Workgroup:	Network Working Group
Internet-Draft:	draft-rpp-core-01
Published:	6 March 2025
Intended Status:	Standards Track
Expires:	7 September 2025
Authors:	M. Wullink *SIDN Labs* P. Kowalik *DENIC*

RESTful Provisioning Protocol (RPP)

Abstract

This document describes a HTTP-based protocol for the provisioning and management of objects in a shared database.

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 7 September 2025.

Copyright Notice

Table of Contents

1. Introduction

This document describes an Application Programming Interface (API) API based on the HTTP protocol [RFC2616] and the principles of [REST]. Conforming to the REST constraints is generally referred to as being "RESTful". Hence the API is dubbed: "'RESTful Provisioning Protocol" or "RPP" for short.

RPP is data format agnostic, this document describes a framework describing protocol messages in any data format. the client uses server-driven content negotiation. Allowing the client to select from a set of representation media types supported by the server, such as JSON [RFC8259], XML or [YAML].

2. Terminology

In this document the following terminology is used.

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

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

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

RESTful Provisioning Protocol or RPP - The protocol described in this document.

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

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

RPP client - An HTTP user agent performing an RPP request

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

3. Conventions Used in This Document

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

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

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

4. Design Considerations

RPP is designed to improve the ease of design, development, deployment, and management of an provisioning service. This section lists the main design criteria.

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

- Security, allow for the use of authentication and authorization solutions available for HTTP based applications. HTTP provides an Authorization header Section 14.8 of [RFC2616].
- Content negotiation, A server may choose to include support for multiple media types. The client must be able to signal to the server what media type the server should expect for the request content and to use for the response content.
- Compatibility with existing EPP semantics defined in the EPP RFCs.
- Simplicity, when the semantics of a resource URL and HTTP method match an EPP command and request message, the use of a request message should be optional.
- Performance, reducing the number of required request and response messages, improves the performance and network bandwidth requirements for both client and server. Not every request will require a message to be created, marshalled, and transmitted.

5. Extension Framework

TODO

6. Resource Naming Convention

The naming convention SHOULD follow best practices defined for RESTful APIs, where the resource name should be a noun. A RPP resource can be a single unique object identifier e.g. a domain name or consist out of a collection of objects. A collection of objects available for registry operations MUST be identified by: `/{context-root}/{version}/{collection}`

- `{context-root}` is the base URL which MUST be specified, the {context-root} MAY be an empty, zero length string.
- `{version}` is a path segment which identifies the version of the RPP implementation.
- `{collection}` MUST be substituted by all collection of objects, e.g. "domains", "hosts" or "contacts" or other supported objects.

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 RPP object instance MUST be identified by {context-root}/ {version}/ {collection}/{id}

An example domain name resource for domain name example.nl is encoded into the following URL:

`/rpp/v1/domains/example.nl`

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

7. Session Management

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

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

A RPP 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 Layer

RPP uses the REST architectural style, each HTTP method is assigned a distinct behavior, requests are expressed by a URL referring to a resource, a HTTP method, HTTP headers and an optional message body containing the request message.

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

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

8.1. Content negotiation

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

The client MUST use the following HTTP headers:

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

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

8.2. Request

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

TODO: the non standard headers mentioned below are linked to EPP and may need to be removed or modified

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

8.3. Response

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

TODO: the non standard headers mentioned below are linked to EPP and may need to be removed.

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

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

8.4. Error Handling

RESTful Provisioning Protocol and HTTP protocol are both an application layer protocol, having their own status- and result codes and the server MUST NOT mix these two distinct codes.

9. Commands

A RPP command contains 4 distinct elements.

1. Resource defined by a URL
2. HTTP method to be used on the resource
3. Request message (Optional)
4. Response message (Optional)

Section 9, Paragraph 5 lists an overview of RPP commands, the subsequent sections provide details for each command. Resource URLs in the table are assumed to be using the prefix: "/{context-root}/{version}/". Some RPP endpoints do not require a request and/or response message, as is indicated by the table columns "Request" and "response".

- `{c}`: An abbreviation for {collection}: this MUST be substituted with "domains", "hosts", "contacts" or any other collection of objects.
- `{i}`: An abbreviation for an object id, this MUST be substituted with the value of a domain name, hostname, contact-id or a message-id or any other defined object.
- `Optional`: A request message is only required when the server requires additional data not avaliable otherwise.

Command | Method | Resource | Request | Response

Hello	OPTIONS	/	No	Yes
Login	N/A	N/A	N/A	N/A
Logout	N/A	N/A	N/A	N/A
Check	HEAD	/{c}/{i}	No	No
Info	GET	/{c}/{i}	Optional	Yes
Poll Request	GET	/messages	No	Yes
Poll Ack	DELETE	/messages/{i}	No	Yes
Create	POST	/{c}	Yes	Yes
Delete	DELETE	/{c}/{i}	Optional	Yes
Renew	POST	/{c}/{i}/renewal	Optional	Yes
Transfer Request	POST	/{c}/{i}/transfer	Optional	Yes
Transfer Query	GET	/{c}/{i}/transfer	Optional	Yes
Transfer Cancel	DELETE	/{c}/{i}/transfer	Optional	Yes

Transfer Approve	PUT	/{c}/{i}/transfer	Optional	Yes
Transfer Reject	DELETE	/{c}/{i}/transfer	Optional	Yes
Update	PATCH	/{c}/{i}	Yes	Yes
Extension [1]	*	/{c}/{i}/extension/*	*	*

Table 1: RPP Commands

[1] This mapping is used as a placeholder for future extensions

9.1. Hello

TODO: rename "Hello" and make it a new command for retrieving server details, such as supported extensions?

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

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

Example request:

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

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 799
S: Content-Type: application/rpp+json
S: Content-Language: en
S:
S: TODO
```

9.2. Login

RPP is stateless and MUST NOT maintain any client state and does not include a Login command. The client MUST include all information in a RPP request, required for the server to be able to properly process the request.

TODO

9.3. Logout

TODO

9.4. Query Resources

A RPP client MAY use the HTTP GET method for executing a query command only when no request data has to be added to the HTTP message body. Sending content using an HTTP GET request is discouraged in [RFC9110], there exists no generally defined semantics for content received in a GET request. When an RPP object requires additional information, 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

TODO: allow more more finegrained status response from a check, not just 0 or 1

The HTTP HEAD method MUST be used for object existence check. The response MUST contain the `RPP-Check-Avail` header and MAY contain the `RPP-Check-Reason` header. The value of the `RPP-Check-Avail` header MUST be "0" or "1", depending on whether the object can be provisioned or not.

The Check endpoint MUST be limited to checking only a single object-id per request, to allow the server to effiently load balance requests.

Example request for a domain name:

```
C: HEAD /rpp/v1/domains/example.nl HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

Example response:

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

9.4.2. Info

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

Example request for an object not using authorization information.

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

```
C: GET /rpp/v1/domains/example.nl HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

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

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

```
C: GET /rpp/v1/domains/example.nl HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
C: RPP-AuthInfo: secret-token
C: RPP-Roid: REG-XYZ-12345
```

Example Info response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 424
S: Content-Type: application/rpp+json
S: Content-Language: en
S: RPP-code: 1000
S:
S: TODO
```

9.4.2.1. Object Filtering

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

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

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

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

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

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

```
C: GET /rpp/v1/domains/example.nl?filter=hosts&val=del HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

9.4.3. Poll

9.4.3.1. Poll Request

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

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

Example request:

```
C: GET /rpp/v1/messages HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 312
S: Content-Type: application/rpp+json
S: Content-Language: en
S: RPP-code: 1301
S:
S: TODO
```

9.4.3.2. Poll Ack

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

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

Example request:

```
C: DELETE /rpp/v1/messages/12345 HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Language: en
S: RPP-code: 1000
S: RPP-Queue-Size: 0
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: Content-Length: 145
S:
S: TODO
```

9.4.4. Transfer Query

A transfer object may not exist, when no transfer has been initiated for the specified object. The client MUST use the HTTP GET method and MUST NOT add content to the HTTP message body.

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

Example domain name Transfer Query request without authorization information required:

```
C: GET /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

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

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

```
C: GET /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
C: RPP-AuthInfo: secret-token
```

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

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

```
C: GET /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-AuthInfo: secret-token
C: RPP-Roid: REG-XYZ-12345
C: Content-Length: 0
C:
```

Example Transfer Query response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 230
S: Content-Type: application/rpp+json
S: Content-Language: en
S: RPP-code: 1000
S:
S: TODO
```

9.5. Transform Resources

9.5.1. Create

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

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

Example Domain Create request:

```
C: POST /rpp/v1/domains HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Content-Type: application/rpp+json
C: Accept-Language: en
C: Content-Length: 220
C:
S: TODO
```

Example Domain Create response:

```
S: HTTP/2 200
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Language: en
S: Content-Length: 642
S: Content-Type: application/rpp+json
S: Location: https://rpp.example.nl/rpp/v1/domains/example.nl
S: RPP-code: 1000
S:
S: TODO
```

9.5.2. Delete

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

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

Example Domain Delete request:

```
C: DELETE /rpp/v1/domains/example.nl HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

Example Domain Delete response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 80
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: RPP-code: 1000
S:
S: TODO
```

9.5.3. Renew

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

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

Example Domain Renew request:

```
C: POST /rpp/v1/domains/example.nl/renewal?current-date=2024-01-01 HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Content-Type: application/rpp+json
C: Accept-Language: en
C: Content-Length: 0
C:
```

Example Domain Renew request, using 1 year period:

```
C: POST /rpp/v1/domains/example.nl/renewal?current-date=2024-01-01?unit=y&value=1
HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Content-Type: application/rpp+json
C: Accept-Language: en
C: Content-Length: 0
C:
```

Example Renew response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Language: en
S: Content-Length: 205
S: Location: https://rpp.example.nl/rpp/v1/domains/example.nl
S: Content-Type: application/rpp+json
S: RPP-code: 1000
S:
S: TODO
```

9.5.4. Transfer

The Transfer command is mapped to a nested resource, named "transfer". The semantics of the HTTP DELETE method are determined by the role of the client executing the DELETE method. The DELETE method is defined as "reject transfer" for the current sponsoring client of the object. For the new sponsoring client the DELETE method is defined as "cancel transfer".

TODO: allow for alternative more client friendly methods for transferring objects, maybe use interactive oauth2.0 flows?

9.5.4.1. Request

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

- Request message: Optional
- Response message: Status

In order to initiate a new object transfer process, the client MUST use the HTTP POST method on a unique resource to create a new transfer resource object. Not all RPP objects support the Transfer command.

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

Example request not using object authorization:

```
C: POST /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example request using object authorization:

```
C: POST /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: RPP-Cltrid: ABC-12345
C: RPP-AuthInfo: secret-token
C: Accept-Language: en
C: Content-Length: 0
```

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

```
C: POST /rpp/v1/domains/example.nl/transfer?unit=y&value=1 HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example Transfer response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Language: en
S: Content-Length: 328
S: Content-Type: application/rpp+json
S: Location: https://rpp.example.nl/rpp/v1/domains/example.nl/transfer
S: RPP-code: 1001
S:
S: TODO
```

9.5.4.2. Cancel

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

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

Example request:

```
C: DELETE /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 80
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: RPP-code: 1000
S:
S: TODO
```

9.5.4.3. Reject

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

The currently sponsoring client of the object MUST use the HTTP DELETE method to reject a started transfer process.

Example request:

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

Example Reject response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 80
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: RPP-code: 1000
S:
S: TODO
```

9.5.4.4. Approve

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

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

Example Approve request:

```
C: PUT /rpp/v1/domains/example.nl/transfer HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example Approve response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 80
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: RPP-code: 1000
S:
S: TODO
```

9.5.5. Update

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

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

TODO: when using JSON, also allow for JSON patch so client can send partial update data only?

Example request:

```
C: PATCH /rpp/v1/domains/example.nl HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Content-Type: application/rpp+json
C: Accept-Language: en
C: Content-Length: 252
C:
S: TODO
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Length: 80
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: RPP-code: 1000
S:
S: TODO
```

9.6. Extension Framework

TODO

9.6.1. Protocol Extension

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

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

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

Example Protocol Extension request:

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

```
C: DELETE /rpp/v1/extensions/domains/example.nl/deletion HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Language: en
S: Content-Length: 0
S: RPP-Svtrid: XYZ-12345
S: RPP-Cltrid: ABC-12345
S: RPP-code: 1000
S:
S: TODO
```

9.6.2. Object Extension

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

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

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

Example request:

```
C: POST /rpp/v1/ips HTTP/2
C: Host: rpp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/rpp+json
C: Accept-Language: en
C: RPP-Cltrid: ABC-12345
C: Content-Type: application/rpp+json
C: Content-Length: 220
C:
S: TODO
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Wed, 24 Jan 2024 12:00:00 UTC
S: Server: Example RPP server v1.0
S: Content-Language: en
S: Content-Length: 642
S: Content-Type: application/rpp+json
S: Location: https://rpp.example.nl/rpp/v1/ips/192.0.2.1
S: RPP-code: 1000
S:
S: TODO
```

9.6.3. Command-Response Extension

TODO

10. IANA Considerations

TODO

11. Internationalization Considerations

TODO

12. Security Considerations

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

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

Due to the stateless nature of RPP, the client MUST include the authentication credentials in each HTTP request. This MAY be done by using JSON Web Tokens (JWT) [RFC7519] or Basic authentication [RFC7617].

TODO: describe common modern authentication solutions such as oath2.0

13. 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>.

[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>.

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

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

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

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

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

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

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

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

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

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

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

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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

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

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

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

Authors' Addresses

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

Pawel Kowalik
DENIC
Email: pawel.kowalik@denic.de
URI: https://denic.de/