/v1/domains/example.nl/transfers/latest
S: REPP-Eppcode: 1001
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1001">
S:      <msg>Command completed successfully; action pending</msg>
S:    </result>
S:    <resData>
S:      <!-- The rest of the response is omitted here -->
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5.4.2. Cancel

- Request: DELETE /{collection}/{id}/transfers/latest
- Request message: Optional
- Response message: Status

The new sponsoring client MUST use the HTTP DELETE method to cancel a requested transfer. The semantics of the HTTP DELETE method are determined by the role of the client sending the request.

Example request:

```
C: DELETE /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5.4.3. Reject

- Request: DELETE /{collection}/{id}/transfers/latest
- Request message: None
- Response message: Status

The semantics of the HTTP DELETE method are determined by the role of the client sending the request. For the current sponsoring client of the object, the DELETE method is defined as "reject transfer".

Example request:

```
C: DELETE /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
```

Example Reject response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5.4.4. Approve

- Request: PUT /{collection}/{id}/transfers/latest
- Request message: Optional
- Response message: Status

The current sponsoring client MUST use the HTTP PUT method to approve a transfer requested by the new sponsoring client.

Example Approve request:

```
C: PUT /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example Approve response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.5.5. Update

- Request: PATCH /{collection}/{id}
- Request message: Object Update message
- Response message: Status

An object Update request MUST be performed using the HTTP PATCH method. The request message body MUST contain an EPP Update request, and the object-id value in the request MUST match the value of the object-id path parameter in the URL.

Example request:

```
C: PATCH /repp/v1/domains/example.nl HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Content-Type: application/epp+xml
C: Accept-Language: en
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Content-Length: 252
C:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
C:  <request>
C:    <body>
C:      <domain:update
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:          <!-- The rest of the request is omitted here -->
C:      </domain:update>
C:    </body>
C:    <clTRID>ABC-12345</clTRID>
C:  </request>
C:</repp>
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.6. Extension Framework

The EPP Extension Framework allows for extending the EPP protocol at different locations, REPP defines additional REST resources for the Protocol and Command-Response extensions.

9.6.1. Protocol Extension

- Request: * /extensions/*
- Request message: *
- Response message: *

EPP Protocol extensions, defined in Section 2.7.1 of [RFC5730] are supported using the "/extensions" root resource. The HTTP method used for a new Protocol extension is not defined but must follow the RESTful principles.

The example below, illustrates the use of the "Domain Cancel Delete" command as defined as a custom command in [SIDN-EXT]. The new command is created below the "extensions" path element and after this element follows the "domains" object collection, finally a special "deletion" path element is added to the end of the URL. A client MUST use the HTPP DELETE method on a domain name deletion resource to cancel an ongoing domain delete transaction and move the domain from the grace state back to the active state.

Example Protocol Extension request:

- Request: DELETE /extensions/{collection}/{id}/deletion
- Request message: Optional
- Response message: Optional error response

```
C: DELETE /repp/v1/extensions/domains/example.nl/deletion HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: REPP-Svcs-Ext: https://rxsd.domain-registry.nl/sidn-ext-epp-1.0
C: REPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: Content-Length: 0
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</repp>
```

9.6.2. Object Extension

An Object extension is differs from the other 2 extension types in the way that an Object extension is implemented using a new Object mapping for a new Object type, while re-using the existing EPP command and response structures. The newly created Object mapping, is similar to the existing Object mappings defined in [RFC5731], [RFC5732] and [RFC5733], and MUST be used in a similar fashion.

A hypothetical new Object mapping for IP addresses, may result in a new resource collection named "ips", the semantics for the HTTP methods would have to be defined. Creating a new IP address may use the HTTP POST method on the "ips" collection.

- Request: POST /{collection}/{id}
- Request message: IP Create Request message
- Response message: IP Create Response message

Example request:

```
C: POST /repp/v1/ips HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-Svcs-Ext: https://example.nl/epp-ips-1.0
C: REPP-Cltrid: ABC-12345
C: Content-Type: application/epp+xml
C: Content-Length: 220
C:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0">
C:  <request>
C:    <body>
C:      <ip:create
C:       xmlns:ip="https://example.nl/epp-ip-1.0">
C:        <ip:address>192.0.2.1</ip:address>
C:        <!-- The rest of the request is omitted here -->
C:      </ip:create>
C:    </body>
C:    <clTRID>ABC-12345</clTRID>
C:  </request>
C:</repp>
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: Content-Length: 642
S: Content-Type: application/epp+xml
S: Location: https://repp.example.nl/repp/v1/ips/192.0.2.1
S: REPP-Eppcode: 1000
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<repp xmlns="urn:ietf:params:xml:ns:repp-1.0"
S:     xmlns:ip="https://example.nl/epp-ip-1.0">
S:   <response>
S:     <result code="1000">
S:       <msg>Command completed successfully</msg>
S:     </result>
S:     <resData>
S:       <ip:creData>
S:         <!-- The rest of the response is omitted here -->
S:       </ip:creData>
S:     </resData>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>XYZ-12345</svTRID>
S:     </trID>
S:   </response>
S:</repp>
```

9.6.3. Command-Response Extension

Command-Response Extensions allow for adding elements to an existing object mapping, therefore no new extension resource is required, the existing resources can be used for existing and future extensions of this type.

10. Protocol Considerations

Section 2.1 of [RFC5730] of the EPP protocol specification describes considerations to be addressed by a transport or protocol mapping. These are satisfied by a combination of REPP features and features provided by HTTP protocol and underlying transport protocols, as described below.

- The consideration: "The transport mapping MUST preserve the stateful nature of the protocol", is updated to: "The transport mapping MUST preserve the stateful nature of the protocol, when using a stateful transport protocol". REPP uses the REST architectural style for defining a stateless API based on the stateless HTTP protocol, and therefore satisfies the updated consideration.
- Section 8 describes how HTTP multiplexing may be used for pipelining multiple requests. A server may allow pipelining, requests are to be processed in the order they have been received.
- REPP is based on the HTTP protocol, which uses the client-server model.

- REPP messages are transmitted using HTTP, this document refers to the HTTP [RFC2616] protocol specification for how data units are framed.
- HTTP/1 and HTTP/2 use TCP as a transport protocol and this includes features to provide reliability, flow control, ordered delivery, and congestion control Section 1.5 of [RFC793] describes these features in detail; congestion control principles are described further in [RFC2581] and [RFC2914]. HTTP/3 uses QUIC (UDP) as a transport protocol, which has built-in congestion control over UDP.
- Section 8 describes how requests are processed independently of each other.
- Errors while processing a REPP request are isolated to this request and do not affect other requests sent by the client or other clients, this is described in Section 8.5.
- Batch-oriented processing (combining multiple EPP commands in a single HTTP request) is not permitted. To maximize scalability every request must contain a single command, as described in Section 8.

11. Formal Syntax

This section contains the XML Schema notation defined for REPP, based on the XML schema defined in [RFC5730]. The XML schema defined in [RFC5730] contains XML elements and attributes that are no longer required in a REPP context.

The following changes have been made:

- deleted hello from eppType
- renamed command to request in eppType
- deleted choice and all child elements from commandtype
- renamed commandtype to requestType
- renamed readWriteType to bodyType
- created body element for requestType
- deleted loginType
- deleted credsOptionsType
- deleted loginSvcType
- deleted pwType
- deleted pollType
- deleted transferType
- deleted transferOpType

The formal syntax presented here is a complete schema representation of REPP suitable for automated validation of REPP XML instances.

```
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="urn:ietf:params:xml:ns:repp-1.0"
xmlns:repp="urn:ietf:params:xml:ns:repp-1.0"
xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

  <!--
```

```
Import common element types.
-->
  <import namespace="urn:ietf:params:xml:ns:eppcom-1.0"/>

  <annotation>
    <documentation>
      RESTful Extensible Provisioning Protocol v1.0 schema.
    </documentation>
  </annotation>

<!--
Every EPP XML instance must begin with this element.
-->

<element name="repp" type="repp:eppType"/>

<!--
An EPP XML instance must contain a greeting, request, response, or extension.
-->
  <complexType name="eppType">
    <choice>
      <element name="greeting" type="repp:greetingType"/>
      <element name="request" type="repp:requestType"/>
      <element name="response" type="repp:responseType"/>
      <element name="extension" type="repp:extAnyType"/>
    </choice>
  </complexType>

<!--
A greeting is sent by a server in response to a client connection
or <hello>.
-->
  <complexType name="greetingType">
    <sequence>
      <element name="svID" type="repp:sIDType"/>
      <element name="svDate" type="dateTime"/>
      <element name="svcMenu" type="repp:svcMenuType"/>
      <element name="dcp" type="repp:dcpType"/>
    </sequence>
  </complexType>

<!--
Server IDs are strings with minimum and maximum length restrictions.
-->
  <simpleType name="sIDType">
    <restriction base="normalizedString">
      <minLength value="3"/>
      <maxLength value="64"/>
    </restriction>
  </simpleType>

<!--
A server greeting identifies available object services.
-->
  <complexType name="svcMenuType">
    <sequence>
      <element name="version" type="repp:versionType"
       maxOccurs="unbounded"/>
```

```
        <element name="lang" type="language"
         maxOccurs="unbounded"/>
        <element name="objURI" type="anyURI"
         maxOccurs="unbounded"/>
        <element name="svcExtension" type="repp:extURIType"
         minOccurs="0"/>
      </sequence>
    </complexType>

  <!--
  Data Collection Policy types.
  -->
    <complexType name="dcpType">
      <sequence>
        <element name="access" type="repp:dcpAccessType"/>
        <element name="statement" type="repp:dcpStatementType"
         maxOccurs="unbounded"/>
        <element name="expiry" type="repp:dcpExpiryType"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="dcpAccessType">
      <choice>
        <element name="all"/>
        <element name="none"/>
        <element name="null"/>
        <element name="other"/>
        <element name="personal"/>
        <element name="personalAndOther"/>
      </choice>
    </complexType>

    <complexType name="dcpStatementType">
      <sequence>
        <element name="purpose" type="repp:dcpPurposeType"/>
        <element name="recipient" type="repp:dcpRecipientType"/>
        <element name="retention" type="repp:dcpRetentionType"/>
      </sequence>
    </complexType>

    <complexType name="dcpPurposeType">
      <sequence>
        <element name="admin"
         minOccurs="0"/>
        <element name="contact"
         minOccurs="0"/>
        <element name="other"
         minOccurs="0"/>
        <element name="prov"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="dcpRecipientType">
      <sequence>
        <element name="other"
         minOccurs="0"/>
```

```
        <element name="ours" type="repp:dcpOursType"
         minOccurs="0" maxOccurs="unbounded"/>
        <element name="public"
         minOccurs="0"/>
        <element name="same"
         minOccurs="0"/>
        <element name="unrelated"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="dcpOursType">
      <sequence>
        <element name="recDesc" type="repp:dcpRecDescType"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <simpleType name="dcpRecDescType">
      <restriction base="token">
        <minLength value="1"/>
        <maxLength value="255"/>
      </restriction>
    </simpleType>

    <complexType name="dcpRetentionType">
      <choice>
        <element name="business"/>
        <element name="indefinite"/>
        <element name="legal"/>
        <element name="none"/>
        <element name="stated"/>
      </choice>
    </complexType>

    <complexType name="dcpExpiryType">
      <choice>
        <element name="absolute" type="dateTime"/>
        <element name="relative" type="duration"/>
      </choice>
    </complexType>

  <!--
  Extension framework types.
  -->
    <complexType name="extAnyType">
      <sequence>
        <any namespace="##other"
         maxOccurs="unbounded"/>
      </sequence>
    </complexType>

    <complexType name="extURIType">
      <sequence>
        <element name="extURI" type="anyURI"
         maxOccurs="unbounded"/>
      </sequence>
    </complexType>
```

```
  <!--
  An EPP version number is a dotted pair of decimal numbers.
  -->
    <simpleType name="versionType">
      <restriction base="token">
        <pattern value="[1-9]+\.[0-9]+"/>
        <enumeration value="1.0"/>
      </restriction>
    </simpleType>

  <!--
  Request type, reduced to a single <body> element for object-data.
  -->

<complexType name="requestType">
  <sequence>
    <element name="body" type="repp:bodyType"/>
    <element name="extension" type="repp:extAnyType"
     minOccurs="0"/>
    <element name="clTRID" type="repp:trIDStringType"
     minOccurs="0"/>
  </sequence>
</complexType>


  <!--
    All other object-centric request bodies. EPP doesn't specify the syntax or
    semantics of object-centric body elements.
    The elements MUST be described in detail in another schema specific to the object.
  -->
    <complexType name="bodyType">
      <sequence>
        <any namespace="##other"/>
      </sequence>
    </complexType>

    <complexType name="trIDType">
      <sequence>
        <element name="clTRID" type="repp:trIDStringType"
         minOccurs="0"/>
        <element name="svTRID" type="repp:trIDStringType"/>
      </sequence>
    </complexType>

    <simpleType name="trIDStringType">
      <restriction base="token">
        <minLength value="3"/>
        <maxLength value="64"/>
      </restriction>
    </simpleType>

  <!--
  Response types.
  -->
    <complexType name="responseType">
      <sequence>
        <element name="result" type="repp:resultType"
```

```
        maxOccurs="unbounded"/>
      <element name="msgQ" type="repp:msgQType"
       minOccurs="0"/>

      <element name="resData" type="repp:extAnyType"
       minOccurs="0"/>
      <element name="extension" type="repp:extAnyType"
       minOccurs="0"/>
      <element name="trID" type="repp:trIDType"/>
    </sequence>
  </complexType>

  <complexType name="resultType">
    <sequence>
      <element name="msg" type="repp:msgType"/>
      <choice minOccurs="0" maxOccurs="unbounded">
        <element name="value" type="repp:errValueType"/>
        <element name="extValue" type="repp:extErrValueType"/>
      </choice>
    </sequence>
    <attribute name="code" type="repp:resultCodeType"
     use="required"/>
  </complexType>

  <complexType name="errValueType" mixed="true">
    <sequence>
      <any namespace="##any" processContents="skip"/>
    </sequence>
    <anyAttribute namespace="##any" processContents="skip"/>
  </complexType>

  <complexType name="extErrValueType">
    <sequence>
      <element name="value" type="repp:errValueType"/>
      <element name="reason" type="repp:msgType"/>
    </sequence>
  </complexType>

  <complexType name="msgQType">
    <sequence>
      <element name="qDate" type="dateTime"
       minOccurs="0"/>
      <element name="msg" type="repp:mixedMsgType"
       minOccurs="0"/>
    </sequence>
    <attribute name="count" type="unsignedLong"
     use="required"/>
    <attribute name="id" type="eppcom:minTokenType"
     use="required"/>
  </complexType>

  <complexType name="mixedMsgType" mixed="true">
    <sequence>
      <any processContents="skip"
       minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="lang" type="language"
     default="en"/>
```

```
  </complexType>

<!--
Human-readable text may be expressed in languages other than English.
-->
  <complexType name="msgType">
    <simpleContent>
      <extension base="normalizedString">
        <attribute name="lang" type="language"
         default="en"/>
      </extension>
    </simpleContent>
  </complexType>

<!--
EPP result codes.
-->
  <simpleType name="resultCodeType">
    <restriction base="unsignedShort">
      <enumeration value="1000"/>
      <enumeration value="1001"/>
      <enumeration value="1300"/>
      <enumeration value="1301"/>
      <enumeration value="1500"/>
      <enumeration value="2000"/>
      <enumeration value="2001"/>
      <enumeration value="2002"/>
      <enumeration value="2003"/>
      <enumeration value="2004"/>
      <enumeration value="2005"/>
      <enumeration value="2100"/>
      <enumeration value="2101"/>
      <enumeration value="2102"/>
      <enumeration value="2103"/>
      <enumeration value="2104"/>
      <enumeration value="2105"/>
      <enumeration value="2106"/>
      <enumeration value="2200"/>
      <enumeration value="2201"/>
      <enumeration value="2202"/>
      <enumeration value="2300"/>
      <enumeration value="2301"/>
      <enumeration value="2302"/>
      <enumeration value="2303"/>
      <enumeration value="2304"/>
      <enumeration value="2305"/>
      <enumeration value="2306"/>
      <enumeration value="2307"/>
      <enumeration value="2308"/>
      <enumeration value="2400"/>
      <enumeration value="2500"/>
      <enumeration value="2501"/>
      <enumeration value="2502"/>
    </restriction>
  </simpleType>

<!--
End of schema.
```

```
    -->
    </schema>
```

12. IANA Considerations

[TBD: at the moment we don't see any]

13. Internationalization Considerations

[TBD: any? Accept-Language in HTTP Header]

14. Security Considerations

Running REPP relies on the security of the underlying HTTP [RFC9110] transport, hence the best common practices for securing HTTP also apply to REPP. It is RECOMMENDED to follow them closely.

Data confidentiality and integrity MUST be enforced, all data transport between a client and server MUST be encrypted using TLS [RFC5246]. Section 9 describes the level of security that is REQUIRED for all REPP endpoints.

The EPP Login command, described by [RFC5730], for creating a client session MUST NOT be used anymore. Due to the stateless nature of REPP, the client MUST include the authentication credentials in each HTTP request. This MAY be done by using JSON Web Tokens (JWT) [RFC7519] or Basic authentication [RFC7617].

The management of authentication credentials, such as the "Change password" functionality of the EPP Login command, MUST be performed by an out-of-band process. REPP (HTTP) servers are vulnerable to common denial-of-service attacks. Therefore, the security considerations of [RFC5734] also apply to REPP.

15. Obsolete EPP Result Codes

The following EPP result codes specified in [RFC5730] are no longer meaningful in the context of RESTful EPP and MUST NOT be used.

Code	Reason
1500	Authentication functionality is delegated to the HTTP protocol layer
2100	The REPP URL includes a path segment for the version
2200	Authentication functionality is delegated to the HTTP protocol layer
2501	Authentication functionality is delegated to the HTTP protocol layer

Code	Reason
2502	Rate limiting functionality is delegated to the HTTP protocol layer

Table 2: Obsolete EPP result codes

16. Overview of EPP modifications

This section lists a non-exhaustive overview of the most important modifications made in RESTful EPP, compared to the EPP RFCs.

- The use of HTTP as an additional application layer protocol.
- HTTP adds additional status codes.
- Some Commands are no longer used, such as the Login and Logout command.
- No client sessions, every request needs to include authentication credentials.
- A command MUST only contain a single object to operate on, the check command. For example, the Check command only supports 1 object per request.
- Request messages may no longer be required for most commands
- Authentication and authorizations have become an out-of-band process.
- Support for additional data formats such as JSON.

17. Acknowledgments

The authors would like to thank Miek Gieben who worked with us on an earlier, similar draft.

18. References

18.1. Normative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, DOI 10.17487/RFC1738, December 1994, <https://www.rfc-editor.org/info/rfc1738>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, DOI 10.17487/RFC2581, April 1999, <https://www.rfc-editor.org/info/rfc2581>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <https://www.rfc-editor.org/info/rfc2617>.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://www.rfc-editor.org/info/rfc2914>.

[RFC3735] Hollenbeck, S., "Guidelines for Extending the Extensible Provisioning Protocol (EPP)", RFC 3735, DOI 10.17487/RFC3735, March 2004, <https://www.rfc-editor.org/info/rfc3735>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <https://www.rfc-editor.org/info/rfc5730>.

[RFC5731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", STD 69, RFC 5731, DOI 10.17487/RFC5731, August 2009, <https://www.rfc-editor.org/info/rfc5731>.

[RFC5732] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Host Mapping", STD 69, RFC 5732, DOI 10.17487/RFC5732, August 2009, <https://www.rfc-editor.org/info/rfc5732>.

[RFC5733] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Contact Mapping", STD 69, RFC 5733, DOI 10.17487/RFC5733, August 2009, <https://www.rfc-editor.org/info/rfc5733>.

[RFC5734] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Transport over TCP", STD 69, RFC 5734, DOI 10.17487/RFC5734, August 2009, <https://www.rfc-editor.org/info/rfc5734>.

[RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <https://www.rfc-editor.org/info/rfc6648>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <https://www.rfc-editor.org/info/rfc7617>.

[RFC793] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[XML] W3C, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", 2013, <https://www.w3.org/TR/xml>.

[YAML] YAML Language Development Team, "YAML: YAML Ain't Markup Language", 2000, <https://yaml.org/spec/1.2.2/>.

18.2. Informative References

[SIDN-EXT] SIDN, "Extensible Provisioning Protocol v1.0 schema .NL extensions", 2019, <http://rxsd.domain-registry.nl/sidn-ext-epp-1.0.xsd>.

Authors' Addresses

Maarten Wullink
SIDN Labs
Email: maarten.wullink@sidn.nl
URI: https://sidn.nl/

Marco Davids
SIDN Labs
Email: marco.davids@sidn.nl
URI: https://sidn.nl/