Workgroup:	Network Working Group
Internet-Draft:	draft-epp-restful-transport-latest
Published:	16 November 2023
Intended Status:	Standards Track
Expires:	19 May 2024
Authors:	M. Wullink *SIDN Labs*	M. Davids *SIDN Labs*
---	---	---

Extensible Provisioning Protocol (EPP) RESTful Transport

Abstract

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

This document includes a mapping of [RFC5730] EPP commands to an HTTP based (RESTful) 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 stored on the EPP server. Each request from a client to the server MUST contain all of the information necessary for the server to process the request.

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 19 May 2024.

Copyright Notice

Table of Contents

1. Introduction

This document describes a new transport protocol for EPP, based on the [REST] architectural style. The newly defined transport 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].

Stateless 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 - The mapping of [RFC5730] XML commands to Stateless EPP.

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.

4. RESTful transport for EPP or REPP

REPP is designed to solve, in the spirit of [RFC3375], the drawbacks as mentioned in the next paragraph and yet maintain compatibility with existing object mapping definitions.

The design intent is to provide a clear, clean and self-explanatory interface that can easily be integrated with existing software systems. On the basis of these principles a [REST] architectural style was chosen. A client interacts with a REPP server via HTTP requests.

A server implementing REPP, MUST NOT keep any client state. Every client request needs to provide all of the information necessary to process the request.

REPP conforms to the EPP transport mapping considerations as defined in [RFC5730], Section 2.1. With REPP, the EPP [RFC5730] commands are mapped to REST URL resources.

5. Drawbacks Associated with Stateful EPP

[RFC5734] requires a stateful session between a client and the EPP server. This is accomplished by setting up a session with a <login> and keeping it alive for some time until issuing a <logout>. This may pose challenges in load-balanced environments, when a running session for whatever reason suddenly has to be switched from one EPP server to another and state is kept on a per server basis.

[RFC5734] EPP sessions can wind up in a state where they are no longer linked to an active TCP connection, especially in an environment where TCP connectivity is flaky. This may raise problems in situations where session limits are enforced.

REPP is designed to avoid these drawbacks, hence making the interaction between an EPP client and an EPP server more robust and efficient.

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 a client can use in registry operations MUST be identified by {context- root}/{version}/{collection}

o {context-root} is the base URL which MUST be specified by each registry. The {context-root} MAY be an empty, zero length string.

o {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.

o {collection} MUST be substituted by "domains", "hosts" or "contacts", referring to either [RFC5731], [RFC5732] or [RFC5733].

o 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 following this naming convention, would look like this:

/rest/v1/domains/example.com

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. Message Exchange

A [RFC5730] XML request includes a command- and object mapping to which a command must be applied. With REPP XML request messages are expressed by using a combination of a URL resource and an HTTP method.

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.

7.1. HTTP Method Definition

The operations on resources MUST be performed by using an HTTP method. 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.

The server MUST not support the following "verbs"

PATCH: Partial updating of a resource is MUST not be allowed.

7.2. REPP Request

7.2.1. EPP Data

The payload data for a REPP request MAY be transmitted to the server using the POST, PUT and GET HTTP methods.

POST and PUT: Payload data, when required, MUST be added to the message-body.

GET: When payload data is required, it concerns <authInfo>. This SHALL be put in the "X-REPP-auth-info" HTTP request-header.

7.2.2. 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 "X-REPP-" prefix.

X-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.

X-REPP-auth-info: The X-REPP-auth-info request-header is used as a mechanism for transporting the authorization information associated with the an object. The <authInfo> element is described in the EPP RFCs and MUST be used accordingly. It MUST contain the entire authorization information element as mentioned in Section 11.1.

7.2.3. Generic HTTP Headers

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

Accept-Language: This request-header is equivalent to the <lang> element in the EPP <login> command, expect that the usage of this header by the client is OPTIONAL. The server MUST support the use of HTTP Accept-Language header in client requests. The client MAY issue a <hello> to discover the languages known by the server. Multiple servers in a load-balanced environment SHOULD reply with consistent <lang> elements in a <greeting>. Clients SHOULD NOT expect that obtained <lang> information remains consistent between different requests. Languages not supported by the server default to "en".

Content-Type: ...

7.3. REPP Response

The server response is made up out of a HTTP Status-Code, HTTP response-headers and it MAY contain an EPP XML message in the HTTP message-body.

7.3.1. REPP Response Headers

HTTP response-headers are used to transmit additional response data to the client. All REPP HTTP headers must have the "X-REPP-" prefix.

X-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.

X-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.

X-REPP-eppcode: This header is the equivalent of the <result code> element in te 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.

X-REPP-avail: The EPP avail header is the alternative of the "avail" attribute of the object:name element in a check response and MUST be used accordingly.

7.3.2. 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: ....

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.

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.

The following EPP result codes specify an interface-, authorization-, authentication- or an internal server error and MUST NOT be used in REPP. Instead, when the related error occurs, an HTTP Status-Code MUST be returned in accordance to the mapping shown in Table 1.

EPP result code: MUST only return EPP result information relating to the EPP protocol. The HTTP header "X-REPP-eppcode" MUST be used for EPP result code information.

EPP result code and HTTP Status-Code mapping.

EPP result code	HTTP Status-Code
2000 unknown command	400
2201 authorization error	401
2202 Invalid authorization information	401
2101 unimplemented command	501

Table 1

8. Command Mapping

This section describes the details of the REST interface by referring to the [RFC5730] Section 2.9 Protocol Commands and defining how these are mapped to RESTful requests.

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. Each URL in the table is prefixed with "/repp/v1/". To make the table fit we use the following abbreviations:

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

{i}: An abbreviation for {id}: a domain name, host name, contact id or a message id.

(opt): The item is optional.

Command mapping from EPP to REPP.

EPP command	Method	Resource	Request payload	Response payload
Hello	OPTIONS	/	N/A	<greeting>
Login	N/A		N/A	N/A
Logout	N/A		N/A	N/A
Check	HEAD	{c}/{i}	N/A	N/A

EPP command	Method	Resource	Request payload	Response payload
Info	GET	{c}/{i}	AUTH(opt)	<info>
Poll request	GET	messages	N/A	<poll>
Poll ack	DELETE	messages/{i}	N/A	<poll> ack
Transfer (query)	GET	{c}/{i}/ transfer	AUTH(opt)	<transfer>
New password	PUT	password	password	N/A
Create	POST	{c}	<create>	<create>
Delete	DELETE	{c}/{i}	N/A	<delete>
Renew	PUT	{c}/{i}/ validity	<renew>	<renew>
Transfer (create)	POST	{c}/{i}/ transfer	<transfer>	<transfer>
Transfer (cancel)	DELETE	{c}/{i}/ transfer	N/A	<transfer>
Transfer (approve)	PUT	{c}/{i}/ transfer	N/A	<transfer>
Transfer (reject)	DELETE	{c}/{i}/ transfer	N/A	<transfer>
Update	PUT	{c}/{i}	<update>	<update>

Table 2

8.1. Hello

- Request: OPTIONS /repp/v1
- Request payload: N/A
- Response payload: <greeting>

The <greeting> (Section 2.4 RFC 5730) MUST NOT be automatically transmitted by the server with each new HTTP connection. The server MUST send a <greeting> element in response to a OPTIONS method on the root "/" resource.

A REPP client MUST NOT use a <hello> XML payload.

8.2. Password

- Request: PUT /repp/v1/password

- Request payload: New password
- Response payload: N/A

The client MUST use the HTTP PUT method on the password resource. This is the equivalent of the <newPW> element in the <login> command described in [RFC5730]. The request message-body MUST contain the new password which MUST be encoded using Base64 [RFC4648].

After a successful password change, the HTTP header "X-REPP-eppcode" must contain EPP result code 1000, otherwise an appropriate 2xxx range EPP result code.

8.3. Session Management Resources

The server MUST NOT create a client session. Login credentials MUST be added to each client request. This SHOULD be done by using any of the available HTTP authentication mechanisms. Basic authentication MAY be, all authentication mechanisms MUST be combined with TLS [RFC5246] for additional security.

To protect information exchanged between an EPP client and an EPP server [RFC5734] Section 9 level of security is REQUIRED.

8.3.1. Login

The <login> command MUST NOT be implemented by a server. The <newPW> element has been replaced by the Password resource. The <lang> element has been replaced by the Accept-Language HTTP request-header. The <svcs> element has no equivalent in RESTful EPP, the client can use a <hello> to discover the server supported namespace URIs. The server MUST check every XML namespace used in client XML requests. An unsupported namespace MUST result in the appropriate EPP result code.

8.3.2. Logout

The <logout> command MUST NOT be implemented by the server.

8.4. Query Resources

TODO: describe these resources use GET method and cannot send request mesage in HTTP messagebody must use resource URL as identifier and options auth header.

8.4.1. Check

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

The HTTP header X-REPP-avail with a value of "1" or "0" is returned, depending on whether the object can be provisioned or not.

A <check> request MUST be limited to checking only one resource {id} at a time. This may seem a step backwards when compared to the check command defined in the object mapping of the EPP RFCs where multiple object-ids are allowed inside a check command. The RESTful check operation can be load balanced more efficient when there is only a single resource {id} that needs to be checked.

The server MUST NOT support any object:reason elements described in the EPP object mapping RFCs.

8.4.2. Info

- Request: GET {collection}/{id}
- Request payload: OPTIONAL X-REPP-auth-info HTTP header with <authInfo>.
- Response payload: Object <info> response.

A object <info> request MUST be performed with the HTTP GET method on a resource identifying an object instance. The response MUST be a response message as described in object mapping of the EPP RFCs.

8.4.2.1. Domain Name

A domain name <info> differs from a contact- and host <info> in the sense that EPP Domain Name Mapping [RFC5731], Section 3.1.2 describes an OPTIONAL "hosts" attribute for the domain:name element. This attribute is mapped to additional REST resources to be used in a domain name info request.

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

The server MAY require the client to include additional authorization information. The authorization data MUST be sent with the "X-REPP- authinfo" HTTP request-header.

8.4.3. Poll

8.4.3.1. Poll Request

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

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

8.4.3.2. Poll Ack

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

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

8.4.3.3. Transfer Query Op

- Request: GET {collection}/{id}/transfer
- Request payload: Optional X-REPP-auth-info HTTP header with <authInfo>
- Response payload: Transfer query response message.

A <transfer> query MUST be performed with the HTTP GET method on the transfer resource of a specific object instance.

8.5. Object Transform Resources

8.5.1. Create

- Request: POST {collection}/
- Request payload: Object <create>.
- Response payload: Object <create> response.

A client MUST create a new object with the HTTP POST method in combination with an object collection.

8.5.2. Delete

- Request: DELETE {collection}/{id}
- Request payload: N/A
- Response payload: Object <delete> response.

Deleting an object from the registry database MUST be performed with the HTTP DELETE method on a REST resource specifying a specific object instance.

8.5.3. Renew

- Request: PUT {collection}/{id}/validity
- 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 validity resource.

8.5.4. 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 specific object resource. The payload MUST contain an <object: update> described in the EPP RFCs.

8.5.5. 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".

8.5.5.1. Create Op

o Request: POST {collection}/{id}/transfer

o Request payload: object:transfer.

o Response Payload: Transfer start response.

Initiating a transfer MUST be done by creating a new "transfer" resource with the HTTP POST method on a specific domain name or contact object instance. The server MAY require authorization information to validate the transfer request.

8.5.5.2. Cancel Op

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

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

8.5.5.3. Approve Op

- Request: PUT {collection}/{id}/transfer
- Request payload: N/A
- Response payload: Transfer approve response message.

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

8.5.5.4. Reject Op

- Request: DELETE {collection}/{id}/transfer
- Request payload: N/A
- Response payload: Transfer reject response message

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

9. Transport Considerations

Section 2.1 of the EPP core protocol specification [RFC5730] describes considerations to be addressed by protocol transport mappings. This document 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 or session.
- The stateful nature of EPP is no longer preserved through managed sessions. There still is a controlled message exchanges because HTTP uses TCP as transport layer protocol.
- HTTP 1.1 allows persistent connections which can be used to send multiple HTTP requests to the server using the same connection. The server MUST NOT allow persistent connections.
- The server MUST NOT allow pipelining and return EPP result code 2002 if pipelining is detected.
- Batch-oriented processing (combining multiple EPP commands in a single HTTP request) MUST NOT be permitted.
- Section 8 of this document describes features to frame EPP request data by adding the data to an HTTP request message-body or request-header.
- 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 another request.

10. IANA Considerations

TODO: This draft defines three resource collections; domains, contacts, hosts. This may require an IANA RESTful EPP collection protocol registry.

11. Internationalization Considerations

TODO

12. Security Considerations

[RFC5730] describes a <login> command for transmitting client credentials. This command MUST NOT be used for RESTful EPP. Due to the stateless nature of REST clients MUST transmit their credentials with each request. The validation of the user credentials must be performed by an out-of-band mechanism. This could be done with Basic and Digest access authentication [RFC2617] or with the use of OAuth [RFC5849].

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

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.
2002	Commands can now be sent in any order.
2100	The REPP URL path includes the version.
2200	The login command is not used anymore.

Table 3

14. Acknowledgments

TODO

15. 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>.

[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>.

[RFC3375] Hollenbeck, S., "Generic Registry-Registrar Protocol Requirements", RFC 3375, DOI 10.17487/RFC3375, September 2002, <https://www.rfc-editor.org/info/rfc3375>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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>.

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

Appendix A. Examples

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

A.1. X-REPP-auth-info

A.1.1. Domain Info with Authorization Data

The X-REPP-auth-info header in a Domain Info Request might look like this: TODO: X-REPP-auth-info must be removed, if auth data is neeeded, then complete req xml must be sent

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
  <extension>
     <re:rest xmlns:re="urn:ietf:params:xml:ns:restful-epp-1.0">
       <re:authorization>
          <re:pw>passwordfordomain</re:pw>
       </re:authorization>
     </re:rest>
  </extension>
</epp>
```

The HTTP header X-REPP-auth-info MUST contain the entire authorization information element, formatted as described in the EPP RFCs.

A.2. Hello Example

A.2.1. RESTful <hello> Request

```
C: OPTIONS /rest/v1/ HTTP/1.1
C: Host: repp.example.com
C: Cache-Control: no-cache
C: Authorization: Basic amRvZTp0ZXN0
C: Pragma: no-cache
C: Accept: application/epp+xml
C: Accept-Encoding: gzip,deflate
C: Accept-Language: en
C: Accept-Charset: utf-8
```

A.2.2. RESTful <hello> Response

```
S: HTTP/1.1 200 OK
S: Date: Sun, 10 Apr 2012 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 799
S: Content-Type: application/epp+xml
S: Connection: close
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:     <!-- rest of the greeting elements -->
S:   </greeting>
S: </epp>
```

A.3. Password Example

A.3.1. Change Password Request

```
C: PUT /rest/v1/password/ HTTP/1.1
C: Host: repp.example.com
C: Cache-Control: no-cache
C: Authorization: Basic amRvZTp0ZXN0
C: Pragma: no-cache
C: Accept-Language: en
C: Accept-Charset: utf-8
C: X-REPP-cltrid: ABC-12345
C: Content-Type: text/plain
C: Content-Length: 44
C:
C: bWFpbG1lYXQ6bWFhcnRlbi53dWxsaW5rQHNpZG4ubmw=
```

A.3.2. Change Password Response

```
S: HTTP/1.1 200 OK
S: Date: Sun, 10 Apr 2012 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 0
S: X-REPP-cltrid: ABC-12345
S: X-REPP-svtrid: 54321-XYZ
S: X-REPP-eppcode: 1000
S: Connection: close
```

A.4. Domain Create Example

A.4.1. Domain Create Request

TODO: remove extension from example below

```
C: POST /rest/v1/domains/ HTTP/1.1
C: Host: repp.example.com
C: Cache-Control: no-cache
C: Authorization: Basic amRvZTp0ZXN0
C: Pragma: no-cache
C: Accept-Language: en
C: Accept-Charset: utf-8
C: Accept: application/epp+xml
C: X-REPP-cltrid: ABC-12345
C: Content-Type: text/plain
C: Content-Length: 543

C: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
C: <epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
C:      xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:  <extension>
C:   <re:rest xmlns:re="urn:ietf:params:xml:ns:restful-epp-1.0">
C:    <domain:create>
C:     <!-- Object specific elements-->
C:    </domain:create>
C:   </re:rest>
C:  </extension>
C: </epp>
```

A.4.2. Domain Create Response:

```
S: HTTP/1.1 200 OK
S: Date: Sun, 10 Apr 2012 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 642
S: X-REPP-cltrid: ABC-12345
S: X-REPP-svtrid: 54321-XYZ
S: X-REPP-eppcode: 1000
S: Content-Type: application/epp+xml
S: Connection: close

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:           <!-- Object specific elements-->
S:       </domain:creData>
S:     </resData>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54321-XYZ</svTRID>
S:     </trID>
S:   </response>
S:</epp>
```

A.5. Domain Delete Example

A.5.1. Domain Delete Request:

```
C: DELETE /rest/v1/domains/example.com HTTP/1.1
C: Host: repp.example.com
C: Cache-Control: no-cache
C: Authorization: Basic amRvZTp0ZXN0
C: Pragma: no-cache
C: Accept-Language: en
C: Accept-Charset: utf-8
C: X-REPP-cltrid: ABC-12345
```

A.5.2. Domain Delete Response:

```
S: HTTP/1.1 200 OK
S: Date: Sun, 10 Apr 2012 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 505
S: X-REPP-cltrid: ABC-12345
S: X-REPP-svtrid: 54321-XYZ
S: X-REPP-eppcode: 1000
S: Content-Type: application/epp+xml
S: Connection: close

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>54321-XYZ</svTRID>
S:      </trID>
S:   </response>
S:</epp>
```

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/