Workgroup:	Network Working Group
Internet-Draft:	draft-wullink-restful-epp-01
Published:	14 December 2023
Intended Status:	Standards Track
Expires:	16 June 2024
Authors:	M. Wullink *SIDN Labs*	M. Davids *SIDN Labs*
---	---	---

Extensible Provisioning Protocol (EPP) RESTful Transport

Abstract

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

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

REPP uses agent-driven content negotiation for supporting multiple presentations, such as XML and JSON.

A server implementing REPP does not maintain client or process state, allowing for scalable EPP services by enabling load balancing at the request level instead of the session level as described in [RFC5734].

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 16 June 2024.

Copyright Notice

Table of Contents

1. Introduction

This document describes a transport protocol for EPP, based on the [REST] architectural style. This transport machanism leverages the HTTP protocol [RFC2616] and the principles of [REST]. Conforming to the REST constraints is generally referred to as being "RESTful". Hence we dubbed the new transport protocol: "'RESTful transport for EPP" or "REPP" for short.

This new REST based transport includes a mapping of [RFC5730] EPP commands to resources based on Uniform Resource Locators [RFC1738]. REPP, in contrast to the EPP specification, is stateless. It aims to provide a mechanism that is more suitable for complex, high availability environments, as well as for environments where TCP connections can be unreliable.

RFC 5730 [RFC5730] Section 2.1 describes that EPP can be layered over multiple transport protocols. Currently, the EPP transport over TCP [RFC5734] is the only widely deployed transport mapping for EPP. This same section defines that newly defined transport mappings must preserve the stateful nature of EPP.

The stateless nature of REPP requires that no session state is maintained on the EPP server. Each client request to the server contains all the information necessary for the server to process the request.

A good understanding of the EPP base protocol specification [RFC5730] is advised, to grasp the command mapping described 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].

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 - A network data object or service that can be 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 resposible for processing requests and returning results in any supported media type.

3. Conventions Used in This Document

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.

4. Design Considerations

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

- Provide a clear, clean, easy to use and self-explanatory interface that can easily be integrated into existing software systems. On the basis of 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 canb be expected to be of the informational type, load balancing and possibly seperating these to dedicated compute resources may also improve registry services and provide better performance for the transform request types.
- Stateless, [RFC5734] requires a stateful session between a client and the EPP server. A REPP server MUST be stateless and MUST NOT keep client session or any other application state. Every client request needs to provide all of 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.
- Content negotiation, A server may choose to include support for multiple media types. The client must be able to signal the server what media type the should use for decoding request content en for encoding 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 commands and corresponding request and response messages.

- Simplicity, when the semantics of a resource URL and HTTP method match an EPP command and request message, the use of an request message should ne optional. If the EPP response message is limited to the EPP result code and transaction identifiers, sending a response message should be optional.
- Performance, reducing the number of required request and response messages, improves the performnce of both client and server, less messages have to be created, marshalled, transmitted and parsed.

5. EPP Extension Framework

Section 2 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: REPP does not define any new high level protocol elements. The Section 9 section describes an extension resource for use with existing and future command extensions.
- Object extension: REPP does not use the "command" concept, because the "command" concept is part of a RPC style and not of the REST style. A REST URL resource and HTTP method combination have replaced the command concept. The Section 9 section describes a command extension resource for each object type and can be used for existing and future command extensions. REPP does not define any new object level extensions. All existing and future object level EPP extensions can be used.
- Command-Response extension: RESTful EPP reuses the existing request and response messages defined in the EPP RFCs.

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 by each registry. The {context-root} MAY be an empty, zero length string.
- {version} is a path segment which identifies the interface version. This is the equivalent of the Version element in the EPP RFCs. The version used in a REPP URL MUST match the version used in EPP request and response messages.
- {collection} MUST be substituted by "domains", "hosts" or "contacts", 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 of the object instance.

Reource URLs used by REPP may contain embedded object identifiers. By using a 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 trade off. Removing the object identifier from the request message would require new object mapping schemas.

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

Session management as described in [RFC5730] requires a stateful server, maintaining client and application state. One of the main design considerations of REPP is to enable more scalable EPP services, for this the REPP server MUST use a stateless architecture. Session management functionality MUST be delegated to the HTTP layer.

The server MUST not create and maintain client sessions for use over multiple client requests and NOT maintain any state information relating to the client or EPP process.

Due to stateless nature of REPP, a 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].

8. HTTP

REPP uses the REST semantics, each HTTP method is assigned a distinct behaviour, section Section 8.1 provides a overview of each the behaviour assinged to each method. REPP requests are expressed by using a URL refering to a resource, an HTTP method, zero or more HTTP headers and a optional message body containing the EPP request message.

An 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.

When using an HTTP version where the TCP connection is not reused, 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 a object resource or a collection of resources
- PUT: 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 REPP server MAY choose to support multiple representations for EPP objects, such as XML and JSON. When multiple representations are supported, the server MUST use agent-driven content negotiation and HTTP headers for content negotiation, as described in Section 12.2 of [RFC2616].

The client MUST use these HTTP headers:

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

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. EPP content

In contrast to EPP over TCP [RFC5734], REPP does not always require a EPP request message to be sent to the server. The information conveyed by HTTP method, URL and request headers is, for some use cases, sufficient for the server to be able to successfully proceses the request. The `Object Info` request for example, does not require an EPP message.

When an EPP request does require an EPP request message, the client MUST use the HTTP POST or PUT method and add the EPP request message content to the HTTP message body.

8.4. Request

HTTP request-headers are used to transmit additional or optional request data to the server. All REPP HTTP headers must have 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 REPP request does not contain an EPP request in the HTTP message body.
- `REPP-svcs`: The namespace used by the client in the EPP request message. The client MUST use this header if the media type used by the client for the message body content requires the server to know what namespaces are used. Such is the case for XML-based request messages. The header value MAY contain multiple comma separated namespaces.

- REPP-authInfo: The client MAY use this header for sending basic password-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 NOT use this header.
- Accept-Language: This header is equivalent to the "lang" element in 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 langauge, the server MUST respond using the default language configured for the server, as required in Section 2.9.1.1 of [RFC5730]
- Connection:

8.5. Response

The server response contains an HTTP status code, HTTP headers and MAY contain an EPP response message in the HTTP message body. HTTP response-headers are used to transmit additional response data to the client. All HTTP headers used by REPP MUST use the "REPP-" prefix.

- 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 in [RFC5730] and MUST be used accordingly. If an HTTP message body with the EPP XML equivalent <clTRID> exists, both values MUST be consistent.
- REPP-eppcode: This header is the equivalent of the result code defined in [RFC5730] and MUST be used accordingly. This header MUST only be used when an REPP response HTTP message body has no content.
- 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.
- Cache-Control: ... TBD: the idea is to prohibit caching. Even though it will probably work and be useful in some scenario's, it also complicates matters.
- Connection: ....

8.6. Error Handling

Restful EPP is designed atop of the HTTP protocol, both are an application layer protocol with their own status- and result codes. The endpoints described in Section 9 MUST return the specified HTTP status 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 a semantically equivalent HTTP status code. Table 1 contains the mapping for EPP result codes to HTTP status codes.

The client MAY use the well defined HTTP status codes for error handling logic, without having to first parse the EPP result mesage.

For example, a client sending an Object Tranfer request when the Object is already linked to an active transfer process, this will cause the server to respond using an EPP result code 2106 this code maps to HTTP status code 400. 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 is required for handling the error.

EPP result code	HTTP status code
2000	501
2001	400
2002	405
2003	400
2004	400
2005	400
2100	400
2101	501
2102	
2103	
2104	
2105	
2106	400
2201	
2202	
2300	
2301	
2302	
2303	
2304	
2305	

EPP result code	HTTP status code
2306	
2307	
2308	
2400	500
2500	500
2501	401
2502	429

Table 1: EPP code to HTTP code mapping

TODO: complete the table

- `EPP result code`: MUST only return EPP result information relating to the EPP protocol. The HTTP header "REPP-eppcode" MAY be used to add EPP result information to the HTTP layer.
- `HTTP status code`: MUST only return status information related to the HTTP protocol

9. Command Mapping

EPP commands are mapped to RESTful EPP transaction consisting out of three elements.

1. A resource defined by a URL.
2. The HTTP method to execute on the resource.
3. The EPP request message, contained in the HTTP message body.

For some EPP transactions a request message is optional or not supported.

Table 2 lists a mapping for each EPP command to REPP transaction, the subsequent sections provide details for each request. Resource URLs in the table are assumed to be using the prefix: "/{context-root}/{version}/".

- `{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.

Command	Method	Resource	Req message	Resp message
Hello	OPTIONS	/	No	TODO
Login	N/A	N/A	N/A	TODO

Command	Method	Resource	Req message	Resp message
Logout	N/A	N/A	N/A	TODO
Check	HEAD	/{c}/{i}	No	TODO
Info	GET/ POST	/{c}/{i}	Optional	TODO
Poll Request	GET	/messages	No	TODO
Poll Ack	DELETE	/messages/{i}	No	TODO
Create	POST	/{c}	Yes	TODO
Delete	DELETE	/{c}/{i}	No	TODO
Renew	POST	/{c}/{i}/renewals	Yes	TODO
Transfer	POST	/{c}/{i}/transfers	Optional	TODO
Transfer Query	GET/ POST	/{c}/{i}/transfers/ latest	Optional	TODO
Transfer Cancel	DELETE	/{c}/{i}/transfers/ latest	Optional	TODO
Transfer Approve	PUT	/{c}/{i}/transfers/ latest	Optional	TODO
Transfer Reject	DELETE	/{c}/{i}/transfers/ latest	Optional	TODO
Update	PUT	/{c}/{i}	Yes	TODO
Extension [1]	*	/{c}/{i}/extension/*	*	TODO
Extension [2]	*	/extension/*	*	TODO

Table 2: Mapping of EPP Command to REPP Request

[1] This mapping is used for Object extensions based on the extension mechanism as defined in [RFC5730, secion 2.7.2]

[2] This mapping is used for protocol extensions based on the extension mechanism as defined in [RFC5730, secion 2.7.1]

When there is a mismatch between the resource identifier in the HTTP message body and the resource identifier in the URL used for a request, then the servr MUST return HTTP status code 400 (Bad Request).

9.1. Hello

- Request: OPTIONS /{context-root}/{version}
- Request payload: No
- Response payload: Greeting response
- HTTP success status code: 200 (OK)

The server MUST return a Greeting response, as defined in Section 2.4 of [RFC5730] in response to request using the HTTP OPTIONS method on the root "/" resource.

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

Example Hello 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: REPP-cltrid: ABC-12345
C: Connection: keep-alive
```

Example Hello response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 799
S: Content-Type: application/epp+xml
S:
S: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
S: <epp xmlns="urn:ietf:params:xml:ns:epp-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: </epp>
```

9.2. Login

The Login command defined in [RFC5730] is used to configure a session and is part of the stateful nature of the EPP protocol. A REPP server is stateless and MUST not maintain any client state and MUST NOT support the Login command. The client MUST include all the

information in a REPP request that is required for the server to be able to properly process the request, this includes the request attributes that are part of the Login command defined in Section 2.9.1.1 of [RFC5730].

The request attributes from the [RFC5730] Login command are are moved to the HTTP layer.

- clID: Replaced by HTTP authentication
- pw:: Replaced by HTTP authentication
- newPW: Replaced by HTTP authentication
- version: Replaced by the {version} path segment in the request URL.
- lang: Replaced by the Accept-Language HTTP header.
- svcs: Replaced by the REPP-svcs 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

The concept of a session no longer exists when using REPP, therefore the Logout command MUST not be implemented by the server.

9.4. Query Endpoints

Sending content using an HTTP GET request is discouraged in [RFC9110], there exists no generally defined semanticsfor content received in a GET request.

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. 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: None
- Response message: None
- HTTP success status code: 200 (OK)

The server MUST support the HTTP HEAD method for the Check endoint, both client and server MUST not put any content to the HTTP message body. The response MUST contain the REPP-check-avail 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 REPP Check endpoint is limited to checking only a single resource {id} per request. This may seem a step backwards compared to the Check command defined in the [RFC5730] where multiple object-ids are allowed inside a Check command. The RESTful Check request can be load balanced more efficiently when a single resource {id} needs to be checked.

Example Check 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 Check response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 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
```

9.4.2. Info

An Info request MUST be performed using the HTTP GET or POST method on a resource identifying an object instance. The response MUST be a response message as described in object mapping of the EPP RFCs.

An object MAY have authorization attachted to it, forcing the client to include the authorization in the request. When the authorization needs to be included in the request the HTTP POST method MUST be used.

A request for an object without authorization information.

- Request: GET /{collection}/{id}
- Request message: None
- Response message: Info response
- HTTP success status code: 200 (OK)

A request for an object that has authorization information attached.

- Request: POST /{collection}/{id}
- Request message: Info request
- Response message: Info response
- HTTP success status code: 200 (OK)

9.4.2.1. Object Filtering

The client MAY choose to use a filter to limit the number of objects returned for a request. The server MUST support the use of query string parameters for the prupose of filtering objects before these are added to a response.

Query string parameters used for filtering:

- attr: The name of the object attribute or field to filter on
- val: The value the server must use when filtering objects

The domain name Info request is different from the Contact- and Host Info request, in the sense that EPP Domain Name Mapping Section 3.1.2 describes an OPTIONAL "hosts" attribute. This attribute is used for filtering hosts returned in the response, the "hosts" attribute is mapped to the generic query string filtering.

The specified default value for the hosts parameter is "all". This default MUST be used by the server when the query string parameter is absent from the request URL.

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

Example Domain Info request including all hosts objects, without any required authorization data:

```
C: GET /repp/v1/domains/example.nl?attr=hosts&val=all 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 Info response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 424
S: Content-Type: application/epp+xml

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

9.4.3. Poll

9.4.3.1. Poll Request

- Request: GET /messages
- Request message: None
- Response message: Poll response
- HTTP success status code: 200 (OK)

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 Poll 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 Poll response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 312
S: Content-Type: application/epp+xml

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

9.4.3.2. Poll Ack

- Request: DELETE /messages/{id}
- Request message: None
- Response message: Poll ack response
- HTTP success status code: 200 (OK)

The client MUST use the HTTP DELETE method on a message instance to to acknowledge receipt of a message of a message from the message queue. The "op=ack" semantics from Section 2.9.2.3 are assigned to the HTTP DELETE method. The "msgID" from a received EPP message MUST be included in the message resource URL, using the {id} path element.

Example Poll Ack 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 Poll Ack response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 312
S: Content-Type: application/epp+xml

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <msgQ count="4" id="12345"/>
S:    <trID>
S:      <clTRID>ABC-12346</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

9.4.4. Transfer Query

The Transfer Query request MUST use the special "latest" 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 NOT add content to the HTTP message body when using the HTTP GET method.

- Request: GET {collection}/{id}/transfers/latest
- Request message: None
- Response message: Transfer Query response
- HTTP success status code: 200 (OK)

If the requested object has associated authorization information linked to a contact object, then the HTTP GET method MUST NOT be used and the HTTP POST method MUST be used and the authorization information MUST be included in the EPP request message inside the HTTP message body.

- Request: POST {collection}/{id}/transfers/latest
- Request message: Transfer Query request
- Response message: Transfer Query response.
- HTTP success status code: 200 (OK)

Example domain name Transfer Query request:

```
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
```

Example domain name Transfer Query request and authorization information in 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-svcs: urn:ietf:params:xml:ns:domain-1.0
C: REPP-authInfo: secret
```

Example domain name Transfer Query request and authorization information in request message:

```
C: POST /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:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <transfer op="query">
C:      <domain:transfer
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:        <domain:authInfo>
C:          <domain:pw roid="MW12345-REP">secret</domain:pw>
C:        </domain:authInfo>
C:      </domain:transfer>
C:    </transfer>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Transfer Query response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 230
S: Content-Type: application/epp+xml
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

9.5. Transform Endpoints

9.5.1. Create

- Request: POST /{collection}
- Request message: Object Create request
- Response message: Object Create response
- HTTP success status code: 201 (CREATED)

The client MUST use the HTTP POST method to create a new object 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:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <create>
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:    </create>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Domain Create response:

```
S: HTTP/2 201 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme 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:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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>54321-XYZ</svTRID>
S:      </trID>
S:   </response>
S:</epp>
```

9.5.2. Delete

- Request: DELETE /{collection}/{id}
- Request message: None
- Response message: object Delete response
- HTTP success status code: 200 (OK)

The client MUST the HTTP DELETE method and a resource identifying a unique object instance.

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: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 505
S: Content-Type: application/epp+xml

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>XYZ-12345</svTRID>
S:      </trID>
S:   </response>
S:</epp>
```

9.5.3. Renew

- Request: POST /{collection}/{id}/renewals
- Request message: object Renew request
- Response message: object Renew response
- HTTP success status code: 201 (CREATED)

The EPP Renew command is mapped to a nested resource, named "renewals". Not all EPP object types include support for the renew command, the server MUST return HTTP status code 404 (Not Found) when the client send a Renew request for an unsupported object type.

Example Domain Renew request:

```
C: POST /repp/v1/domains/example.nl/renewals 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: 325
C:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <renew>
C:      <domain:renew
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:        <domain:curExpDate>2023-11-17</domain:curExpDate>
C:        <domain:period unit="y">1</domain:period>
C:      </domain:renew>
C:    </renew>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Renew response:

```
S: HTTP/2 201 CREATED
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 505
S: Content-Type: application/epp+xml
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

9.5.4. Transfer

Transferring an object from one sponsoring client to another 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 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. Create

- Request: POST /{collection}/{id}/transfers
- Request payload: Optional Transfer request
- Response message: Transfer response.
- HTTP success status code: 201 (CREATED)

To start a new object transfer process, the client MUST use the HTTP POST method for a unique resource, not all EPP include support for 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 server does not require authorization information associated with a contact object, then the client MAY choose to send an empty HTTP message body.

Example Create request wiithout 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: Accept-Language: en
C: REPP-cltrid: ABC-12345
```

Example Create request using object authorization not linked to a contact:

```
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-cltrid: ABC-12345
C: REPP-authInfo: secret
C: Accept-Language: en
```

Example Create request using object authorization linked to a contact object:

```
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: Content-Length: 252

C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <transfer op="request">
C:      <domain:transfer
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:        <domain:authInfo>
C:          <domain:pw roid="DOM-12345">secret</domain:pw>
C:        </domain:authInfo>
C:      </domain:transfer>
C:    </transfer>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Transfer response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 328
S: Content-Type: application/epp+xml
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

9.5.4.2. Cancel

- Request: DELETE /{collection}/{id}/transfers/latest
- Request message: Optional Transfer Reject request
- Response message: Transfer cancel response message.
- HTTP success status code: 200 (OK)

The semantics of the HTTP DELETE method are determined by the role of the client sending the request. For the new sponsoring client the DELETE method is defined as "cancel transfer".

The new sponsoring client MUST use the HTTP DELETE method to cancel a requested transfer.

Example Cancel 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 Cancel response:

```
TODO
```

9.5.4.3. Reject

- Request: DELETE /{collection}/{id}/transfers/latest
- Request message: Optional Transfer Reject request
- Response message: Transfer response
- HTTP success status code: 200 (OK)

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".

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

Example Reject 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:

```
TODO
```

9.5.4.4. Approve

- Request: PUT /{collection}/{id}/transfers/latest
- Request message: Optional Transfer Approve request
- Response message: Transfer response.
- HTTP success status code: 200 (OK)

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
```

Example Approve response:

```
TODO
```

9.5.5. Update

- Request: PUT /{collection}/{id}
- Request message: Object:update.
- Response message: Update response message
- HTTP success status code: 200 (OK)

An object Update request MUST be performed with the HTTP PUT method on a unique object resource. The payload MUST contain an Update request as described in the EPP RFCs.

Example Update request:

```
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: 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:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <update>
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 response is omitted here -->
C:      </domain:update>
C:    </update>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Update response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 328
S: Content-Type: application/epp+xml

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

9.6. Extensions

- Request: * /extensions/*
- Request message: *
- Response message: *
- HTTP success status code: *

EPP protocol extensions, as defined in secion 2.7.3 are supported using the generic "/extensions" resource. The HTTP method used for a extension is not defined but must follow the RESTful principles.

Example Extension request: The example below, shows the use of the "Domain Cancel Delete" command as defined as a custom command in [SIDN-EXT] by the .nl domain registry operator. Where the registrar can use the HTPP DELETE method on a domain name resource to cancel an active domain delete transaction and move the domain from the quarantine state back to the active state.

```
C: DELETE /repp/v1/extensions/domains/example.nl/quarantine 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-cltrid: ABC-12345
```

Example Extension response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 328
S: Content-Type: application/epp+xml

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-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:</epp>
```

10. Transport Considerations

Section 2.1 of [RFC5730] of the EPP protocol specification describes considerations to be addressed by a protocol transport mapping. This section addresses each of the considerations using a combination of REPP features and features provided by HTTP as follows:

- When using load balancing to distribute requests over multiple stateless REPP servers the return order of the results cannot be guaranteed. Therefore the client is responsible for sending results in the correct order, and may have to wait for a server response for a previous request, if a request depends on the response of a previous request.
- Sessions are delegated to the HTTP layer, which uses the client-server paradigm. HTTP is an application layer protocol which uses TCP as a transport protocol. TCP 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 is a stateless protocol and as such it does not maintain any client state.
- The stateful nature of EPP is no longer preserved through EPP managed sessions. Session management is delegated to the stateless HTTP layer. EPP session related information, such as authentication credentials MUST be included in every HTTP request. This is required for the server to be able to process the request successfully.
- HTTP 1.1 allows persistent connections which can be used to send multiple HTTP requests to the server using the same connection.
- The server MAY allow pipelining, [RFC9000] descibes a mechanism for multiplexing multiple request streams.
- Batch-oriented processing (combining multiple EPP commands in a single HTTP request) MUST NOT be permitted. To maximize scalability every request MUST contain oly a single command.

- A request processing failure has no influence on the processing of other requests. The stateless nature of the server allows a client to retry a failed request by re-sending the request.
- Due to the stateless nature of a REPP service, errors while processing a EPP command or other errors are isolated to a single request. The Error status MUST be communicated to the client using the appropriate HTTP status codes.

11. IANA Considerations

TODO: any?

12. Internationalization Considerations

TODO: any? Accept-Language in HTTP Header

13. Security Considerations

[RFC5730] describes a Login command for transmitting client credentials. This command MUST NOT be used for REPP. Due to the stateless nature of REPP, the client MUST include the authentication credentials in each HTTP request. The validation of the user credentials must be performed by an out-of-band mechanism. Examples of authentication mechanisms are Basic and Digest access authentication [RFC2617] or OAuth [RFC5849].

To protect data confidentiality and integrity, all data transport between the client and server MUST use TLS [RFC5246]. Section 9 describes the level of security that is REQUIRED.

EPP does not use XML encryption for protecting messages. Furthermore, REPP (HTTP) servers are vulnerable to common denial-of-service attacks. Therefore, the security considerations of [RFC5734] also apply to REPP.

14. Obsolete EPP Result Codes

TODO: check list of RFC5730 codes and see which ones are not used anymore.

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

Code	Reason
1500	The logout command is not used anymore.
2100	The REPP URL already includes the version.
2002	Commands can now be sent in any order. TODO: is order guaranteed?
2200	The login command is not used anymore.

Table 3: Obsolete EPP result codes

15. Acknowledgments

TODO Move Miek from Authors to Acknowledgments section?

16. References

16.1. 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>.

[RFC5849] Hammer-Lahav, E., Ed., "The OAuth 1.0 Protocol", RFC 5849, DOI 10.17487/RFC5849, April 2010, <https://www.rfc-editor.org/info/rfc5849>.

[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>.

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

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[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>.

16.2. 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/