Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 25, 2012

M. Wullink
SIDN
M. Davids
R. Gieben
SIDN Labs
April 23, 2012

RESTful interface for the Extensible Provisioning Protocol
draft-wullink-restful-epp-00

Abstract

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

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

With REPP, no session is created on the EPP server. Each request from client to server will contain all of the information necessary to understand the request. The server will close the connection after each HTTP request.

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 http://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 October 25, 2012.

Copyright Notice

Table of Contents

1. Introduction

This document describes a new EPP Protocol Extension and a mapping of [RFC5730] XML-commands to a [REST] interface which, in contrast to the current EPP specification, is stateless. It aims to provide a mechanism that is more suitable for complex, high availability environments, as well as for environments where TCP connections can be unreliable.

The newly defined protocol extensions described in this memo leverage the HTTP protocol [RFC2616] and the principles of [REST]. Conforming to the REST constraints is generally referred to as being "RESTful". Hence we dubbed the new protocol extension: "RESTful EPP" or "REPP" for short.

RFC 5730 [RFC5730] Section 2.1 describes that EPP can be layered over multiple transport protocols. Currently, the EPP transport over TCP [RFC5734] is the only widely deployed transport mapping for EPP. This same section defines that newly defined transport mappings must preserve the stateful nature of EPP.

With REPP, no session is created on the EPP server. Each request from client to server will contain all of the information necessary to understand the request. The server will close the connection after each HTTP request.

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

A good understanding of the EPP base protocol specification [RFC5730] is advised, to grasp the extension and mapping described in this document.

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

2. Terminology

In this document the following terminology is used.

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

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

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

Stateful EPP - The definition according to Section 2 of [RFC5730].

Stateless EPP or REPP - The RESTful EPP interface described in this document.

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

Resource - A network data object or service that can be identified by a URL.

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

3. Conventions Used in This Document

XML is case sensitive. Unless stated otherwise, XML specifications and examples provided in this document MUST be interpreted in the character case presented to develop a conforming implementation.

4. Stateless EPP or REPP

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

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

A server implementing REPP, MUST NOT keep any client state and is not compatible with [RFC5730], Section 2, which explicitly states that EPP is stateful.

REPP cannot be classified as an EPP transport mapping as defined in [RFC5730], Section 2.1. With REPP, the EPP [RFC5730] XML commands are mapped to a REST interface and as such, RESTful EPP is regarded as an interface mapping. Since REPP relies on a newly defined XSD schema with protocol elements, RESTful EPP can also be referred to as an [RFC5730], Section 2.7.1 protocol extension.

5. Drawbacks Associated with Stateful EPP

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

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

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

6. EPP Extension Framework

According to [RFC3735], Section 2, EPP provides an extension framework that allows features to be added at the protocol, object, and command-response levels. RESTful EPP (REPP) affects the following levels:

Protocol extension: RESTful EPP defines a new namespace "urn:ietf:params:xml:ns:restful-epp-1.0". It declares new elements, which MUST be used for RESTful EPP. The root element for the new namespace is the <rest> element. This element MUST contain an object mapping defined by the object mapping schemas.

Object extension: RESTful EPP does not define any new object level extensions. The existing object level extensions can be reused. However, any existing object mapping element, including any added extension elements it might contain, SHALL be added as a child to the new <rest> element.

Command-Response extension: RESTful EPP does not use the "command" concept, because the ’command’ concept is part of a RPC style and not a RESTful style. A REST URL and HTTP method combination have replaced the command structure. All command extensions can be reused as a rest extension.

RESTful EPP reuses the existing response messages defined in the EPP RFCs. The EPP response MUST be added to the standard <epp> element and SHALL NOT be part of any <rest> element.

The DNSSEC [RFC5910], E.164 number [RFC4114] and ENUM validation information [RFC5076] extension mapping elements can be added as children of the <rest> element.

7. Resource Naming Convention

A resource can be a single unique object identifier e.g. a domain name, or a collection of objects. The complete set of objects a client can use in registry operations MUST be identified by {context-root}/{version}/{collection}

- {context-root} is the base URL which MUST be specified by each registry.
- {version} is a label which identifies the interface version. This is the equivalent of the <version> element in the EPP RFCs.
- {collection} MUST be substituted by "domains", "hosts" or "contacts", referring to either [RFC5731], [RFC5732] or [RFC5733].
- A trailing slash MAY be added to each request. Implementations MUST consider requests which only differ with respect to this trailing slash as identical.

A specific object instance MUST be identified by {context-root}/{version}/{collection}/{id} where {id} is a unique object identifier described in EPP RFCs.

An example domain name resource following this naming convention, would look like this:

/rest/v1/domains/example.com

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

With RESTful EPP the object identifiers are embedded in URLs. This makes any object identifier in the request messages superfluous. However, since the goal of RESTful EPP is to stay compatible with the existing EPP object mapping schemas, this redundancy is accepted as a trade off. Removing the object identifier from the request message would require new object mapping schemas.

The server MUST return HTTP Status-Code 412 when the object identifier (for example <domain:name>, <host:name> or <contact:id>) in the HTTP message-body does not match the {id} object identifier in

the URL.

8. Message Exchange

A [RFC5730] request includes a command- and object mapping to which a command must be applied. With RESTful EPP, some of the request messages are expressed by a combination of a resource and an HTTP method.

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

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

8.1. HTTP Method Definitions

The operations on resources MUST be performed by an HTTP method. The server MUST support the following "verbs" ([REST]).

GET: Request a representation of a resource or a collection of resources.

PUT: Update an existing resource.

POST: Create a new resource.

DELETE: Delete an existing resource.

HEAD: Check for the existence of a resource.

OPTIONS: Request a greeting.

8.2. REPP Request

8.2.1. Payload Data

The payload data of a RESTful EPP request can be transmitted to the server using the POST, PUT and GET HTTP methods.

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

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

8.2.2. Request Headers

HTTP request-headers are used to transmit additional or optional request data to the server. All RESTful EPP HTTP headers must have the "X-REPP-" prefix.

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

X-REPP-authinfo: The X-REPP-authinfo request-header is the alternative of the <authInfo> element in the EPP RFCs and MUST be used accordingly. It MUST contain the entire authorization information element as mentioned in Section 11.1.

8.2.3. General Headers

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

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

8.3. REPP Response

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

8.3.1. Response Headers

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

X-REPP-svtrid: This header is the equivalent of the <svTRID> element in the EPP RFCs and MUST be used accordingly. If an HTTP message-body with the EPP XML equivalent <svTRID> exists, both values MUST be consistent.

X-REPP-cltrid: This header is the equivalent of the <clTRID> element in the EPP RFCs and MUST be used accordingly. If an HTTP message-body with the EPP XML equivalent <clTRID> exists, both values MUST be consistent.

X-REPP-eppcode: This header is the equivalent of the <result code> element in te EPP RFCs and MUST be used accordingly.If an HTTP message-body with The EPP XML equivalent <result code> exists, both values MUST be consistent.

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

8.3.2. General Headers

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

Cache-Control: This general-header... [TBD: the idea is to prohibit caching. Even though it will probably work and be useful in some scenario’s, it also complicates matters.]

Connection: The server MUST add the "Connection: close" general-header to each HTTP response.

8.4. Error Handling

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

HTTP Status-Code: MUST only return status information related to the HTTP protocol, When there is a mismatch between the object identifier in the HTTP message-body and the resource URL HTTP Status-Code 412 MUST be returned.

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

mapping shown in Table 1.

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

EPP result code and HTTP Status-Code mapping.

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

Table 1

9. Interface Mapping

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

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

The following table lists them all and the subsequent sections provide details for each request. Each URL in the table is prefixed with "/rest/v1/". To make the table fit we use the following abbreviations:

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

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

(opt): The item is optional.

Command mapping from Stateful EPP to Stateless EPP.

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

Table 2

9.1. Hello

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

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

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

9.2. Password

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

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

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

9.3. Session Management Resources

The server MUST NOT create a client session. Login credentials MUST be added to each client request. This SHOULD be done with any of the well known HTTP authentication mechanisms. Basic authentication MAY be used but MUST be combined with TLS [RFC5246] for added security.

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

9.3.1. Login

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

9.3.2. Logout

The <logout> command MUST NOT be implemented by the server. The server MUST add the "Connection: close" HTTP general-header to each response.

9.4. Query Resources

9.4.1. Check

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

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

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

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

9.4.2. Info

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

A object <info> request MUST be performed with the HTTP GET method on a resource identifying an object instance. The response MUST be a response message as described in object mapping of the EPP RFCs, possibly extended with an [RFC3915] extension element (<rgp: infData>).

9.4.2.1. Domain Name

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

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

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

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

9.4.3. Poll

9.4.3.1. Poll Request

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

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

9.4.3.2. Poll Ack

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

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

9.4.4. Transfer Query Op

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

A <transfer> query MUST be performed with the HTTP GET method on the

transfer resource of a specific object instance.

9.5. Object Transform Resources

9.5.1. Create

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

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

9.5.2. Delete

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

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

9.5.3. Renew

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

Renewing an object is only specified by [RFC5731], the <renew> command has been mapped to a validity resource.

9.5.4. Update

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

An object <update> request MUST be performed with the HTTP PUT method on a specific object resource. The payload MUST contain an <object:

update> described in the EPP RFCs, possibly extended with [RFC3915] <update> extension elements.

9.5.5. Transfer

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

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

9.5.5.1. Create Op

- Request: POST {collection}/{id}/transfer
- Request payload: <object:transfer>.
- Response Payload: Transfer start response.

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

9.5.5.2. Cancel Op

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

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

9.5.5.3. Approve Op

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

The current sponsoring client MUST use the HTTP PUT method to approve

a transfer requested by the new sponsoring client.

9.5.5.4. Reject Op

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

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

10. Transport Considerations

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

- HTTP is an application layer protocol which uses TCP as a transport protocol. TCP includes features to provide reliability, flow control, ordered delivery, and congestion control. Section 1.5 of RFC 793 describes these features in detail; congestion control principles are described further in RFC 2581 and RFC 2914. HTTP is a stateless protocol and as such it does not maintain any client state or session.
- The stateful nature of EPP is no longer preserved through managed sessions. There still is a controlled message exchanges because HTTP uses TCP as transport layer protocol.
- HTTP 1.1 allows persistent connections which can be used to send multiple HTTP requests to the server using the same connection. The server MUST NOT allow persistent connections.
- The server MUST NOT allow pipelining and return EPP result code 2002 if pipelining is detected.
- Batch-oriented processing (combining multiple EPP commands in a single HTTP request) MUST NOT be permitted.
- Section 8 of this document describes features to frame EPP request data by adding the data to an HTTP request message-body or request-header.

- o A request processing failure has no influence on the processing of other requests. The stateless nature of the server allows a client to retry a failed request or send another request.

11. Formal Syntax

The extension used by RESTful EPP is specified in XML Schema notation. The formal syntax presented here is a complete schema representation of RESTful EPP suitable for automated validation of EPP XML instances. The schema is based on the XML schemas defined in [RFC5730]. [RFC3735] Section 2.3 states that it MUST be announced in the <greeting> element.

11.1. RESTful EPP XML Schema

The RESTful EPP Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:repp="urn:ietf:params:xml:ns:restful-epp-1.0"
        xmlns:epp="urn:ietf:params:xml:ns:epp-1.0"
        xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
        xmlns="HTTP://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:ietf:params:xml:ns:restful-epp-1.0"
        elementFormDefault="qualified">

  <!--  Import common element types.   -->
  <import namespace="urn:ietf:params:xml:ns:eppcom-1.0"
          schemaLocation="eppcom-1.0.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:epp-1.0"
          schemaLocation="epp-1.0.xsd"/>

  <annotation>
     <documentation>
        RESTful EPP schema.
     </documentation>
  </annotation>

  <!-- The rest element should be used as extension root. -->
  <element name="rest" type="epp:extAnyType"/>

  <!-- A request which requires auth info can use this
       authorization shortcut without an object id. -->

  <element name="authorization" type="re:authInfoType"/>

  <!-- The authinfo element. For use with domain and host info
       and domain transfer. -->
  <complexType name="authInfoType">
     <choice>
        <element name="pw" type="eppcom:pwAuthInfoType"/>
        <element name="ext" type="eppcom:extAuthInfoType"/>
     </choice>
  </complexType>

</schema>
```

Figure 1

12. IANA Considerations

[TBD: This draft defines three resource collections; domains, contacts, hosts. This may require an IANA RESTful EPP collection protocol registry. RFC3688 defines an IANA XML Registry and 'restful-epp-1.0' defined here would have to be added to that: http://www.iana.org/assignments/xml-registry-index.html]

13. Internationalization Considerations

[TBD: Do we need them?]

14. Security Considerations

RFC 5730 describes a <login> command for transmitting client credentials. This command MUST NOT be used for RESTful EPP. Due to the stateless nature of REST clients MUST transmit their credentials with each request. The validation of the user credentials must be performed by an out-of-band mechanism. This could be done with Basic and Digest access authentication [RFC2617] or with the use of OAuth [RFC5849].

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

15. Obsolete EPP Result Codes

The following result codes specified in [RFC5730] are no longer meaningful in RESTful EPP and MUST NOT be used.

Code	Reason
1500	The logout command is not used anymore.
2002	Commands can now be sent in any order.
2100	The protocol version is embedded in the base URL of the interface.
2200	The login command is not used anymore.

16. References

16.1. Normative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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, June 1999.

[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, June 1999.

[RFC3915] Hollenbeck, S., "Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP)", RFC 3915, September 2004.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, August 2009.

[RFC5731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", STD 69, RFC 5731, August 2009.

[RFC5732] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Host Mapping", STD 69, RFC 5732, August 2009.

[RFC5733] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Contact Mapping", STD 69, RFC 5733, August 2009.

[RFC5734] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Transport over TCP", STD 69, RFC 5734, August 2009.

16.2. Informative References

[RFC3375] Hollenbeck, S., "Generic Registry-Registrar Protocol Requirements", RFC 3375, September 2002.

[RFC3735] Hollenbeck, S., "Guidelines for Extending the Extensible

Provisioning Protocol (EPP)", RFC 3735, March 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4114] Hollenbeck, S., "E.164 Number Mapping for the Extensible Provisioning Protocol (EPP)", RFC 4114, June 2005.

[RFC5076] Hoeneisen, B., "ENUM Validation Information Mapping for the Extensible Provisioning Protocol", RFC 5076, December 2007.

[RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

[RFC5910] Gould, J. and S. Hollenbeck, "Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)", RFC 5910, May 2010.

Appendix A. Examples

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

A.1. X-REPP-authinfo

A.1.1. Domain Info with Authorization Data

The X-REPP-authinfo header in a Domain Info Request might look like this:

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

So this HTTP header MUST contain the entire authorization information

element as mentioned in Section 11.1.

A.2. Hello Example

A.2.1. RESTful <hello> Request:

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

A.2.2. RESTful <hello> Response:

```
S: HTTP/1.1 200 OK
S: Date: Sun, 10 Apr 2012 12:00:00 UTC
S: Server: Acme REPP server v1.0
S: Content-Length: 799
S: Content-Type: application/epp+xml
S: Connection: close
S:
S: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
S: <epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
S:   <greeting>
S:     <!-- rest of the greeting elements -->
S:   </greeting>
S: </epp>
```

A.3. Password Example

A.3.1. RESTful Change Password Request:

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

A.3.2. RESTful Change Password Response:

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

A.4. Domain Create Example

A.4.1. RESTful Domain Create Request:

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

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

A.4.2. RESTful Domain Create Response:

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

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
S:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:   <response>
S:      <result code="1000">
S:         <msg>Command completed successfully</msg>
S:      </result>
S:      <resData>
S:         <domain:creData
S:             <!-- Object specific elements-->
S:         </domain:creData>
S:      </resData>
S:      <trID>
S:         <clTRID>ABC-12345</clTRID>
S:         <svTRID>54321-XYZ</svTRID>
S:      </trID>
S:   </response>
S:</epp>
```

A.5. Domain Delete Example

A.5.1. RESTful Domain Delete Request:

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

A.5.2. RESTful Domain Delete Response:

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

S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
S:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:   <response>
S:      <result code="1000">
S:         <msg>Command completed successfully</msg>
S:      </result>
S:      <trID>
S:         <clTRID>ABC-12345</clTRID>
S:         <svTRID>54321-XYZ</svTRID>
S:       </trID>
S:   </response>
S:</epp>
```

Authors' Addresses

Maarten Wullink
SIDN
Meander 501
Arnhem, 6825 MD
NL

Phone: +31 26 3525555
Email: maarten.wullink@sidn.nl
URI: https://sidn.nl/

Marco Davids
SIDN Labs
Meander 501
Arnhem, 6825 MD
NL

Phone: +31 26 3525555
Email: marco.davids@sidn.nl
URI: https://sidn.nl/

R. (Miek) Gieben
SIDN Labs
Meander 501
Arnhem, 6825 MD
NL

Phone: +31 26 3525555
Email: miek.gieben@sidn.nl
URI: https://sidn.nl/