

---

Workgroup:	Network Working Group
Internet-Draft:	draft-wullink-restful-epp-01
Published:	22 November 2023
Intended	Standards Track
Status:	25 May 2024
Expires:	M. Wullink     M. Davids
Authors:	<i>SIDN Labs     SIDN Labs</i>

# Extensible Provisioning Protocol (EPP) RESTful Transport

---

## Abstract

This document specifies a 'RESTful transport for EPP' (REPP) with the aim of improving the efficiency and interoperability of EPP.

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

REPP allows for a stateless server implementation, no session data is maintained on the EPP server, this allows for a better scalable EPP service.

## 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 25 May 2024.

## Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and

restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction	4
2. Terminology	4
3. Conventions Used in This Document	5
4. Design Considerations	5
5. EPP Extension Framework	6
6. Resource Naming Convention	6
7. HTTP Usage	7
7.1. Method Definition	7
7.2. Media types	7
7.3. Request	8
7.3.1. EPP content	8
7.3.2. REPP Request Headers	8
7.3.3. Generic HTTP Headers	8
7.4. Response	8
7.4.1. REPP Response Headers	9
7.4.2. Generic Headers	9
7.5. Response Status Codes	9
8. Command Mapping	10
8.1. Hello	11
8.2. Session Management	12
8.2.1. Login	12
8.2.2. Logout	12
8.3. Query	13
8.3.1. Check	13
8.3.2. Info	14
8.3.2.1. Domain Name	15

---

8.3.3. Poll	16
8.3.3.1. Poll Request	16
8.3.3.2. Poll Ack	17
8.3.4. Transfer Query	18
8.4. Transform	19
8.4.1. Create	19
8.4.2. Delete	21
8.4.3. Renew	21
8.4.4. Transfer	23
8.4.4.1. Create	23
8.4.4.2. Cancel	25
8.4.4.3. Reject	25
8.4.4.4. Approve	26
8.4.5. Update	26
8.5. Extensions	27
9. Transport Considerations	29
10. IANA Considerations	29
11. Internationalization Considerations	29
12. Security Considerations	29
13. Obsolete EPP Result Codes	30
14. Acknowledgments	30
15. References	30
15.1. Normative References	30
15.2. Informative References	32
Authors' Addresses	32

## 1. Introduction

This document describes a transport protocol for EPP, based on the [\[REST\]](#) architectural style. This transport mechanism 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 [\[URI\]](#) resources. 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 dictates that no session state is maintained on the EPP server. Each request from client to server will contain all of the information necessary to understand the request. The server will close the connection after each HTTP request.

With a stateless mechanism, some drawbacks of EPP (as mentioned in Section 5) are circumvented.

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.

REPP client - An HTTP user agent performing an REPP request

REPP server - An HTTP server responsible for processing requests and returning a result 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 this 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 describes 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.

The main benefit of using HTTP is the widespread availability of knowledge about developing and deploying HTTP based services.

- 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 TCP sessions. A TCP session may be used to transmit multiple request messages and these are processed by a single server and not load balanced across a pool of servers.
  - 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 available authentication and authorization solutions for HTTP based APIs.
- Support for multiple resource representation media types and provide a mechanism for the client to signal the server what media type the server should use. This document only describes the use of [XML] but use of response media types such as JSON [[RFC7159](#)] should also be possible.

- Compatibility with existing EPP RFCs.
- Optional use of EPP request messages when the semantics of a resource URL and HTTP method match the contents of an EPP request.

## 5. EPP Extension Framework

EPP provides an extension framework, [Section 2](#), describes how to extend EPP by adding new features at the protocol, object and command-response levels. REPP affects the following levels:

Protocol Extension: REPP does not use the "command" concept, because the "command" concept is part of a RPC style and not of a RESTful style. A REST URL and HTTP method combination have replaced the command structure. The [Section 8](#) section describes an extension resource for use with existing and future command extensions.

Object extension: REPP does not define any new object level extensions. Any existing 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 resource can be a single uniquely object identifier e.g. a domain name, or a collection of objects. The complete set of objects available to client 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 label 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 by EPP in the upper layer.
- {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 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 example.nl, would look like this:

/repp/v1/domains/example.nl

The level below a collection MUST be used to identify an object instance, the level below an object instance MUST be used to identify attributes of the object instance.

With REPP the object identifiers are embedded in URLs. This makes any object identifier in the request messages superfluous. However, since the goal of REPP is to stay compatible with the 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 HTTP message-body does not match the {id} object identifier in the URL.

## 7. HTTP Usage

A [\[RFC5730\]](#) EPP request includes a command- and object mapping to which a command must be applied. REPP uses the REST semantics each HTTP method is assigned a distinct behaviour, section [Section 7.1](#) provides a overview of each the behaviour assigned to each method. REPP requests are expressed by using an URL resource, an HTTP method and optionally an HTTP header and message body.

Data (payload) belonging to a request or response is added to the HTTP message- body or sent as using an HTTP header, depending on the nature of the request as defined in Section 9.

An HTTP request **MUST** contain no more than one EPP command. HTTP requests **MUST** be processed independently of each other and in the same order as the server receives them.

A client **MAY** use the "Connection" header to request for the server to not close the existing connection, so it can be used for future requests. The server **MAY** choose not to honor this request.

### 7.1. 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 "verbs" ([\[REST\]](#)).

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

### 7.2. Media types

TODO Describe how REP **MUST** be data format agnostic and support multiple data formats. XML is already defined, but future JSON mappings **MUST** also be supported.

now using http header: Accept: application/epp+xml **MUST** support other content-types e.g.: application/epp+json

## 7.3. Request

TODO: add text that due to stateless nature a request must contain all data required to execute command. includes data previously part of client session (login)

### 7.3.1. 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 encoded in the URL and request headers is sufficient for the server to be able to successfully process some request, such the Object Info request.

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

### 7.3.2. REPP Request Headers

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 from [[RFC6648](#)].

REPP-cltrid: The client transaction identifier is the equivalent of the <clTRID> element in the EPP RFCs and **MUST** be used accordingly. When this header is present in a client request, an equivalent element in the message-body **MAY** also be present, but **MUST** than be consistent with the header.

REPP-svcs: The namespace used by the client in the EPP request document in the message-body of the HTTP request.

### 7.3.3. Generic HTTP Headers

Generic HTTP headers as defined in HTTP/1.1 [[RFC2616](#)], the following general-headers are **REQUIRED** in REPP HTTP requests.

Accept-Language: This request-header is equivalent to the "lang" element in the EPP Login command. The server **MUST** support the use of HTTP Accept-Language header in client requests. 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 does not have to match any of the languages from Hello response. When the server receives a request using an unsupported language, the server **MUST** respond using the default language configured for the server, this differs from the server behaviour described in [Section 2.9.1.1](#) of [[RFC5730](#)].

TODO: Connection: ...

## 7.4. Response

The server response contains an HTTP Status-Code, HTTP response-headers and it **MAY** contain an EPP response message in the HTTP message-body.



#### 7.4.1. REPP Response Headers

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 in the EPP RFCs and MUST be used accordingly. 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 the EPP RFCs 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> element in the EPP RFCs and MUST be used accordingly. If an HTTP message-body with The EPP XML equivalent <result code> exists, both values MUST be consistent.

REPP-check-avail: An alternative for the "avail" attribute of the [object:name](#) element in a check response and MUST be used accordingly.

REPP-check-reason: An optional alternative for the "object:reason" element in a check response and MUST be used accordingly.

#### 7.4.2. Generic Headers

Generic HTTP headers MAY be used as defined in HTTP/1.1 [[RFC2616](#)]. For REPP, the following general-headers are REQUIRED in HTTP responses.

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: ....

### 7.5. Response Status Codes

TODO: see for example: <https://datatracker.ietf.org/doc/html/rfc7480>

#### ## Error Handling

REPP is designed atop of the HTTP protocol, both are an application layer protocol with their own status- and result codes. The value of an EPP result code and HTTP Status-Code MUST remain independent of each other. E.g. an EPP result code indicating an error can be combined with an HTTP request with Status-Code 200 when the EPP command result contains an error, but the HTTP request was successful.

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, When there is a mismatch between the object identifier in the HTTP message-body and the resource URL HTTP Status-Code 412 MUST be returned.

## 8. Command Mapping

REPP conforms to the EPP transport mapping considerations as defined in [Section 2.1](#) of [\[RFC5730\]](#). The EPP [\[RFC5730\]](#) commands are mapped to RESTful operation.

Each RESTful operation consists of four parts: 1. the resource, 2. the HTTP method 3. the request payload, which is the HTTP message- body of the request, 4. the response payload, being the HTTP message- body of the response.

Table 2 list a mapping for each EPP to REPP, the subsequent sections provide details for each request. All resource URLs in the table are assumed to use the prefix: `"/{context-root}/{version}/"`.

`{c}`: An abbreviation for `{collection}`: this MUST be substituted with "domains", "hosts", "contacts" or "messages".

`{i}`: An abbreviation for `{id}`: this MUST be substituted with the value of a domain name, hostname, contact-id or a message-id.

TODO: add resource extension commands for such as nl cancel-delete

Command	Method	Resource
Hello	OPTIONS	/
Login	N/A	N/A
Logout	N/A	N/A
Check	HEAD	/ {c}/ {i}
Info	GET/POST	/ {c}/ {i}
Poll Request	GET	/messages
Poll Ack	DELETE	/messages/{i}
Create	POST	/ {c}
Delete	DELETE	/ {c}/ {i}
Renew	PUT	/ {c}/ {i}/period
Transfer	POST	/ {c}/ {i}/transfers
Transfer Query	GET/POST	/ {c}/ {i}/transfers/latest
Transfer Cancel	DELETE	/ {c}/ {i}/transfers/latest
Transfer Approve	PUT	/ {c}/ {i}/transfers/latest

Command	Method	Resource
Transfer Reject	DELETE	/ {c} / {i} / transfers / latest
Update	PUT	/ {c} / {i}
Extensions [1]	*	/ extensions / *

Table 1: Mapping of EPP Command to REPP Request

[1] This mapping is used for protocol extensions based on the extension mechanism as defined in [RFC5730, section 2.7.3]

## 8.1. Hello

- Request: OPTIONS / {context-root} / {version}
- Request payload: N/A
- Response payload: Greeting response

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

The version information returned by the server in the Hello response MUST match the version used in the URL of the REPP server.

Example Hello request:

```
C: OPTIONS /repp/v1/ HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: Connection: keep-alive
```

Example Hello response:

```
S: HTTP/1.1 200 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:     <!-- The rest of the response is omitted here -->
S:   </greeting>
S: </epp>
```

## 8.2. Session Management

One of the design goals of REPP is to increase the scalability of an EPP service. This means that session management as described in [RFC5730](#) is not supported, session management functions are delegated to the HTTP layer.

The server **MUST** not create a client session for use across multiple client requests. The server **MUST** not maintain any state information relating to the client or EPP process state. The client **MUST** include authentication credentials with each request. This **MAY** be done by using any of the available HTTP authentication mechanisms, such as those described in [RFC2617](#).

### 8.2.1. 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.

Data attributes from the [RFC5730](#) Login command are either no longer used or are moved to the HTTP layer using a HTTP-header.

- cIID: No longer used
- pw: No longer used
- newPW: No longer used
- version: This attributed has been replaced by the version in the server URL.
- lang: This attribute has been replaced by the Accept-Language HTTP request-header.
- svcs: This attribute has been replaced by 1 or more REPP-svcs HTTP request-headers.

The server **MUST** check the namespace used in all REPP-svcs HTTP request-header. An unsupported namespace **MUST** result in the appropriate EPP result code.

### 8.2.2. Logout

The concept of a session is no longer sued by REPP, therefore the Logout command **MUST** not be implemented by the server.

### 8.3. Query

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

#### 8.3.1. Check

- Request: HEAD /{collection}/{id}
- Request payload: N/A
- Response payload: N/A

The server MUST support the HTTP HEAD method for the Check command, the client and the server MUST not add 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 "1" or "0" as described in the EPP RFCs, depending on whether the object can be provisioned or not.

A Check request using the HTTP HEAD method is limited to checking only a single resource {id}. This may seem a step backwards when compared to the Check command defined in the EPP RFCs where multiple object-ids are allowed inside a Check command. The RESTful Check command can be load balanced more efficiently when the request contains only a single resource {id} that needs to be checked.

<!-- do we also need GET method for check? this old text described addign GET method but we already use this for Info command, so need other resource for check?

the server MAY also support the HTTP GET method. If the HTTP HEAD method is used, the client and the server MUST not add any content to the HTTP message-body. If the HTTP GET method is used the client and the server MUST add the Check content to the message-body. The HTTP response for a request using the HTTP GET method, MUST contain the REPP-check-avail and MAY contain the REPP-check-reason header. The value of REPP-check-avail header MUST be "1" or "0" as described in the EPP RFCs, depending on whether the object can be provisioned or not.

A Check request using the HTTP HEAD method is limited to checking only a single resource {id}. This may seem a step backwards when compared to the Check command defined in the EPP RFCs where multiple object-ids are allowed inside a Check command. The RESTful Check command can be load balanced more efficiently when the request contains only a single resource {id} that needs to be checked. When the HTTP GET method is used, the EPP request in the message-body MUST also be limited to a single object to check. The server MUST return EPP result code 2002, when the Check request contains more than 1 object to check.

Request with a request message:

- Request: GET /{collection}/{id}
- Request payload: Check request
- Response payload: Check Response -->

Example Check request for a domain name:

```
C: HEAD /repp/v1/domains/example.nl HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example Check response:

```
S: HTTP/1.1 200 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
```

### 8.3.2. 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 attached 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 payload: N/A
- Response payload: Info response

A request for an object that has authorization information attached.

- Request: POST /{collection}/{id}

- Request payload: Info request
- Response payload: Info response

#### 8.3.2.1. Domain Name

A domain name Info request is different from a contact- and host Info request in the sense that EPP Domain Name Mapping [[RFC5731](#)], Section 3.1.2 describes an OPTIONAL "hosts" attribute. This attribute is mapped to a nested resource of the domains collection.

The specified default value is "all". This default is mapped to a shortcut, the resource object instance URL without any additional labels.

- default: GET /domains/{id}
- Hosts=all: GET /domains/{id}/hosts/all
- Hosts=del: GET /domains/{id}/hosts/del
- Hosts=sub: GET /domains/{id}/hosts/sub
- Hosts=none: GET /domains/{id}/hosts/none

Example domain Info including all hosts, without authorization data:

```
C: GET /repp/v1/domains/example.nl/hosts/all HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example Info response:

```
S: HTTP/1.1 200 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>
```

### 8.3.3. Poll

#### 8.3.3.1. Poll Request

- Request: GET /messages
- Request payload: N/A
- Response payload: Poll response

A client MUST use the HTTP GET method on the messages collection to request the message at the head of the queue.

Example Poll request:

```
C: GET /repp/v1/messages HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
```

Example Poll response:



```
S: HTTP/1.1 200 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>
```

### 8.3.3.2. Poll Ack

- Request: DELETE /messages/{id}
- Request payload: N/A
- Response payload: Poll ack response

A client MUST use the HTTP DELETE method on a message instance to acknowledge the removal of the message from the message queue.

Example Poll Ack request:

```
C: GET /repp/v1/messages/12345 HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
```

Example Poll Ack response:

```
S: HTTP/1.1 200 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>
```

#### 8.3.4. Transfer Query

The Transfer Query request uses the special "latest" resource to refer to the latest active object transfer.

- Request: GET {collection}/{id}/transfers/latest
- Request payload: N/A
- Response payload: Transfer respons.

If the requested object has associated authorization information then the HTTP GET method MAY be used, otherwise the HTTP POST method MUST be used.

- Request: POST {collection}/{id}/transfers/latest
- Request payload: Transfer Query request
- Response payload: Transfer Query response.

Example domain name Transfer Query request:

```
C: GET /repp/v1/domains/example.nl/transfers/latest HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example Transfer Query response:

```
S: HTTP/1.1 200 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>
```

## 8.4. Transform

### 8.4.1. Create

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

A client MUST create a new object using the HTTP POST method on an object collection resource.

Example Domain Create request:

```
C: POST /repp/v1/domains/ HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
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/1.1 200 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: 642
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:   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>
```

#### 8.4.2. Delete

- Request: DELETE /{collection}/{id}
- Request payload: N/A
- Response payload: Object Delete response

Deleting an object from the registry database **MUST** be performed using the HTTP DELETE method on a REST resource identifying a unique object instance.

Example Domain Delete request:

```
C: DELETE /repp/v1/domains/example.nl HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
```

Example Domain Delete response:

```
S: HTTP/1.1 200 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>
```

#### 8.4.3. Renew

- Request: PUT /{collection}/{id}/period
- Request payload: Object <renew>.
- Response payload: Object <renew> response.

Renewing an object is only specified by [\[RFC5731\]](#), the <renew> command has been mapped to a period resource.

Example Renew request:

```
C: POST /repp/v1/domains/example.nl/period HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
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:     </command>
C:   </epp>
```

Example Renew response:

```
S: HTTP/1.1 200 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:
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>
```

#### 8.4.4. Transfer

Transferring an object from one sponsoring client to another is only specified in [\[RFC5731\]](#) and [\[RFC5733\]](#). The <transfer> command has been mapped to a transfer resource.

The semantics of the HTTP DELETE method are determined by the role of the client executing the method. For the current sponsoring registrar the DELETE method is defined as "reject transfer". For the new sponsoring registrar the DELETE method is defined as "cancel transfer".

##### 8.4.4.1. Create

- Request: POST /{collection}/{id}/transfers
- Request payload: Optional Transfer Approve request
- Response Payload: Transfer response.

To start a new object transfer, the client **MUST** use the HTTP POST method on a unique domain name or contact object instance. If the server only requires the domain name to be able to create a new transfer, then the client **MAY** choose to send an empty HTTP message-body. [Section 3.2.4](#) of [\[RFC5730\]](#) described additional information the server might require.

Example Create request using no object authorization:

```
C: POST /repp/v1/domains/example.nl/transfers HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

Example Create request using object authorization:

```
C: POST /repp/v1/domains/example.nl/transfers HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
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:     <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">kds78jhbfdsk</domain:pw>
C:           </domain:authInfo>
C:         </domain:transfer>
C:       </transfer>
C:     <clTRID>ABC-12345</clTRID>
C:   </command>
C: </epp>
```

Example Transfer response:



```
S: HTTP/1.1 200 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>
```

#### 8.4.4.2. Cancel

- Request: DELETE /{collection}/{id}/transfers/latest
- Request payload: N/A
- Response payload: Transfer cancel response message.

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

Example Cancel request:

```
C: DELETE /repp/v1/domains/example.nl/transfers/latest HTTP/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

#### 8.4.4.3. Reject

- Request: DELETE /{collection}/{id}/transfers/latest
- Request payload: Optional Transfer Rject request
- Response payload: Transfer response

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/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

#### 8.4.4.4. Approve

- Request: PUT /{collection}/{id}/transfers/latest
- Request payload: Optional Transfer Approve request
- Response payload: Transfer response.

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/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-cltrid: ABC-12345
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
```

#### 8.4.5. Update

- Request: PUT /{collection}/{id}
- Request payload: Object:update.
- Response payload: Update response message

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/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
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/1.1 200 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>
```

## 8.5. Extensions

- Request: \* /extensions/\*
- Request payload: \*

- Response payload: \*

EPP protocol extensions, as defined in [section 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 HTTP 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/1.1
C: Host: repp.example.nl
C: Cache-Control: no-cache
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
C: REPP-cltrid: ABC-12345
```

Example Extension response:

```
S: HTTP/1.1 200 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. Transport Considerations

[Section 2.1](#) of [[RFC5730](#)] of the EPP protocol specification describes considerations to be addressed by protocol transport mappings. This section addresses each of the considerations using a combination of features described in this document and features provided by HTTP as follows:

- 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 nomlonger preserved through EPP managed sessions. The session management is delegated to the stateless HTTP layer. Required EPP session 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.
- Batch-oriented processing (combining multiple EPP commands in a single HTTP request) **MUST NOT** be permitted.
- 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 or send a new request.

## 10. IANA Considerations

TODO: any?

## 11. Internationalization Considerations

TODO: any? Accept-Language in HTTP Header

## 12. 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.

## 13. Obsolete EPP Result Codes

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 2: Obsolete EPP result codes

## 14. Acknowledgments

TODO

## 15. References

### 15.1. 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](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)>.
- [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>>.
- [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>>.
-

## 15.2. 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](mailto:maarten.wullink@sidn.nl)

URI: <https://sidn.nl/>

### Marco Davids

SIDN Labs

Email: [marco.davids@sidn.nl](mailto:marco.davids@sidn.nl)

URI: <https://sidn.nl/>