| | |
|---|---|
| Workgroup: | Network Working Group |
| Internet-Draft: | draft-wullink-restful-epp-json-00 |
| Published: | 15 January 2024 |
| Intended Status: | Standards Track |
| Expires: | 18 July 2024 |
| Authors: | M. Wullink<br>*SIDN Labs* | M. Davids<br>*SIDN Labs* |
|---|---|---|

# XML to JSON mapping for RESTful EPP

## Abstract

This document describes how an EPP [RFC5730] XML message can be translated to a JSON [RFC8259] messages for use with RESTful EPP.

## Status of This Memo

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

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

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

This Internet-Draft will expire on 18 July 2024.

## Copyright Notice

# Table of Contents

# 1. Introduction

The Extensible Provisioning Protocol (EPP) [RFC5730] uses an XML based protocol. The schemas for validating EPP XML messages are published as part of the EPP RFCs.

RESTful EPP (REPP), however has suport for multiple data formats such as the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259].

This document describes the rules for converting a valid EPP XML message to JSON message, which can be used with REPP.

# 2. Terminology

In this document the following terminology is used.

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.

# 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].

JSON is case sensitive. Unless stated otherwise, JSON specifications and examples provided in this document MUST be interpreted in the character case presented.

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

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

# 4. Conversion Rules

In general a single XML element allows for the following forms

1. Empty
2. Pure text content
3. Attributes only
4. Pure text content and attributes
5. Child elements with different names
6. Child elements with identical names
7. Child elements and contiguous text

## 4.1. Empty

An empty XML element MUST be mapped to to a key matching the name of the element and a null value.

XML:

```
<hello/>
```

JSON:

```
{
  "hello": null
}
```

## 4.2. Pure text content

An XML element containing text only MUST be mapped to a key matching the name of the element and the text MUST be used for the value

XML:

```
<lang>en</lang>
```

JSON:

```
{
  "lang": "en"
}
```

## 4.3. Attributes only

An XML element containing one or more atributes only, MUST be mapped to a JSON object matching the name of the element. Each XML attribute, prefixed using the @ character, MUST be added as a key-value pair to the object.

XML:

```
<msgQ count="5" id="12345"/>
```

JSON:

```
{
  "msgQ": {
     "@count": "5",
     "@id": "12345"
  }
}
```

## 4.4. Pure text content and attributes

An XML element containing one or more atributes and text content only, MUST be mapped to a JSON object matching the name of the element. The text content MUST, prefixed using the string `#text`, MUST be added as a key-value pair to the object.

XML:

```
<msg lang="en">Command completed successfully</msg>
```

JSON:

```
{
   "msg": {
      "@lang": "en",
      "#text": "Command completed successfully"
   }
}
```

## 4.5. Child elements with different names

An XML element containing one or more child elements, where each child uses an unique name, MUST be mapped to a JSON object matching the name of the element. Each child element MUST be added as a key-value pair to the parent object.

XML:

```
<trID>
   <clTRID>ABC-12345</clTRID>
   <svTRID>54321-XYZ</svTRID>
</trID>
```

JSON:

```
{
   "trID": {
      "clTRID": "ABC-12345",
      "svTRID": "54321-XYZ"
   }
}
```

## 4.6. Child elements with identical names

An XML element containing multiple child elements, where multiple child elements use the same name, MUST be mapped to a JSON object containing an array. The name of the array MUST match the name of the non-unique children, each child element MUST be converted to JSON and added to the array.

XML:

```
<host>
   <addr>192.0.2.1</addr>
   <addr>192.0.2.2</addr>
</host>
```

JSON:

```
{
   "host": {
      "addr": [
         "192.0.2.1",
         "192.0.2.2"
      ]
   }
}
```

## 4.7. Child elements and contiguous text

An XML element containing one or more child elements and contiguous text, MUST be mapped to a JSON object containing a key-value entry for each child element, the text value MUST result in a key named `#text`.

XML:

```
<msg lang="en">
   Credit balance low.
   <limit>100</limit>
   <bal>5</bal>
</msg>
```

JSON:

```
{
   "msg": {
      "@lang": "en",
      "limit": 100,
      "bal": 5,
      "#text": "Credit balance low."
   }
}
```

When child elements are mixed with multiple text segments, the resulting #text key-value entry MUST be an array, containing all text segments.

XML:

```
<msg lang="en">
   Credit balance low.
   <limit>100</limit>
   <bal>5</bal>
   Please increase balance.
</msg>
```

JSON:

```
{
   "msg": {
      "@lang": "en",
      "limit": 100,
      "bal": 5,
      "#text": ["Credit balance low.", "Please increase balance asap."]
   }
}
```

The rules above are based on the conversion approach found on [XMLCOM-WEB]

# 5. Examples

TODO: replace XML with JSON

## 5.1. Hello

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

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

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

Example Hello request:

```
C: OPTIONS /repp/v1/ HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C: REPP-cltrid: ABC-12345
C: Connection: keep-alive
```

Example Hello response:

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

## 5.2. Login

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

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

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

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

## 5.3. Logout

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

## 5.4. Query Resources

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

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

#### 5.4.1. Check

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

The server MUST support the HTTP HEAD method for the Check endoint, both client and server MUST not put any content to the HTTP message body. The response MUST contain the REPP-check-avail and MAY contain the REPP-check-reason header. The value of the REPP-check-avail header MUST be "0" or "1" as described in Section 2.9.2.1 of [RFC5730], depending on whether the object can be provisioned or not.

The REPP Check endpoint is limited to checking only a single resource {id} per request. This may seem a step backwards compared to the Check command defined in the [RFC5730] where multiple object-ids are allowed inside a Check command. The RESTful Check request can be load balanced more efficiently when a single resource {id} needs to be checked.

Example Check request for a domain name:

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

Example Check response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 0
S: REPP-cltrid: ABC-12345
S: REPP-svtrid: XYZ-12345
S: REPP-check-avail: 0
S: REPP-check-reason: In use
S: REPP-result-code: 1000
```

### 5.4.2. Info

An Info request MUST be performed using the HTTP GET method on a resource identifying an object instance. An object MAY have authorization attachted to it, the client then MUST use the HTTP POST method and include the authorization information in the request.

A request for an object not using authorization information.

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

A request for an object that has authorization information attached.

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

Example Info response:

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

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

```
### Poll

#### Poll Request

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

The client MUST use the HTTP GET method on the messages resource collection to
request the message at the head of the queue. The "op=req" semantics from [@!RFC5730,
Section 2.9.2.3] are assigned to the HTTP GET method.

Example Poll request:
```

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

Example Poll response:

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

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?> S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"> S: <response> S: <result code="1301"> S: <msg>Command completed successfully; ack to dequeue</msg> S: </result> S: <msgQ count="5" id="12345"> S: <qDate>2000-06-08T22:00:00.0Z</qDate> S: <msg>Transfer requested.</msg> S: </msgQ> S: <resData> S: S: </resData> S: <trID> S: <clTRID>ABC-12345</clTRID> S: <svTRID>XYZ-12345</svTRID> S: </trID> S: </response> S:</epp>

#### Poll Ack

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

The client MUST use the HTTP DELETE method on a message instance to to acknowledge receipt of a message of a message from the message queue. The "op=ack" semantics from [@!RFC5730, Section 2.9.2.3] are assigned to the HTTP DELETE method. The "msgID" from a received EPP message MUST be included in the message resource URL, using the {id} path element.

Example Poll Ack request:

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

Example Poll Ack response:

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

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

### Transfer Query

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

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

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

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

Example domain name Transfer Query request:

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

Example domain name Transfer Query request and authorization information in REPP-authInfo header:

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

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

```xml
C: POST /repp/v1/domains/example.nl/transfers/latest HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Accept-Language: en
C:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <transfer op="query">
C:      <domain:transfer
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:        <domain:authInfo>
C:          <domain:pw roid="MW12345-REP">secret</domain:pw>
C:        </domain:authInfo>
C:      </domain:transfer>
C:    </transfer>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Transfer Query response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 230
S: Content-Type: application/epp+xml
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <!-- The rest of the response is omitted here -->
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

## 5.5. Transform Resources

### 5.5.1. Create

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

The client MUST use the HTTP POST method to create a new object resource. If the EPP request results in a newly created object, then the server MUST return HTTP status code 201 (Created).

Example Domain Create request:

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

Example Domain Create response:

```
S: HTTP/2 201 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 642
S: Content-Type: application/epp+xml
S: Location: https://repp.example.nl/repp/v1/domains/example.nl
S:
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
S:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:   <response>
S:     <result code="1000">
S:       <msg>Command completed successfully</msg>
S:     </result>
S:     <resData>
S:       <domain:creData
S:         <!-- The rest of the response is omitted here -->
S:       </domain:creData>
S:     </resData>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54321-XYZ</svTRID>
S:     </trID>
S:   </response>
S:</epp>
```

#### 5.5.2. Delete

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

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

Example Domain Delete request:

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

Example Domain Delete response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 505
S: Content-Type: application/epp+xml

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

#### 5.5.3. Renew

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

The EPP Renew command is mapped to a nested resource, named "renewals". Not all EPP object types include support for the renew command. If the EPP request results in a renewal of the object, then the server MUST return HTTP status code 201 (Created).

Example Domain Renew request:

```
C: POST /repp/v1/domains/example.nl/renewals HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Content-Type: application/epp+xml
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: Content-Length: 325
C:
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <renew>
C:      <domain:renew
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:        <domain:curExpDate>2023-11-17</domain:curExpDate>
C:        <domain:period unit="y">1</domain:period>
C:      </domain:renew>
C:    </renew>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Renew response:

```
S: HTTP/2 201 CREATED
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 505
S: Location: https://repp.example.nl/repp/v1/domains/example.nl
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>
```

#### 5.5.4. Transfer

Transferring an object from one sponsoring client to another is specified in [RFC5731] and [RFC5733]. The Transfer command is mapped to a nested resource, named "transfers".

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

##### 5.5.4.1. Request

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

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

If the EPP request is successful, then the server MUST return HTTP status code 201 (Created). The client MAY choose to send an empty HTTP message body when the object is not linked to authorization information associated with a contact object. The server MUST also include the Location header in the HTTP response.

Example Create request not using using object authorization:

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

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

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

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

```
C: POST /repp/v1/domains/example.nl/transfers HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: Content-Length: 252

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

Example Transfer response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 328
S: Content-Type: application/epp+xml
S: Location: https://repp.example.nl/repp/v1/domains/example.nl/transfers/latest
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>
```

#### 5.5.4.2. Cancel

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

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

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

Example Cancel request:

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

Example Cancel response:

```
TODO
```

#### 5.5.4.3. Reject

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

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

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

Example Reject request:

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

Example Reject response:

```
TODO
```

#### 5.5.4.4. Approve

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

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

Example Approve request:

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

Example Approve response:

```
TODO
```

### 5.5.5. Update

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

An object Update request MUST be performed with the HTTP PUT method on a unique object resource. The payload MUST contain an Update request as described in the EPP RFCs. The HTTP response MUST contain a Location header and an optional EPP response message in the message body.

Example Update request:

```
C: POST /repp/v1/domains/example.nl HTTP/2
C: Host: repp.example.nl
C: Authorization: Bearer <token>
C: Accept: application/epp+xml
C: Content-Type: application/epp+xml
C: Accept-Language: en
C: REPP-svcs: urn:ietf:params:xml:ns:domain-1.0
C: Content-Length: 252

C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <update>
C:      <domain:update
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.nl</domain:name>
C:          <!-- The rest of the response is omitted here -->
C:      </domain:update>
C:    </update>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example Update response:

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

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

Example Update response, without EPP response in message body:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Language: en
S: Content-Length: 0
S: REPP-svtrid: XYZ-12345
S: REPP-cltrid: ABC-12345
S: REPP-eppcode: 1000
```

## 5.6. Extensions

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

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

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

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

Example Extension response:

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

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

# 6. IANA Considerations

TODO

# 7. Internationalization Considerations

TODO

# 8. Security Considerations

TODO

# 9. Acknowledgments

TODO

# 10. References

## 10.1. Normative References

**[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>.

**[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>.

**[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>.

### 10.2. 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>.

**[XMLCOM-WEB]** XML.com, "Converting Between XML and JSON", 2006, <https://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html>.

## 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/