| | |
|---|---|
| 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 *SIDN Labs* M. Davids *SIDN Labs*

# XML to JSON mapping for RESTful EPP

## Abstract

This document describes the rules for converting an EPP [RFC5730] XML message to a JSON [RFC8259] message for use with RESTful EPP [REF-TO-REPP-HERE].

## Status of This Memo

## 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 element(s) 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

### 5.1. Hello

The Hello request message does not exist in the context of REPP.

Example XML response:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<epp
   xmlns="urn:ietf:params:xml:ns:epp-1.0">
   <greeting>
      <svID>Example EPP server epp.example.com</svID>
      <svDate>2000-06-08T22:00:00.0Z</svDate>
      <svcMenu>
         <version>1.0</version>
         <lang>en</lang>
         <lang>fr</lang>
         <objURI>urn:ietf:params:xml:ns:obj1</objURI>
         <objURI>urn:ietf:params:xml:ns:obj2</objURI>
         <objURI>urn:ietf:params:xml:ns:obj3</objURI>
         <svcExtension>
            <extURI>http://custom/obj1ext-1.0</extURI>
         </svcExtension>
      </svcMenu>
      <dcp>
         <access>
            <all/>
         </access>
         <statement>
            <purpose>
               <admin/>
               <prov/>
            </purpose>
            <recipient>
               <ours/>
               <public/>
            </recipient>
            <retention>
               <stated/>
            </retention>
         </statement>
      </dcp>
   </greeting>
</epp>
```

Example JSON response:

XML namespaces are not converted to JSON and are ignored.

```
{
   "epp": {
      "@xmlns": "urn:ietf:params:xml:ns:epp-1.0",
      "greeting": {
         "svID": "Example REPP server v1.0",
         "svDate": "2000-06-08T22:00:00.0Z",
         "svcMenu": {
            "version": "1.0",
            "lang": [
               "en",
               "fr"
            ]
         },
         "dcp": {
            "access": {
               "all": null
            },
            "statement": {
               "purpose": {
                  "admin": null,
                  "prov": null
               },
               "recipient": {
                  "ours": null,
                  "public": null
               },
               "retention": {
                  "stated": null
               }
            }
         }
      }
   }
}
```

## 5.2. Login

The Login command is not used anynmore for REPP.

## 5.3. Logout

The Logout command is not used anynmore for REPP.

## 5.4. Check

The Check request and responses messages are not used anynmore for REPP.

### 5.4.1. Info

The Info request message is not used anynmore for REPP.

Example XML response:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<epp
   xmlns="urn:ietf:params:xml:ns:epp-1.0">
   <response>
      <result code="1000">
         <msg>Command completed successfully</msg>
      </result>
      <resData>
         <domain:infData
            xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
            <domain:name>example.com</domain:name>
            <domain:roid>EXAMPLE1-REP</domain:roid>
            <domain:status s="ok"/>
            <domain:registrant>jd1234</domain:registrant>
            <domain:contact type="admin">sh8013</domain:contact>
            <domain:contact type="tech">sh8013</domain:contact>
            <domain:ns>
               <domain:hostObj>ns1.example.com</domain:hostObj>
               <domain:hostObj>ns1.example.net</domain:hostObj>
            </domain:ns>
            <domain:host>ns1.example.com</domain:host>
            <domain:host>ns2.example.com</domain:host>
            <domain:clID>ClientX</domain:clID>
            <domain:crID>ClientY</domain:crID>
            <domain:crDate>1999-04-03T22:00:00.0Z</domain:crDate>
            <domain:upID>ClientX</domain:upID>
            <domain:upDate>1999-12-03T09:00:00.0Z</domain:upDate>
            <domain:exDate>2005-04-03T22:00:00.0Z</domain:exDate>
            <domain:trDate>2000-04-08T09:00:00.0Z</domain:trDate>
            <domain:authInfo>
               <domain:pw>2fooBAR</domain:pw>
            </domain:authInfo>
         </domain:infData>
      </resData>
      <trID>
         <clTRID>ABC-12345</clTRID>
         <svTRID>54322-XYZ</svTRID>
      </trID>
   </response>
</epp>
```

Example JSON response:

```
{
   "epp": {
      "@xmlns": "urn:ietf:params:xml:ns:epp-1.0",
      "response": {
         "result": {
            "@code": "1000",
            "msg": "Command completed successfully"
         },
         "resData": {
            "domain:infData": {
               "@xmlns:domain": "urn:ietf:params:xml:ns:domain-1.0",
               "domain:name": "example.com",
               "domain:roid": "EXAMPLE1-REP",
               "domain:status": {
                 "@s": "ok"
               },
               "domain:registrant": "jd1234",
               "domain:contact": [
                 {
                   "@type": "admin",
                   "#text": "sh8013"
                 },
                 {
                   "@type": "tech",
                   "#text": "sh8013"
                 }
               ],
               "domain:ns": {
                 "domain:hostObj": [
                   "ns1.example.com",
                   "ns1.example.net"
                 ]
               },
               "domain:host": [
                 "ns1.example.com",
                 "ns2.example.com"
               ],
               "domain:clID": "ClientX",
               "domain:crID": "ClientY",
               "domain:crDate": "1999-04-03T22:00:00.0Z",
               "domain:upID": "ClientX",
               "domain:upDate": "1999-12-03T09:00:00.0Z",
               "domain:exDate": "2005-04-03T22:00:00.0Z",
               "domain:trDate": "2000-04-08T09:00:00.0Z",
               "domain:authInfo": {
                 "domain:pw": "2fooBAR"
               }
            }
         },
         "trID": {
            "clTRID": "ABC-12345",
            "svTRID": "54322-XYZ"
         }
      }
   }
}
```

### 5.4.2. Poll

#### 5.4.2.1. 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. The "op=req" semantics from Section 2.9.2.3 are assigned to the HTTP GET method.

Example 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 response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 312
S: Content-Type: application/epp+xml
S: Content-Language: en
S: REPP-Eppcode: 1301
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="1301">
S:      <msg>Command completed successfully; ack to dequeue</msg>
S:    </result>
S:    <msgQ count="5" id="12345">
S:      <qDate>2000-06-08T22:00:00.0Z</qDate>
S:      <msg>Transfer requested.</msg>
S:    </msgQ>
S:    <resData>
S:       <!-- The rest of the response is omitted here -->
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

#### 5.4.2.2. 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 "op=ack" semantics from Section 2.9.2.3 are assigned to the HTTP DELETE method. The "msgID" attribute of a received EPP Poll message MUST be included in the message resource URL, using the {id} path element. The server MUST use REPP headers to return the EPP result code and the number of messages left in the queue. The server MUST NOT add content to the HTTP message body of a successfull response, the server may add content to the message body of an error response.

Example 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 response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: REPP-Eppcode: 1000
S: REPP-Queue-Size: 0
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: Content-Length: 145
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:    <msgQ count="0" id="12345"/>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

### 5.4.3. 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 use the HTTP GET method and MUST NOT add content to the HTTP message body.

- Request: GET {collection}/{id}/transfers/latest
- Request message: None
- Response message: Transfer Query response

Example domain name Transfer Query request without authorization information required:

```
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
```

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 REPP-AuthInfo header.

Example domain name Transfer Query request using 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-AuthInfo: secret-token
C: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
```

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

Example domain name Transfer Query request and authorization using REPP-AuthInfo and the REPP-Roid 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-AuthInfo: secret-token
C: REPP-Roid: REG-XYZ-12345
C: Content-Length: 0
C:
```

Example Transfer Query response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 230
S: Content-Type: application/epp+xml
S: Content-Language: en
S: REPP-Eppcode: 1000
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

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 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 /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/2 200
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example 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: REPP-Eppcode: 1000
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: 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 /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: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
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:    <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

The Renew command is mapped to a nested collection, named "renewals". 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 reponse MUST include the Location header for the renewed object.

Example Domain Renew request:

```
C: POST /repp/v1/domains/example.nl/renewals?current-date=2024-01-01 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: 0
C:
```

Example Domain Renew request, using 1 year period:

```
C: POST /repp/v1/domains/example.nl/renewals?current-date=2024-01-01?
unit=y&value=1 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: 0
C:
```

Example Renew response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Language: en
S: Content-Length: 205
S: Location: https://repp.example.nl/repp/v1/domains/example.nl
S: Content-Type: application/epp+xml
S: REPP-Eppcode: 1000
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 client 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 DELETE 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 message: None
- Response message: Status

To start a new object transfer process, the client MUST use the HTTP POST method for a unique resource to create a new transfer resource object, not all EPP objects support 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 transfer request is successful, then the reponse MUST include the Location header for the object being transferred.

Example request not 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: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: Accept-Language: en
C: REPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example request 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: REPP-Svcs: urn:ietf:params:xml:ns:domain-1.0
C: REPP-Cltrid: ABC-12345
C: REPP-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 /repp/v1/domains/example.nl/transfers?unit=y&value=1 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: REPP-Cltrid: ABC-12345
C: Content-Length: 0
```

Example Transfer response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example 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: REPP-Eppcode: 1001
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: None
- Response message: Status

The new sponsoring client MUST use the HTTP DELETE method to cancel a requested transfer. The semantics of the HTTP DELETE method are determined by the role of the client sending the request.

Example 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 response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
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:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

#### 5.5.4.3. Reject

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

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".

Example 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:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
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:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

#### 5.5.4.4. Approve

- Request: PUT /{collection}/{id}/transfers/latest
- Request message: None
- Response message: Status

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
C: Content-Length: 0
```

Example Approve response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
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:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>XYZ-12345</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

#### 5.5.5. 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.

Example request:

```
C: PATCH /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:
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 request is omitted here -->
C:      </domain:update>
C:    </update>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example response:

```
S: HTTP/2 200 OK
S: Date: Fri, 17 Nov 2023 12:00:00 UTC
S: Server: Example REPP server v1.0
S: Content-Length: 80
S: REPP-Svtrid: XYZ-12345
S: REPP-Cltrid: ABC-12345
S: REPP-Eppcode: 1000
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:    <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>.

### 10.2. Informative References

**[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/