Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: November 11, 2017

A. Fregly
S. Sheth
S. Hollenbeck
Verisign Labs

May 10, 2017

Registration Data Access Protocol (RDAP) Search Using POSIX Regular Expressions
draft-fregly-regext-rdap-search-regex-02

Abstract

The Registration Data Access Protocol (RDAP) provides limited search functionality based on pattern matching. This document describes an RDAP query extension that provides additional search functionality using POSIX extended regular expressions.

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 November 11, 2017.

Copyright Notice

Table of Contents

1. Introduction

The search patterns for Registration Data Access Protocol (RDAP) search as described in RFC 7482 [RFC7482] are limited. The protocol described in this specification extends RDAP search capabilities by adding path segments for RDAP search functions using a RESTful web service and POSIX [IEEE.1003.1_2013_EDITION] extended regular expressions. The service is implemented using the Hypertext Transfer Protocol (HTTP) [RFC7230] and the conventions described in RFC 7480 [RFC7480].

1.1. 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 RFC 2119 [RFC2119].

2. RDAP Path Segment Specification

The path segments defined in this section are OPTIONAL extensions of path segments defined in RFC 7482 [RFC7482]. The resource type path segments for search are:

- 'domains': Used to identify a domain name information search using a pattern to match a fully-qualified domain name.
- 'nameservers': Used to identify a name server information search using a pattern to match a host name.
- 'entities': Used to identify an entity information search using a pattern to match a string identifier.

The search patterns in the path segments MUST be POSIX extended regular expressions. Non-URL-safe characters in Search patterns MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. Percent-encoding will eliminate errors that might occur due to web-servier or app-server interpretation of certain unsafe characters and will eliminate errors due to inconsistent encoding and decoding semantics for certain characters. For instance, the space character may be encoded as "+" when submitted through a HTML form and encoded as "%20" when submitted through the address bar of a Web browser. Detailed results can be retrieved using the HTTP GET method and the path segments specified here.

This document defines an RDAP query parameter, "searchtype", that is used to identify search requests that require specialized processing beyond the limited functionality described in RFC 7482 [RFC7482]. Search processing using POSIX [IEEE.1003.1_2013_EDITION] extended regular expressions is indicated with a query string parameter value of "regex", e.g. "searchtype=regex". Other forms of search processing are possible and can be described in other specifications using other values for the "searchtype" query parameter. See Section 2.4 for additional information.

2.1. Domain Search

Syntax: domains?name=<domain search pattern>&searchtype=regex

Syntax: domains?nsLdhName=<domain search pattern>&searchtype=regex

Syntax: domains?nsIp=<domain search pattern>&searchtype=regex

Searches for domain information by name are specified using this form:

domains?name=XXXX&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then XXXX MUST be a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against domains in a name space administered by the server operator. Domain names are as defined by RFC 5890 [RFC5890] in "letters, digits, hyphen" format. The following URL would be used to find information for domain names matching the "e[a-z]ample\.com" pattern:

https://example.com/rdap/domains?name=e%5Ba-z%5Dample%5C.com&searchtype=regex

Internationalized Domain Names (IDNs) in U-label format [RFC5890] can also be matched by POSIX extended regular expression search patterns. Search patterns for these names are of the form /domains?name=XXXX&searchtype=regex, where XXXX is a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against domain names in U-label format. See section 6.1 of RFC 7482 [RFC7482] for information describing U-label character encoding. See Section 5 for other considerations relative to regular expression matching of IDNs.

Searches for domain information by name server name are specified using this form:

domains?nsLdhName=YYYY&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then YYYY MUST be a POSIX extended regular expression. Non-URL-safe characters in YYYY MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against host names in a name space administered by the server operator. Host names are as defined by RFC 5890 [RFC5890] in "letters, digits, hyphen" format. The following URL would be used to search for domains delegated to name servers matching the "ns[1-9]\.e[a-z]ample\.com" pattern:

https://example.com/rdap/domains?nsLdhName=ns%5B1-9%5D%5C.e%5Ba-z%5Dample%5C.com&searchtype=regex

Searches for domain information by name server IP address are specified using this form:

domains?nsIp=ZZZZ&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then ZZZZ MUST be a POSIX extended regular expression. Non-URL-safe characters in ZZZZ MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against IPv4 addresses [RFC1166] and IPv6 addresses [RFC5952] associated with specific name servers. The following URL would be used to search for domains that have been delegated to name servers that have IP addresses matching the "192\.0\.[1-9]\.0" pattern:

https://example.com/rdap/
domains?nsIp=192%5C.0%5C.%5B1-9%5D%5C.0&searchtype=regex

2.2. Name Server Search

Syntax: nameservers?name=<name server search pattern>&searchtype=regex

Syntax: nameservers?ip=<name server search pattern>&searchtype=regex

Searches for name server information by name server name are specified using this form:

nameservers?name=XXXX&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then XXXX MUST be a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against name server names in a name space administered by the server operator. Name server names are as defined in RFC 5890 [RFC5890] in "letters, digits, hyphen" format. Matches will return information for the matching name servers. The following URL would be used to find information for name server names matching the "ns[1-9]\.e[a-z]ample\.com" pattern:

https://example.com/rdap/nameservers?name=ns%5B1-9%5D%5C.e%5Ba-z%5Dample%5C.com&searchtype=regex

Internationalized name server names in U-label format [RFC5890] can also be matched by POSIX extended regular expression search patterns. Search patterns for these names are of the form /nameservers?name=XXXX&searchtype=regex, where XXXX is a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against name server names in U-label format. See section 6.1

of RFC 7482 [RFC7482] for information describing U-label character encoding. See Section 5 for other considerations relative to regular expression matching of U-labels.

Searches for name server information by name server IP address are specified using this form:

nameservers?ip=YYYY&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then YYYY MUST be a POSIX extended regular expression. Non-URL-safe characters in YYYY MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against IPv4 addresses [RFC1166] and IPv6 addresses [RFC5952] associated with specific name servers. The following URL would be used to search for name server names that resolve to addresses matching the "192\.0\.[1-9]\.0" pattern:

https://example.com/rdap/
nameservers?ip=192%5C.0%5C.%5B1-9%5D%5C.0&searchtype=regex

2.3. Entity Search

Syntax: entities?fn=<entity name search pattern>&searchtype=regex

Syntax: entities?handle=<entity handle search pattern>&searchtype=regex

Searches for entity information by name are specified using this form:

entities?fn=XXXX&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then XXXX MUST be a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against the "FN" property of an entity (such as a contact, registrant, or registrar) name as specified in Section 5.1 of RFC 7483 [RFC7483]. The following URL would be used to find information for entity names matching the "Bobby[[:space:]]Joe[a-z]*" pattern:

https://example.com/rdap/
entities?fn=Bobby%5B%5B%3Aspace%3A%5D%5DJoe%5Ba-z%5D%2A&searchtype=regex

Searches for entity information by handle are specified using this form:

entities?handle=XXXX&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then XXXX MUST be a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against an entity (such as a contact, registrant, or registrar) identifier whose syntax is specific to the registration provider. The following URL would be used to find information for entity handles matching the "CID-4[0-9]*" pattern:

https://example.com/rdap/entities?handle=CID-4%5B0-9%5D%2A&searchtype=regex

2.4. Future Path Segments

OPTIONAL extensions to new RDAP path segments defined in future RDAP specifications MAY be implemented to support POSIX extended regular expressions search capability. The syntax for such OPTIONAL extensions MUST be modeled on the syntax defined in Section 2.1, Section 2.2, and Section 2.3. The following syntax template MUST be followed:

Syntax: {path_segment}?{property}=XXXX&searchtype=regex

If the URL query string parameter "searchtype" has a value of "regex", then XXXX MUST be a POSIX extended regular expression. Non-URL-safe characters in XXXX MUST be percent-encoded. Percent-encoding MUST be as described in section 2.1 of RFC 3986 [RFC3986]. The supplied regular expression will be matched against the property specified by {property} for the path segment specified by {path_segment}. For example, if a new RDAP path segment "foo" is defined and has a property "bar", the following URL would be used to find information for the "foo" resource type with a "bar" property matching the "widget:.*mech.*" pattern:

https://example.com/rdap/foo?bar=widget%3A.%2Amech.%2A&searchtype=regex

3. Search Pattern Syntax

POSIX extended regular expression search pattern syntax is defined in Section 9 of IEEE Std 1003.1, 2013 Edition [IEEE.1003.1_2013_EDITION]. An RDAP service implementation MAY

implement a subset of the extended regular expression syntax and capabilities defined by the specification. An RDAP service implementation MUST specify the regular expression syntax and capabilities it supports in response to a query to the /help path segment as specified in section 3.1.6 of RFC 7482 [RFC7482].

Characters within a regular expression search pattern may be URI reserved characters. To avoid ambiguity in parsing a URL containing a regular expression search pattern, non-URL-safe character in the regular expression search pattern MUST be percent-encoded as described in RFC 3986 [RFC3986].

4. Query Processing

RDAP clients using regular expression search patterns MUST percent-encode non-URL-safe characters in the regular expression search pattern as described in RFC 3986 [RFC3986]. The regular expression SHOULD be consistent with the regular expression syntax and capabilities supported by the RDAP service implementation that is being queried in order to provide predictable results. The use of a regular expression that is not consistent with the capabilities of the RDAP service implementation MUST result in the return of an HTTP 400 response code as described in section 5.4 of RFC 7480 [RFC7480].

An RDAP service implementation will receive regular expressions search patterns that contain percent-encoded characters. Prior to processing a regular expression, the RDAP service MUST decode the received percent-encoded characters in regular expressions as described in RFC 3986 [RFC3986]. After decoding the received regular expression, the regular expression MUST be matched as described in Section 2.1, Section 2.2 and Section 2.3. Matching records related to the search are then returned in the client.

The POSIX regular expression specification [IEEE.1003.1_2013_EDITION] allows implementations to provide case insensitive searching. RDAP service implementations SHOULD implement case insensitive searching as described in the specification. This will allow for consistency in search results regardless of the case of the RDAP data being searched. For example, some RDAP service implementations may represent domain names in upper case during searching while other RDAP service implementations may represent domain names in lower case or mixed case during searching. Case insensitive searching will alleviate the need for search clients to know how each RDAP service implementation represents the case of searchable data. RDAP service implementations that do not perform case insensitive searching may produce unexpected search results for entities that are not aware of how the service represents the case of searchable data.

An RDAP service implementation MUST specify its support or lack of support for case insensitive searching in response to a query to the /help path segment as specified in section 3.1.6 of RFC 7482 [RFC7482].

Servers indicate the success or failure of query processing of a regular expression search pattern by returning an appropriate HTTP response code to the client. Response codes not specifically identified in this document are described in RFC 7480 [RFC7480].

5. Internationalization Considerations

An RDAP service implementation that supports regular expression search patterns MUST support pattern construction and pattern matching using UTF-8 encoded character strings. Other character encoding considerations are described in section 6.1 of RFC 7482 [RFC7482].

6. Implementation Considerations

The set of related records that may be returned in response to a search with a regular expression search pattern are subject to the constraints specified in section 4.2 of RFC 7482 [RFC7482].

An RDAP service implementation MAY choose to limit the scope of searches to RDAP data that is managed by the RDAP service implementation. For example, an RDAP response to a query that could be matched against multiple TLDs or data in related RDAP repositories (such as those distributed between domain registry and domain registrar) need only return matches for the data managed by the RDAP service implementation.

Implementators should take care to ensure that decoding of percent-encoded characters in a received regular expression is only performed once. Standard APIs for processing HTTP requests will likely perform decoding of percent-encoded characters prior to providing a recieved regular expression to the RDAP service implementation code. In such case case, the RDAP service implementation code should not attempt to perform decoding for percent-encoded characters.

Regular expression matching results for some search patterns may vary based on the regular expression search engine used, the version of the engine used, and configuration of the search engine. For example, POSIX [IEEE.1003.1_2013_EDITION] defines different semantics based on whether a search is using Basic Regular Expressions (BRE) or Extended Regular Expressions (ERE). Search mechanisms that perform search processing compliant with Perl Compatible Regular Expressions (PCRE) as defined by pcre.org [PCRE] and in Perl 5 [PERLRE] may also

produce matches that differ from matches produced by POSIX compatible regular expression matching. Differences in regular expression matching between POSIX BRE, POSIX ERE and PCRE are illustrated in the examples below, where the "sed" command without the "-E" option is used for POSIX BRE matching, the "sed" command with the "-E" option is used for POSIX ERE matching, and the "perl" command is used for PCRE matching.

```
$ echo 'abcdef' | sed 's/ab(cd)?(cdef)?/[xxxx]/'
abcdef
$ echo 'abcdef' | sed -E 's/ab(cd)?(cdef)?/[xxxx]/'
[xxxx]
$ echo 'abcdef' | perl -p -e 's/ab(cd)?(cdef)?/[xxxx]/'
[xxxx]ef

$ echo 'aaa' | sed 's/a\{3,\}/[xxxx]/'
[xxxx]
$ echo 'aaa' | sed 's/a{3,}/[xxxx]/'
aaa
$ echo 'aaa' | sed -E 's/a\{3,\}/[xxxx]/'
aaa
$ echo 'aaa' | sed -E 's/a{3,}/[xxxx]/'
[xxxx]
$ echo 'aaa' | perl -p -e 's/a\{3,\}/[xxxx]/'
aaa
$ echo 'aaa' | perl -p -e 's/a{3,}/[xxxx]/'
[xxxx]
```

Use of POSIX extended regular expressions is motivated by broad support in the form of API availability [GNU] and database support, with the following major databases supporting POSIX extended regular expressions:

Oracle [ORACLE]
MySQL [MYSQL]
Postgres [POSTGRES]

7. IANA Considerations

FOR DISCUSSION: The URL query parameter "searchtype" with a value of "regex" is specified here-in as syntax for specifying that the RDAP query search pattern is a POSIX extended regular expression. The same approach could be used for specifying future OPTIONAL RDAP search mechanisms. An IANA-maintained registry of RDAP search mechanisms is recommended for recording a list of allowable values for the "searchtype" query parameter.

8. Implementation Status

Note to RFC Editor: Please remove this entire section before publication along with the reference to RFC7942 [RFC7942].

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

8.1. Verisign Labs

Responsible Organization: Verisign Labs
Location: https://rdap.verisignlabs.com/
Description: This implementation includes support for POSIX extended regular expression domain registry RDAP queries using live data from the .cc and .tv country code top-level domains. This implementation also supports federated authentication using OpenID Connect providers as described in [RDAPOPENID]. Three access levels are provided based on the authenticated identity of the client:

1. Unauthenticated: Limited information is returned in response to queries from unauthenticated clients.
2. Basic: Clients who authenticate using a publicly available identity provider like Google Gmail or Microsoft Hotmail will receive all of the information available to an unauthenticated client plus additional registration metadata, but no personally identifiable information associated with entities.
3. Advanced: Clients who authenticate using a more restrictive identity provider will receive all of the information available to a Basic client plus whatever information the server operator deems appropriate for a fully authorized

client. Currently supported identity providers include those developed by Verisign Labs (https://testprovider.rdap.verisignlabs.com/) and CZ.NIC (https://www.mojeid.cz/).
Level of Maturity: This is a "proof of concept" research implementation.
Coverage: This implementation includes all of the features described in this specification.
Contact Information: Swapneel Sheth, ssheth@verisign.com

9. Security Considerations

Security services for the operations specified in this document are described in RFC 7481 [RFC7481].

Search functionality typically requires more server resources (such as memory, CPU cycles, and network bandwidth) when compared to basic lookup functionality. This increases the risk of server resource exhaustion and subsequent denial of service due to abuse. This risk can be mitigated by developing and implementing controls to restrict search functionality to identified and authorized clients. If those clients behave badly, their search privileges can be suspended or revoked. Rate limiting as described in Section 5.5 of RFC 7480 [RFC7480] can also be used to control the rate of received search requests. Server operators can also reduce their risk by restricting the amount of information returned in response to a search request.

Search functionality also increases the privacy risk of disclosing object relationships that might not otherwise be obvious. For example, a search that returns IDN variants [RFC6927] that do not explicitly match a client-provided search pattern can disclose information about registered domain names that might not be otherwise available. Implementers need to consider the policy and privacy implications of returning information that was not explicitly requested.

Note that there might not be a single, static information return policy that applies to all clients equally. Client identity and associated authorizations can be a relevant factor in determining how broad the response set will be for any particular query.

10. Acknowledgements

The author would like to acknowledge the following individuals for their contributions to the development of this document: TBD.

11. References

11.1. Normative References

[IEEE.1003.1_2013_EDITION]
IEEE, "Standard for Information TechnologyPortable Operating System Interface (POSIX(R)) Base Specifications, Issue 7", IEEE 1003.1, 2013 Edition, DOI 10.1109/ieeestd.2013.6506091, April 2013, <http://ieeexplore.ieee.org/servlet/opac?punumber=6506089>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <http://www.rfc-editor.org/info/rfc5890>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", RFC 7480, DOI 10.17487/RFC7480, March 2015, <http://www.rfc-editor.org/info/rfc7480>.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", RFC 7481, DOI 10.17487/RFC7481, March 2015, <http://www.rfc-editor.org/info/rfc7481>.

[RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", RFC 7482, DOI 10.17487/RFC7482, March 2015, <http://www.rfc-editor.org/info/rfc7482>.

[RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, DOI 10.17487/RFC7483, March 2015, <http://www.rfc-editor.org/info/rfc7483>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <http://www.rfc-editor.org/info/rfc7942>.

11.2. Informative References

[GNU] gnu.org, "GNU Regular Expression Matching", <https://www.gnu.org/software/libc/manual/html_node/Regular-Expressions.html>.

[MYSQL] mysql.com, "MySQL Regular Expressions", <http://dev.mysql.com/doc/refman/5.7/en/regexp.html>.

[ORACLE] Oracle Corporation, "Oracle SQL and POSIX Regular Expression Standard", <https://docs.oracle.com/database/121/ADFNS/adfns_regexp.htm#ADFNS231>.

[PCRE] pcre.org, "Perl Compatible Regular Expressions", <http://www.pcre.org/>.

[PERLRE] perl.org, "Perl regular expressions", <http://perldoc.perl.org/perlre.html>.

[POSTGRES] postgresql.org, "PostgreSQL POSIX Regular Expressions", <https://www.postgresql.org/docs/9.3/static/functions-matching.html>.

[RDAPOPENID] ietf.org, "Federated Authentication for the Registration Data Access Protocol (RDAP) using OpenID Connect", <https://tools.ietf.org/html/draft-hollenbeck-regext-rdap-openid-01.txt>.

[RFC1166] Kirkpatrick, S., Stahl, M., and M. Recker, "Internet numbers", RFC 1166, DOI 10.17487/RFC1166, July 1990, <http://www.rfc-editor.org/info/rfc1166>.

[RFC6927] Levine, J. and P. Hoffman, "Variants in Second-Level Names Registered in Top-Level Domains", RFC 6927, DOI 10.17487/RFC6927, May 2013, <http://www.rfc-editor.org/info/rfc6927>.

Appendix A. Change Log

00: Initial version.
01: Renewed and moved invalid Normative References to Informative References
02: Specified use of percent encoding for reserved URL reserved characters in regulard expressions and removed specifcation for base64url encoding for regular expressions

Authors' Addresses

Andrew Fregly
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
USA

Email: afregly@verisign.com
URI: http://www.verisignlabs.com/

Swapneel Sheth
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
USA

Email: ssheth@verisign.com
URI: http://www.verisignlabs.com/

Scott Hollenbeck
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
USA

Email: shollenbeck@verisign.com
URI: http://www.verisignlabs.com/

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 1, 2017

J. Gould
VeriSign, Inc.

March 30, 2017

Data Set File Format
draft-gould-regext-dataset-01

Abstract

This document defines a Data Set File (DSF) format that can be used to define and pass bulk data between a client and a server. This format is extensible to pass any set of simple data types in a set of records contained in the body of the file. The file format also supports storing the result of processing the data set that MAY be generated by the server and returned to the client. The file format consists of an XML definition header and a Comma-Separated Values (CSV) data section delimited by "-----BEGIN DATA SET-----" and "-----END DATA SET-----" lines. The XML definition header defines the format of the CSV data section, contains additional meta-data, and optionally includes a digital signature to identify the source and ensure that the data has not been tampered with between the parties (source, client, and server). The interface (manual or automated) that is used to submit the file and receive the result file is not defined within this document.

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 1, 2017.

Table of Contents

1. Introduction

This document defines a Data Set File (DSF) format that can be used to define and pass bulk data between a client and a server. The client and server MAY leverage a provisioning protocol like EPP, as defined in [RFC5730], to provision individual objects, but the provisioning protocol MAY NOT handle all provisioning use cases. This document defines a format for bulk provisioning requests that can be generated by authorized third parties, can be generated by clients that choose not to leverage the provisioning protocol, and can be supported by servers to process bulk requests in a controlled manner that throttles the processing against the provisioning database. Bulk requests can include court orders, out-of-band client requests to fix data, the backfill of data like contact or verification code data, and a large set of ad-hoc needs.

This format is extensible to pass any set of simple data types in a set of records contained in the body of the file. The file format also supports storing the result of processing the data set that MAY be generated by the server and returned to the client. The file format consists of an XML definition header and a Comma-Separated Values (CSV) data section delimited by "-----BEGIN DATA SET-----" and "-----END DATA SET-----" lines. The XML definition header defines the format of the CSV data section, contains additional meta-data, and optionally includes a digital signature to identify the source and ensure that the data has not been tampered with between the parties (source, client, and server) using XML Signature [W3C.CR-xmldsig-core2-20120124]. All XML Signature [W3C.CR-xmldsig-core2-20120124] data is "base64" encoded, in conformance with [RFC2045], by default to mitigate any validation errors due to whitespace formatting. The interface (manual or automated) that is used to submit the file and receive the result file is not defined within this document and must be documented independently. Please see Section 9 for a description of the issues associated with transport security.

1.1. 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 RFC 2119 [RFC2119].

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

The following XML abbreviations are used:

"eppcom" XML namespace prefix refers to "urn:ietf:params:xml:ns:eppcom-1.0" XML namespace in [RFC5730].
"domain" XML namespace prefix refers to "urn:ietf:params:xml:ns:domain-1.0" XML namespace in [RFC5731].
"host" XML namespace prefix refers to "urn:ietf:params:xml:ns:host-1.0" XML namespace in [RFC5732].
"contact" XML namespace prefix refers to "urn:ietf:params:xml:ns:contact-1.0" XML namespace in [RFC5733].
"dataSet-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dataSet-1.0".
"dataSet" XML namespace prefix refers to "urn:ietf:params:xml:ns:dataSet-1.0" XML namespace.
"dsfDomain-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dsfDomain-1.0".
"dsfDomain" XML namespace prefix refers to "urn:ietf:params:xml:ns:dsfDomain-1.0" XML namespace.
"dsfHost-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dsfHost-1.0".
"dsfHost" XML namespace prefix refers to "urn:ietf:params:xml:ns:dsfHost-1.0" XML namespace.
"dsfContact-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dsfContact-1.0".
"dsfContact" XML namespace prefix refers to "urn:ietf:params:xml:ns:dsfContact-1.0" XML namespace.
"dsfVerificationCode-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dsfVerificationCode-1.0".
"dsfVerificationCode" XML namespace prefix refers to "urn:ietf:params:xml:ns:dsfVerificationCode-1.0" XML namespace.
"dsfRouting-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dsfRouting-1.0".

Implementations MUST NOT depend on the XML namespace prefix used in this document and instead MUST employ a proper namespace-aware XML parser and serializer to interpret and output the XML.

2. General Conventions

This document includes a general set of attributes and terms that are described here.

2.1. Date and Time

Numerous fields indicate "dates", such as the creation date of the DSF. These fields SHALL contain timestamps indicating the date and time in UTC as specified in [RFC3339], with no offset from the zero meridian.

2.2. Checksum

The checksum of the body of the file, as defined by the body rule of the Data Set File (DSF) ABNF (Section 3), MUST use CRC32, that is the algorithm used in the ISO 13239 standard [iso13239] and in section 8.1.1.6.2 of ITU-T recommendation V.42 [itu-t-V.42].

2.3. Type and Subtype

There can be many different Data Set File (DSF) types supported based on the objects, operations, and the specific scenarios. To make it easier to negotiate the client's intent of the DSF with the server's set of supported DSF files, the DSF supports the definition of a type and an optional sub-type. The supported list of types and sub-types is up to server policy. The server SHOULD provide the possible set of supported DSF type and sub-type values to the parties generating the DSF. The mechanism for providing the supported DSF type and sub-type values is not defined in this document but MAY require registration within an IANA registry.

The <dataSet:type> element formally defines the DSF type, which indicates the expected set of fields defined under the <dataSet:fields> element and the expected operation to be taken. An OPTIONAL "subType" attribute defines the sub-type that is used to further refine the purpose of the DSF.

It is recommended that the server follows a consistent naming scheme for the <dataSet:type> values. One option is to delimit the value with a '.' and to include the object ("domain", "host", "contact, "verificationCode", etc.), the operation ("create", "renew, "transfer", "delete", "update", etc.), and the scoped name of the DSF. For example, to support the bulk update of encoded signed verification codes, the <dataSet:type> value can be set to "verificationCode.update.encodedSignedCode". To further sub-divide the "verificationCode.update.encodedSignedCode" type by locality, the "subType" attribute can be set to the locality name. For example,

the "china" verification profile supports two verification codes of type "domain" and "real-name", so the "verificationCode.update.encodedSignedCode" DSF type can include "china" for the "subType" attribute, as in <dataSet:type subType="china">verificationCode.update.encodedSignedCode</dataSet:type>.

2.4. Fields

The <dataSet:fields> element, an element in the header (Section 3.1), contains an ordered list of CSV fields used in the body of the file delimited by the character defined by the OPTIONAL "sep" attribute, with a default value of ",". A field child element substitutes for the <dataSet:field> abstract element. Each element defines the format of the CSV field contained in the body. The <dataSet:field> elements support the "type" attribute that defines the XML schema simple type of the field element.

The abstract <dataSet:field> element does not directly define any attributes, but there are a couple attributes that a data set processor SHOULD support including:

isRequired When the "isRequired" attribute is "true", it indicates that the field MUST be non-empty in the body and when set to "false", it indicates that the field MAY be empty in the body. The "isRequired" attribute MAY be specifically set for the field elements in the XML schema and MAY be overriden by specifying the attribute in the fields under the <dataSet:fields> element. If no "isRequired" attribute is defined, the default value is "false".

isPrimaryKey When the "isPrimaryKey" attribute is "true", it indicates that the field is part of the primary key of the records. The combination of the primary key fields MUST be unique across the set of records in the file. The "isPrimaryKey" attribute MAY be specifically set for the field elements in the XML schema and MAY be overridden by specifying the attribute in the fields under the <dataSet:fields> element. If no "isPrimaryKey" attribute is defined, the default value is "false".

2.4.1. Base Field Types

New field types can be defined that reside in the CSV records. To make the definition of new field types easier, a set of base field types are defined in the dataSet-1.0 XML schema that include:

dataSet:fieldOptionalType The "dataSet:fieldOptionalType" type can be extended by a field that can be empty ("isRequired" = "false")

and is not a member of the primary key ("isPrimaryKey" = "false").

dataSet:fieldRequiredType The "dataSet:fieldRequiredType" type can be extended by a field that cannot be empty ("isRequired" = "true") and is not a member of the primary key ("isPrimaryKey" = "false").

dataSet:fieldPrimaryKeyType The "dataSet:fieldPrimaryKeyType" type can be extended by a field that cannot be empty ("isRequired" = "true") and is a member of the primary key ("isPrimaryKey" = "true").

dataSet:fListItemType The "dataSet:fListItemType" type is an extension of the "dataSet:fieldOptionalType" type that adds an OPTIONAL "op" attribute that reflects the operation to apply for the list item. The default "op" value is "replace". The list of possible "op" values is include below. A "dataSet:fListItemType" field with "op" set to "replace" MUST NOT be mixed with a matching "dataSet:fListItemType" field with "op" set to either "add" or "remove".

"replace" Specifies that the list item will replace was is currently set for the object. The list of list items provided will replace the existing list of items set on the object. The concrete "dataSet:fListItemType" field along with the DSF type can determine what is the possible set of list item values.

"add" Specifies that the list item should be added to the object if it is not already set.

"rem" Specifies that the list item should be removed from the object and expects it to already be set.

Example definition of the <dataSet:fName> field element that extends the "dataSet:fieldPrimaryKeyType" base field type with the XML type dataSet:labelType and with the required "class" attribute:

```
<element name="fName" type="dataSet:fNameType"
  substitutionGroup="dataSet:field"/>

<complexType name="fNameType">
  <complexContent>
    <extension base="dataSet:fieldPrimaryKeyType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:labelType"/>
      <attribute name="class" type="token"
        use="required"/>
    </extension>
  </complexContent>
</complexType>
```

2.4.2. Data Set Field Elements

The dataSet-1.0 XML schema predefines a set of field elements that can be used as child elements of the <dataSet:fields> element of the header to define the CSV record fields. The dataSet-1.0 field elements with the XML schema type value and the base type include:

<dataSet:fName> Field that represents the name of an object. This field extends a "dataSet:fieldPrimaryKeyType", as defined in Section 2.4.1, with the type="token". The required "class" attribute defines the class of the object, like the use of "domain" for a domain name. It is recommended to use an object specific name field if available, like <dsfDomain:fName>.
<dataSet:fAuthInfo> Object authorization information with type="eppcom:pwAuthInfoType".
<dataSet:fResultCode> Field that represents the result of processing an individual record using the codes defined in Section 5. The field extends a "dataSet:fieldRequiredType", as defined in Section 2.4.1, with the type="dataSet:resultCodeType".
<dataSet:fResultMsg> Field containing the human-readable description of the result code. The field extends a "dataSet:fieldOptionalType", as defined in Section 2.4.1, with the type="normalizedString". The field includes the OPTIONAL "lang" attribute with the default value of "en" (English).
<dataSet:fResultReason> Field containing the human-readable message that describes the reason for the error processing the record. The field extends the "dataSet:fieldOptionalType", as defined in Section 2.4.1, with the type="normalizedString". The language of the reason is identified the OPTIONAL "lang" attribute. If not specified, the default attribute value is "en" (English).

Example <dataSet:fields> definition for a result file generated by a server that includes a domain name with a result code, with an OPTIONAL result message, and an OPTIONAL result reason:

```
<dataSet:fields>
  <dataSet:fName class="domain"/>
  <dataSet:fResultCode/>
  <dataSet:fResultMsg/>
  <dataSet:fResultReason/>
</dataSet:fields>
```

2.4.2.1. Field Element Extensibility

New field elements MAY be defined in separate XML schemas and referenced as child elements of the <dataSet:fields> element. The convention is to prefix all field elements with a lowercase "f", as in <dataSet:fName> for the object name or <dataSet:fResultCode> for

the result code. The following are the steps to define a new field element:

1. Define a new XML schema or extend an existing XML schema to include the definition of the field element. The XML schema will have it's own XML namespace that will be referenced in the header of the Data Set File (DSF).
2. Define a new XML schema complexType that is a direct or indirect extension of the "dataSet:fieldType" complexType. The new complexType MUST directly, or indirectly define through extension, the following attibutes:

   "type" Defines the XML schema data type of the CSV field. The "default" value of the attribute is used to define the default CSV field type using an XML schema simple data type. Any XML namespaces MUST be escaped ("\:" instead of ":") so the XML parser of the header can ignore the reference during XML validation. The "type" attribute is used during CSV validation leveraging the XML schema simple data type to define the format. The "type" attribute can be overridden when referencing the field element in the header. The "type" attribute for the "dataSet:fResultCodeType" complexType is defined as <attribute name="type" default="dataSet\:resultCodeType">. When using the default XML namespace, no escaping is necessary. For example for the "dataSet:fResultMsg" complexType, the "type" attribute is defined as <attribute name="type" default="normalizedString"/>. A validator of the Data Set File (DSF) will apply the XML schema type to the CSV fields.

   "isRequired" See the definition of the "isRequired" attribute in Section 2.4. A required field can extend the "dataSet:fieldRequiredType" complexType, as defined in Section 2.4.1, or can explicitly specify the attribute. For example, the "isRequired" attribute is set in the "dataSet:fieldRequiredType" complexType as <attribute name="isRequired" default="true">.

   "isPrimaryKey" See the definition of the "isPrimaryKey" attribute in Section 2.4. A primary key field can extend the "dataSet:primaryKeyType" complexType, as defined in Section 2.4.1, or can explicitly specify the attribute. For example, the "isPrimaryKey" attribute is set in the "dataSet:fieldPrimaryKeyType" complexType as <attribute name="isPrimaryKey" default="true"/>.

3. Define the field element that uses the new complexType defined above and that substitutes for the "dataSet:field" abstract element. An example of defining the "dataSet:fResultCode" field element is <element name="fResultCode" type="dataSet:fResultCodeType" substituteGroup="dataSet:field"/>.

An example of defining the "dataSet:fResultReason" field element is <element name="fResultReason" type="dataSet:fResultMsgType" subsititionGroup="dataSet:field"/>.

Example definition of the <dataSet:fResultCode> field element that is required to be non-empty and with the CSV field type "dataSet:resultCodeType":

```
 <element name="fResultCode"
   type="dataSet:fResultCodeType"
   substitutionGroup="dataSet:field"/>

 <complexType name="fResultCodeType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:resultCodeType"/>
    </extension>
  </complexContent>
 </complexType>
```

Example definition of the <dataSet:fResultMsg> field element that may be empty (optional), with the CSV field type "normalizedString", and with the additional optional "lang" attribute:

```
 <element name="fResultMsg"
   type="dataSet:fResultMsgType"
   substitutionGroup="dataSet:field"/>

 <complexType name="fResultMsgType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="normalizedString"/>
      <attribute name="lang" type="language"
        default="en"/>
    </extension>
  </complexContent>
 </complexType>
```

2.5. Routing

A Data Set File (DSF) MAY include data that targets multiple independent backend services, but the object alone cannot be used to determine the appropropriate backend service. An example is creating a Contact Object (Section 4.3) in the .EXAMPLE1 domain registry

service and creating another Contact Object (Section 4.3) in the .EXAMPLE2 domain registry service within a single DSF, where .EXAMPLE1 and .EXAMPLE2 are independent domain registry services. The Data Set File (DSF) processor MAY coordinate with the backend services to process each of the DSF records, but it is dependent on the client to specify the target backend service with each record.

The Data Set File (DSF) field element used for routing a record to a specific backend service includes:

<dsfRouting:fSubProduct> Contains the unique sub-product name of a backend service with type="token". It is up to server policy on the valid list of sub-product values. An example of a sub-product value is the Top Level Domain (TLD) of a domain registry service.

Routing is most applicable to objects like a Contact Object (Section 4.3) and a Host Object (Section 4.2), where the keys (<dsfContact:fId> for the Contact Object and <dsfHost:fName> for the Host Object) MAY NOT be enough to uniqually identify a target domain registry service. An example of using the <dsfRouting:fSubProduct> field for routing is provided with the "contact.create.routing" DSF in Section 4.3.

3. Data Set File (DSF) Format

The Data Set File (DSF) format supports multiple types of requests and response files. All of these types of files support a general file format that includes a header that provides meta-data about the data including the what, who, when, and optionally the digital signature of the information, and the body containing the data. The Data Set File (DSF) syntax is specified using Augmented Backus-Naur Form (ABNF) grammar [RFC5234] as follows:

Data Set File (DSF) ABNF

```
file         = header body
header       = dataSet-1-0 LF
dataSet-1-0  = 1*(1*VCHAR LF) ; compliant to dataSet-1.0
body         = start [data] end
start        = "-----BEGIN DATA SET-----" LF
data         = 1*VCHAR LF ; CSV compliant <dataSet:fields>
end          = "-----END DATA SET-----" *1LF
```

3.1. Header Format

The header of the Data Set File (DSF) is an XML document that complies with the dataSet-1.0 XML schema. The purpose of the header is to provide the meta-data about the data contained in the body. The <dataSet:definition> element is the root XML element of the header and contains the following child element:

<dataSet:defData> ; or <dataSet:encodedSignedDefData> or <dataSet:resultData>

- <dataSet:defData> Provides meta-data about the body of the file that is not digitally signed, as described in Section 3.1.1.
- <dataSet:encodedSignedDefData> Contains the encoded form of the digitally signed <dataSet:signedDefData> element, as described in in Section 3.1.2.
- <dataSet:resultData> Provides the high level result data with the processing of a request Data Set File (DSF) by the server, as described in Section 3.1.3.

3.1.1. <dataSet:defData> element

The <dataSet:defData> element contains the meta-data of the body of the file that is not digitally signed. The <dataSet:defData> element contains the following child elements:

<dataSet:type> The type value and OPTIONAL "subType" attribute value of the Data Set File (DSF), as defined in Section 2.3.
<dataSet:fields> Ordered list of CSV fields as defined in Section 2.4.
<dataSet:dataSetId> OPTIONAL unique identifier of the Data Set File (DSF). The client SHOULD assign a unique identifer when generating the Data Set File (DSF) and set it in the <dataSet:dataSetId> element.
<dataSet:crDate> Contains the date and time of the Data Set File (DSF) creation.

Example <dataSet:defData> element for setting verification codes on a domain name:

```
<dataSet:defData>
  <dataSet:fields>
    <dataSet:type subType="locality">
      verificationCode.update.encodedSignedCode
    </dataSet:type>
    <dsfDomain:fName/>
    <dsfVerificationCode:fEncodedSignedCode type="domain"/>
    <dsfVerificationCode:fEncodedSignedCode type="real-name"/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
```

3.1.2. <dataSet:encodedSignedDefData> element

The <dataSet:encodedSignedDefData> element contains the encoded form of the digitally signed <dataSet:signedDefData> element, described in Section 3.1.2.1, with the encoding defined by the "encoding" attribute with the default "encoding" value of "base64". The "base64" encoded text of the <dataSet:signedDefData> element MUST conform to [RFC2045].

Example <dataSet:encodedSignedDefData> element, where "..." represents continuation of data for brevity:

```
<dataSet:encodedSignedDefData encoding="base64">
  ICAgICAgPHZlcmlmaWNhdGlvbkNvZGU6c2lnbmVkQ29kZQogICAgICAgIHhtbG5z
  ...
  b25Db2RlOnNpZ25lZENvZGU+Cg==
</dataSet:encodedSignedDefData>
```

3.1.2.1. Signed Definition Data

The Signed Definition Data, represented by the <dataSet:signedDefData> element, is a signed extension of the <dataSet:defData> element, defined in Section 3.1.1. The <dataSet:signedDefData> provides the meta-data about the body of the file, that is a fragment of XML that is digitally signed using XML Signature [W3C.CR-xmldsig-core2-20120124]. The <dataSet:signedDefData> element includes a required "id" attribute of type XSD ID for use with the IDREF URI from the Signature element. Setting the "id" attribute value to "signedData" is recommended. The certificate of the issuer MUST be included with the Signature so it can be chained with the issuer’s certificate by the validating client. A checksum, as defined by the Section 2.2, of the body of

the file, as defined by the body rule of the Data Set File (DSF) ABNF, is included in the digitally signed data to ensure that it's covered by the Signature. The <dataSet:signedDefData> element contains the following child elements:

<dataSet:type> The type value and OPTIONAL "subType" attribute value of the Data Set File (DSF), as defined in Section 2.3.
<dataSet:fields> Ordered list of CSV fields as defined in Section 2.4.
<dataSet:dataSetId> OPTIONAL unique identifier of the data set. The source SHOULD assign a unique identifer when generating the data set and set it in the <dataSet:dataSetId> element.
<dataSet:crDate> Contains the date and time of the data set creation.
<dataSet:cksum> Checksum of the body of the file, as defined by the body rule of the Data Set File (DSF) ABNF, using the mechanism defined by the Section 2.2.
<Signature> XML Signature [W3C.CR-xmldsig-core2-20120124] for the <dataSet:signedDefData>. Use of a namespace prefix, like "dsig", is recommended for the XML Signature [W3C.CR-xmldsig-core2-20120124] elements.

Example <dataSet:signedDefData> element, where "..." represents continuation of data for brevity:

```
<dataSet:signedDefData
    xmlns:dataSet=
      "urn:ietf:params:xml:ns:dataSet-1.0"
    xmlns:dsfDomain=
      "urn:ietf:params:xml:ns:dsfDomain-1.0"
    xmlns:dsfVerificationCode=
      "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
    id="signedData">
    <dataSet:type subType="locality">
      verificationCode.update.encodedSignedCode
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfVerificationCode:fCode type="domain"/>
      <dsfVerificationCode:fCode type="real-name"/>
    </dataSet:fields>
     <dataSet:dataSetId>abc-123</dataSet:dataSetId>
     <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
     <dataSet:cksum>6F2B988F</dataSet:cksum>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
    <CanonicalizationMethod
 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

```
      <SignatureMethod
    Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
      <Reference URI="#signedData">
       <Transforms>
        <Transform
    Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
       </Transforms>
       <DigestMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <DigestValue>wgyW3nZPoEfpptlhRILKnOQnbdtU6ArM7ShrAfHgDFg=
    </DigestValue>
      </Reference>
     </SignedInfo>
     <SignatureValue>
    jMu4PfyQGiJBF0GWSEPFCJjmywCEqR2h4LD+ge6XQ+JnmKFFCuCZS/3SLKAx0L1w
    ...
    ipJsXNa6osTUw1CzA7jfwA==
     </SignatureValue>
     <KeyInfo>
      <X509Data>
      <X509Certificate>
    MIIESTCCAzGgAwIBAgIBAjANBgkqhkiG9w0BAQsFADBiMQswCQYDVQQGEwJVUzEL
    ...
    AdXitTWFipaIGea9lEGFM0L9+Bg7XzNn4nVLXokyEB3bgS4scG6QznX23FGk
     </X509Certificate>
     </X509Data>
     </KeyInfo>
    </Signature>
  </dataSet:signedDefData>
```

3.1.3. <dataSet:resultData> element

The <dataSet:resultData> element contains the high level result data with the processing of a request Data Set File (DSF) by the server. the "code" attribute describes the success or failure of the processing using the code values defined in Section 5. The <dataSet:resultData> element contains the following child elements:

<dataSet:type> OPTIONAL type value and OPTIONAL "subType" attribute value associaed with the request Data Set File (DSF), as defined in Section 2.3. The <dataSet:type> element MUST be returned if the request header is successfully parsed.

<dataSet:fields> OPTIONAL ordered list of CSV fields as defined in Section 2.4. If the <dataSet:fields> element is not defined, then the data as defined by the data rule of the Data Set File (DSF) ABNF MUST be empty.

<dataSet:dataSetId> OPTIONAL identifier of the Data Set File (DSF) formed using the <dataSet:dataSetId> element associated with the

request if supplied by the client and the request header is successfully parsed.

<dataSet:msg> Human-readable description of the result code. The language of the result is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value is "en" (English).

<dataSet:reason> OPTIONAL human-readable message that describes the reason for the error. The language of the reason is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value is "en" (English).

<dataSet:records> OPTIONAL summary record result data. The summary data is associated with the count of the records (lines in request data). The <dataSet:records> element contains the following child elements:

<dataSet:total> Total count of the records (lines in request data) processed by the server.

<dataSet:success> Total count of the records (lines in request data) successfully processed by the server.

<dataSet:failed> Total count of the records (lines in request data) that failed processing by the server.

Example successful <dataSet:resultData> element:

```
<dataSet:resultData code="1000">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:fields>
    <dsfDomain:fName/>
    <dataSet:fResultCode/>
    <dataSet:fResultMsg/>
    <dataSet:fResultReason/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Code Set processed successfully.</dataSet:msg>
  <dataSet:records>
    <dataSet:total>2</dataSet:total>
    <dataSet:success>2</dataSet:success>
    <dataSet:failed>0</dataSet:failed>
  </dataSet:records>
</dataSet:resultData>
```

Example <dataSet:resultData> element with some failures:

```
<dataSet:resultData code="1001">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:fields>
    <dsfDomain:fName/>
    <dataSet:fResultCode/>
    <dataSet:fResultMsg/>
    <dataSet:fResultReason/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Success with failures</dataSet:msg>
  <dataSet:records>
    <dataSet:total>4</dataSet:total>
    <dataSet:success>1</dataSet:success>
    <dataSet:failed>3</dataSet:failed>
  </dataSet:records>
</dataSet:resultData>
```

Example failed <dataSet:resultData> element based on malformed request file:

```
<dataSet:resultData code="2000">
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>File syntax error</dataSet:msg>
  <dataSet:reason lang="en">Malformed request file
  </dataSet:reason>
</dataSet:resultData>
```

Example failed <dataSet:resultData> element based on header syntax error:

```
<dataSet:resultData code="2001">
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Header syntax error</dataSet:msg>
  <dataSet:reason lang="en">Header XML syntax error
  </dataSet:reason>
</dataSet:resultData>
```

Example failed <dataSet:resultData> element based on body syntax error:

```
<dataSet:resultData code="2002">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Body syntax error</dataSet:msg>
  <dataSet:reason lang="en">Invalid with fields definition
  </dataSet:reason>
</dataSet:resultData>
```

3.2. Body Format

The body of the Data Set File (DSF) is a set of Comma-Separated Values (CSV) data that includes a leading "-----BEGIN CODE SET-----" line and a trailing "-----END CODE SET-----" line. The format of the CSV data is defined by the <dataSet:fields> element in the header (Section 3.1), that includes the delimiter and the ordered set of fields (Section 2.4) with their XML simple type and descriptor attributes like "isRequired", "isPrimaryKey", "class", and "codeType".

Example body for a result file generated by a server that includes a success and a set of failed records with a mix of messages and reasons.:

```
-----BEGIN DATA SET-----
domain1.example,1000,Success,
domain2.example,2303,Object does not exist,
domain3.example,2201,Authorization error,
domain4.example,2302,Object exists,domain code already
-----END DATA SET-----
```

4. Object Description

This section describes the base objects supported by this specification:

4.1. Domain Name Object

The domain name object is based on the EPP domain name mapping specified in [RFC5731].

The Data Set File (DSF) field elements specific to the domain name object include:

<dsfDomain:fName> Domain name field with type="eppcom:labelType" and isRequired="true".
<dsfDomain:fPeriod> Domain name initial or extension period of the expiration date with type="domain:pLimitType".
<dsfDomain:fPeriodUnit> Domain name initial or extension period unit of the expiration date with type="domain:pUnitType".
<dsfDomain:fNs> Domain name server is an extension of the "dataSet:fListItemType" field, described in Section 2.4.1, with type="eppcom:labelType".
<dsfDomain:fContact> Domain contact identifer with type="eppcom:clIDType" and with required "role" attribute. The possible values of the "role" attribute include "registrant", "admin", "tech", and "billing".
<dsfDomain:fStatus> The status of the domain name is an extension of the "dataSet:fListItemType" field, described in Section 2.4.1, with type="domain:statusValueType".
<dsfDomain:fKeyTag> Contains the DS key tag value per [RFC5910] with type="unsignedShort".
<dsfDomain:fDsAlg> Contains the DS algorithm value per [RFC5910] with type="unsignedByte".
<dsfDomain:fDigestType> Contains the DS digest type value per [RFC5910] with type="unsignedByte".
<dsfDomain:fDigest> Contains the DS digest value per [RFC5910] with type="hexBinary".
<dsfDomain:fFlags> Contains the flags field value per [RFC5910] with type="unsignedShort".
<dsfDomain:fProtocol> Contains the Key protocol value per [RFC5910] with type="unsignedByte".
<dsfDomain:fKeyAlg> Contains the Key algorithm value per [RFC5910] with type="unsignedByte".
<dsfDomain:fPubKey> Contains the public key value per [RFC5910] with type="secDNS:keyType".
<dsfDomain:fMaxSigLife> Indicates a child’s preference for the number of seconds after signature generation when the parent’s signature on the DS information provided by the child will expire with type="secDNS:maxSigLifeType" defined in [RFC5910].

The following are example DSF files using the domain object fields. The different forms of domain object DSF files is up to server policy.

Example of a "domain.update.replaceClientStatuses" DSF that will set the client statuses of a domain name to those specified in the DSF record. The client statuses set will be replaced by the set of client statuses specified. Empty statuses will result in the removal of those statuses if previously set. Non-empty statuses will be added if not previously set. Non-empty statuses will result in no change if previously set. All five client statuses defined in [RFC5731] are defined in the DSF in fixed order.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
  "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.replaceClientStatuses
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <!-- clientDeleteProhibited -->
      <dsfDomain:fStatus/>
      <!-- clientHold -->
      <dsfDomain:fStatus/>
      <!-- clientRenewProhibited -->
      <dsfDomain:fStatus/>
      <!-- clientTransferProhibited -->
      <dsfDomain:fStatus/>
      <!-- clientUpdateProhibited -->
      <dsfDomain:fStatus/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,clientDeleteProhibited,,,,clientUpdateProhibited
domain2.example,,clientHold,,,
domain3.example,,,clientRenewProhibited,clientTransferProhibited,
domain4.example,,,,,
-----END DATA SET-----
```

Example of a "domain.update.addRemoveStatus" DSF that will add a status and remove a status for each domain name. Any status can be empty to indicate no action for that domain name.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
  "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.addRemoveNs
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fNs op="add"/>
      <dsfDomain:fNs op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ns1.domain1.example,n2.domain1.example
domain2.example,ns1.domain1.example,
domain3.example,,ns2.domain1.example
-----END DATA SET-----
```

Example of a "domain.update.replaceNs" DSF that will set the name servers of a domain name to those specified in the DSF record. The server supports setting up to 13 name servers to a domain name, so there are 13 <dsfDomain:fNs> defined. All existing name servers set will be replaced with the name servers specified.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
  "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.replaceNs
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ns1.domain1.example,n2.domain1.example,,,,,,,,,,
domain2.example,ns1.domain1.example,,,,,,,,,,,,
domain3.example,,,,,,,,,,,,
-----END DATA SET-----
```

Example of a "domain.update.addRemoveNs" DSF that will add a name server and remove a name server for each domain name. Any name server can be empty to indicate no action for that domain name.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
  "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.addRemoveNs
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fNs op="add"/>
      <dsfDomain:fNs op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ns1.domain1.example,n2.domain1.example
domain2.example,ns1.domain1.example,
domain3.example,,ns2.domain1.example
-----END DATA SET-----
```

Example of a "domain.update.contacts" DSF that supports updating the contacts (registrant, admin, tech, and billing) for each domain name. All four contacts fields are set as required using the "isRequired" attribute and any existing contacts set for each domain name are replaced.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
  "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.contacts
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fContact role="registrant" isRequired="true"/>
      <dsfDomain:fContact role="admin" isRequired="true"/>
      <dsfDomain:fContact role="tech" isRequired="true"/>
      <dsfDomain:fContact role="billing" isRequired="false"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,jd1234,sh813,sh813,sh813
domain2.example,jd1234,sh813,sh813,
-----END DATA SET-----
```

Example of a "domain.create.standard" DSF that supports creating each domain name record with a standard set of fields / attributes for a thick domain name.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
  "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.create.standard
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fPeriod/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fContact role="registrant"/>
      <dsfDomain:fContact role="admin"/>
      <dsfDomain:fContact role="tech"/>
      <dsfDomain:fContact role="billing"/>
      <dataSet:fAuthInfo/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,1,ns1.dom.example,,jd1234,sh813,sh813,sh813,2fooBAR
domain2.example,1,,,jd1234,sh813,sh813,sh813,2fooBAR
-----END DATA SET-----
```

4.2. Host Object

The host object is based on the EPP host mapping specified in [RFC5732].

The Data Set File (DSF) field elements specific to the host object include:

<dsfHost:fName> Host name field with type="eppcom:labelType" and isRequired="true".
<dsfHost:fNewName> Host new name field used to rename the host with type="eppcom:labelType".
<dsfHost:fAddrVersion> The address version ("v4" or "v6") is an extension of the "dataSet:fListItemType" field, described in

Section 2.4.1, with type="eppcom:addrStringType". The "dsfHost.fAddrVersion" field usually corresponds with a following "dsfHost:fAddr" field. The "dsfHost:fAddrVersion" and "dsfHost.fAddr" fields are included in pairs.

<dsfHost:fAddr> The address is an extension of the "dataSet:fListItemType" field, described in Section 2.4.1, with type="eppcom:addrStringType". The "dsfHost.fAddr" field usually corresponds with a previous "dsfHost:fAddrVersion" field that defines the type of address. The "dsfHost:fAddrVersion" and "dsfHost.fAddr" fields are included in pairs.

<dsfHost:fStatus> The status of the host is an extension of the "dataSet:fListItemType" field, described in Section 2.4.1, with type="host:statusValueType".

The following are example DSF files using the host object fields. The different forms of host object DSF files is up to server policy.

Example of a "host.update.replaceClientStatuses" DSF that will set the client statuses of a host to those specified in the DSF record. The client statuses set will be replaced by the set of client statuses specified. Empty statuses will result in the removal of those statuses if previously set. Non-empty statuses will be added if not previously set. Non-empty statuses will result in no change if previously set. All two client statuses defined in [RFC5732] are defined in the DSF in fixed order.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
  "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.replaceClientStatuses
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <!-- clientDeleteProhibited -->
      <dsfHost:fStatus/>
      <!-- clientUpdateProhibited -->
      <dsfHost:fStatus/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,clientDeleteProhibited,clientUpdateProhibited
ns2.domain.example,,clientUpdateProhibited
ns3.domain.example,clientDeleteProhibited,
ns4.domain.example,,
-----END DATA SET-----
```

Example of a "host.update.addRemoveStatus" DSF that will add a status and remove a status for each host. Any status can be empty to indicate no action for that host.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
  "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.addRemoveClientStatuses
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fStatus op="add"/>
      <dsfHost:fStatus op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,clientDeleteProhibited,clientUpdateProhibited
ns2.domain.example,,clientUpdateProhibited
ns3.domain.example,clientUpdateProhibited,
-----END DATA SET-----
```

Example of a "host.update.replaceAddr" DSF that will set the addresses of a host to those specified in the DSF record. The server supports setting up to 6 addresses to a host, so there are 6 <dsfHost:fAddrVersion> and <dsfHost:fAddr> field pairs defined. All existing addresses set will be replaced with the addresses specified.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
  "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.replaceAddr
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,v4,192.0.2.2,v4,192.0.2.29,,,,,,,
ns2.domain.example,v6,1080:0:0:0:8:800:200C:417A,,,,,,,,,
ns3.domain.example,,,,,,,,,,,
-----END DATA SET-----
```

Example of a "host.update.addRemoveAddr" DSF that will add an address and remove an address for each host. Any address can be empty to indicate no action for that host. The <dsfHost:fAddrVersion> and <dsfHost:fAddr> field pairs are provided with matching list operations.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
  "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.addRemoveAddr
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fAddrVersion op="add"/>
      <dsfHost:fAddr op="add"/>
      <dsfHost:fAddrVersion op="remove"/>
      <dsfHost:fAddr op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,v4,192.0.2.2,v4,192.0.2.29
ns2.domain.example,v6,1080:0:0:0:8:800:200C:417A,,
ns3.domain.example,,v4,192.0.2.29
-----END DATA SET-----
```

Example of a "host.create.standard" DSF that supports creating each host record with a standard set of fields / attributes for a host. There is an example of creating an internal host (ns1.domain.example and ns2.domain.example) with up to two addresses and an external host (ns1.domain.external) without addresses.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
  "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.create.standard
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,v4,192.0.2.2,v4,192.0.2.29
ns2.domain.example,v6,1080:0:0:0:8:800:200C:417A,
ns1.domain.external,,,,
-----END DATA SET-----
```

4.3. Contact Object

The contact object is based on the EPP host mapping specified in [RFC5733].

Some elements MAY be provided in either internationalized form ("int") or provided in localized form ("loc"). Those elements use a field value or "isLoc" attribute to specify the form used. If an "isLoc" attribute is used, a value of "true" indicates the use of the localized form and a value of "false" indicates the use of the internationalized form. This MAY override the form specified for a parent element. A value of "int" is used to indicate the internationalized form and a value of "loc" is used to indicate the localized form. When the internalized form ("int") is provided, the field value MUST be represented in a subset of UTF-8 that can be

represented in the 7-bit US-ASCII character set. When the localized form ("loc") is provided, the field value MAY be represented in unrestricted UTF-8. Some of the contact field elements specify the internationalized form with the isLoc="false" attribute.

The Data Set File (DSF) field elements specific to the contact object include:

<dsfContact:fId> Contains the server-unique contact identifier with type="eppcom:clIDType" and isRequired="true".
<dsfContact:fEmail> Contains the contact's email address with type="eppcom:minTokenType" and isRequired="true".
<dsfContact:fVoice> Contains the contact's voice telephone number with type="contact:e164StringType".
<dsfContact:fVoiceExt> Contains the contact's voice telephone number extension with type="token".
<dsfContact:fFax> Contains the contact's facsimile telephone number with type="contact:e164StringType".
<dsfContact:fFaxExt> Contains the contact's facsimile telephone number extension with type="token".
<dsfContact:fName> Contains the contact's name of the individual or role represented by the contact with type="contact:postalLineType" and isRequired="true". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.
<dsfContact:fStreet> Contains the contact's contact's street address line with type="contact:fPostalLineType". An index attribute is required to indicate which street address line the field represents with index "0" for the first line and index "2" for the last line. An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.
<dsfContact:fCity> Contains the contact's city with type="contact:postalLineType" and isRequired="true". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.
<dsfContact:fCc> Contains the contact's country code with type="contact:ccType" and isRequired="true". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.
<dsfContact:fPostalType> Contains the form of the postal-address information with type="contact:postalLineType". This field specifies the form ("int" or "loc") of the <dsfContact:fName>, <dsfContact:fOrg>, <dsfContact:fStreet>, <dsfContact:fCity>, <dsfContact:fSp>, <dsfContact:fPc>, <dsfContact:fCc> fields.
<dsfContact:fOrg> Contains the name of the organization with which the contact is affiliated with type="contact:optPostalLineType". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fSp> Contains the contact's state or province with type="contact:optPostalLineType". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.
<dsfContact:fPc> Contains the contact's postal code with type="contact:pcType". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.
<dsfContact:fDiscloseFlag> Contains flag with a value of "true" or "1" (one) notes the preference to allow disclosure of the specified elements as an exception to the stated data-collection policy. A value of "false" or "0" (zero) notes a client preference to not allow disclosure of the specified elements as an exception to the stated data-collection policy with type="boolean". The additional fields define specific exceptional disclosure preferences based on the <dsfContact:fDiscloseFlag> field.
<dsfContact:fDiscloseNameLoc> Exceptional disclosure preference flag for the localized form of the contact name with type="boolean".
<dsfContact:fDiscloseNameInt> Exceptional disclosure preference flag for the internationalized form of the contact name with type="boolean".
<dsfContact:fDiscloseOrgLoc> Exceptional disclosure preference flag for the localized form of the contact organization with type="boolean". with type="boolean".
<dsfContact:fDiscloseOrgInt> Exceptional disclosure preference flag for the internationalized form of the contact organization with type="boolean". with type="boolean".
<dsfContact:fDiscloseAddrLoc> Exceptional disclosure preference flag for the localized form of the contact address with type="boolean".
<dsfContact:fDiscloseAddrInt> Exceptional disclosure preference flag for the internationalized form of the contact address with type="boolean".
<dsfContact:fDiscloseVoice> Exceptional disclosure preference flag of the contact voice telephone number with type="boolean".
<dsfContact:fDiscloseFax> Exceptional disclosure preference flag of the contact facsimile telephone number with type="boolean".
<dsfContact:fDiscloseEmail> Exceptional disclosure preference flag of the contact email address with type="boolean".
<dsfContact:fDiscloseAll> Exceptional disclosure preference flag of all of the supported disclose fields with type="boolean". This single field represents the <dsfContact:fDiscloseFlag> field value for all of the disclose fields, that include <dsfContact:fDiscloseNameLoc>, <dsfContact:fDiscloseNameInt>, <dsfContact:fDiscloseOrgLoc>, <dsfContact:fDiscloseOrgInt>, <dsfContact:fDiscloseAddrLoc>, <dsfContact:fDiscloseAddrInt>, <dsfContact:fDiscloseVoice>, <dsfContact:fDiscloseFax>, and <dsfContact:fDiscloseEmail>.

The following are example DSF files using the contact object fields. The different forms of contact object DSF files is up to server policy.

Example of a "contact.update.replaceClientStatuses" DSF that will set the client statuses of a contact to those specified in the DSF record. The client statuses set will be replaced by the set of client statuses specified. Empty statuses will result in the removal of those statuses if previously set. Non-empty statuses will be added if not previously set. Non-empty statuses will result in no change if previously set. All three client statuses defined in [RFC5733] are defined in the DSF in fixed order.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
  "urn:ietf:params:xml:ns:dsfContact-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.update.replaceClientStatuses
    </dataSet:type>
    <dataSet:fields>
      <dsfContact:fId/>
      <!-- clientDeleteProhibited -->
      <dsfContact:fStatus/>
      <!-- clientTransferProhibited -->
      <dsfContact:fStatus/>
      <!-- clientUpdateProhibited -->
      <dsfContact:fStatus/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
sh1111,clientDeleteProhibited,,clientUpdateProhibited
sh2222,,clientUpdateProhibited
sh3333,,clientTransferProhibited,
sh4444,,,
-----END DATA SET-----
```

Example of a "contact.update.addRemoveStatus" DSF that will add a status and remove a status for each contact. Any status can be empty to indicate no action for that contact.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
  "urn:ietf:params:xml:ns:dsfContact-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.update.addRemoveClientStatuses
    </dataSet:type>
    <dataSet:fields>
      <dsfContact:fId/>
      <dsfContact:fStatus op="add"/>
      <dsfContact:fStatus op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
sh1111,clientDeleteProhibited,clientUpdateProhibited
sh2222,,clientUpdateProhibited
sh3333,clientUpdateProhibited,
sh4444,clientTransferProhibited,
-----END DATA SET-----
```

Example of a "contact.create.standard" DSF that supports creating each contact record with a standard set of fields / attributes for a contact. A "|" character is used as a separator to minimize conflicting with the contact data. This is an example of creating two contacts, where indented data set lines are a continuation from the prior line.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
  "urn:ietf:params:xml:ns:dsfContact-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.create.standard
    </dataSet:type>
    <dataSet:fields sep="|">
      <dsfContact:fId/>
      <dsfContact:fPostalType/>
      <dsfContact:fName/>
      <dsfContact:fOrg/>
      <dsfContact:fStreet index="0"/>
      <dsfContact:fStreet index="1"/>
      <dsfContact:fStreet index="2"/>
      <dsfContact:fCity/>
      <dsfContact:fSp/>
      <dsfContact:fPc/>
      <dsfContact:fCc/>
      <dsfContact:fVoice/>
      <dsfContact:fVoiceExt/>
      <dsfContact:fFax/>
      <dsfContact:fFaxExt/>
      <dsfContact:fEmail/>
      <dataSet:fAuthInfo/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
sh1111|int|John Doe|Example Inc.|
  123 Example Dr.|||Dulles|VA|20166-6503|US|
  +1.7035555555|1234|+1.7035555556||
  johndoe@example.com|2fooBAR
sh2222|int|Jane Doe|Example Inc.|
  123 Example Dr.|Suite 2222||Dulles|VA|20166-6503|US|
  +1.7035555555||||
  janedoe@example.com|2fooBAR
-----END DATA SET-----
```

Example of a "contact.create.routing" DSF that extends the "contact.create.standard" DSF with the <dsfContact:fSubProduct> field (Section 2.5) to route the create to the appropriate Top Level Domain (TLD) or group of TLDs. A "|" character is used as a separator to

minimize conflicting with the contact data. This is an example of creating two contacts, where indented data set lines are a continuation from the prior line.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
  "urn:ietf:params:xml:ns:dsfContact-1.0"
  xmlns:dsfRouting=
  "urn:ietf:params:xml:ns:dsfRouting-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.create.routing
    </dataSet:type>
    <dataSet:fields sep="|">
      <dsfRouting:fSubProduct/>
      <dsfContact:fId/>
      <dsfContact:fPostalType/>
      <dsfContact:fName/>
      <dsfContact:fOrg/>
      <dsfContact:fStreet index="0"/>
      <dsfContact:fStreet index="1"/>
      <dsfContact:fStreet index="2"/>
      <dsfContact:fCity/>
      <dsfContact:fSp/>
      <dsfContact:fPc/>
      <dsfContact:fCc/>
      <dsfContact:fVoice/>
      <dsfContact:fVoiceExt/>
      <dsfContact:fFax/>
      <dsfContact:fFaxExt/>
      <dsfContact:fEmail/>
      <dataSet:fAuthInfo/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
EXAMPLE|sh1111|int|John Doe|Example Inc.|
  123 Example Dr.|||Dulles|VA|20166-6503|US|
  +1.7035555555|1234|+1.7035555556||
  johndoe@example.com|2fooBAR
EXAMPLE|sh2222|int|Jane Doe|Example Inc.|
  123 Example Dr.|Suite 2222||Dulles|VA|20166-6503|US|
  +1.7035555555||||
  janedoe@example.com|2fooBAR
-----END DATA SET-----
```

4.4. Verification Code Object

The Verification Code Object is defined in the Verification Code Extension for the Extensible Provisioning Protocol [I-D.gould-eppext-verificationcode]. This section defines the Verification Code specific Data Set File (DSF) field elements and sample DSF files used to set verification codes with a bulk update.

The Verification Service Provider (VSP) is a certified party to verify that data is in compliance with the policies of a locality. A locality MAY require the client to have data verified in accordance with local regulations or laws utilizing data sources not available to the server. The VSP has access to the local data sources and is authorized to verify the data. Examples include verifying that the domain name is not prohibited and verifying that the domain name registrant is a valid individual, organization, or business in the locality. The data verified, and the objects and operations that require the verification code to be passed to the server is up to the policies of the locality. The verification code represents a marker that the verification was completed. The data verified by the VSP MUST be stored by the VSP along with the generated verification codes in order to address any compliance issues. The signer certificate and the digital signature of the verification code MUST be verified by the server.

The Data Set File (DSF) field elements specific to the Verification Code Object include:

dsfVerificationCode:fCode Field that represents a Verification Code Token value. The field extends a "dataSet:fieldOptionalType", as defined in Section 2.4.1, with the type="dataSet:verificationCodeValueType". The field includes the required "codeType" attribute that represents the type of verification code, as defined by the "type" attribute of the <verificationCode:signedCode> element in [I-D.gould-eppext-verificationcode]. The verification code field can be empty since it is a "dataSet:fieldOptionalType".

dsfVerificationCode:fEncodedSignedCode Field that represents a Encoded Signed Code. The field extends a "dataSet:fieldOptionalType", as defined in Section 2.4.1, with the type="token". The field includes the OPTIONAL "encoding" attribute with the default value of "base64". The line breaks, defined in [RFC2045], MUST NOT be used with the "base64" Encoded Signed Code.

The Data Set File (DSF) used for setting of the verification codes MAY include fields from other objects and the fields specific to the Verification Code Object. There are two scenarios in passing the

verification codes based on who generates the DSF file, which includes:

VSP Generated DSF The VSP generates the Verification Code Token values, represented with the <dsfVerificationCode:fCode> field, and digitally signs the DSF file by referencing the <dataSet:encodedSignedDefData> element in the DSF header.

Client Generated DSF The client generates the DSF file with a set of encoded signed codes, represented with the <dsfVerificationCode:fEncodedSignedCode> field, collected from the VSP to bulk submit them to the server. The <dataSet:defData> element, described in Section 3.1.1, is referenced in the DSF header.

The following examples reference verification codes supported by the China Locality, which includes the Domain Name Verification Code (DNVC) and the Real Name Verification Code (RNVC). Both China Locality verification codes are set on a domain name with the verification code type of "domain" used for the DNVC and the verification code type of "real-name" used for the RNVC. Other localities MAY include a different set of verification codes.

Example VSP Generated DSF that includes a domain name primary key field, a "domain" code token value field, and a "real-name" token value field. The "base64" value includes "..." representing continuation of data for brevity:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0">
  <dataSet:encodedSignedDefData encoding="base64">
  ICAgICAgPHZlcmlmaWNhdGlvbkNvZGU6c2lnbmVkQ29kZQogICAgICAgIHhtbG5z
  ...
  b25Db2RlOnNpZ25lZENvZGU+Cg==
  </dataSet:encodedSignedDefData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,0-abc111,0-abc222
domain2.example,0-abc333,0-abc444
domain3.example,0-abc555,
domain3.example,,0-abc666
-----END DATA SET-----
```

Example Client Generated Data Set File (DSF) that includes a domain name primary key field, a "domain" encoded signed code field, and a "real-name" encoded signed code field. The "base64" values include no line breaks and include "..." representing continuation of data for brevity:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0"
  xmlns:dsfVerificationCode=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
  >
  <dataSet:defData>
    <dataSet:type subType="china">
      verificationCode.update.encodedSignedCode
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfVerificationCode:fEncodedSignedCode type="domain"/>
      <dsfVerificationCode:fEncodedSignedCode type="real-name"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ICAg...GlvbkN,CAgICA...uOmlldG
domain2.example,YW1zOb2R...GlvbkNvZGU6c2ln,
domain3.example,,CAgICAnZ...htbDpZmljYXMCIKI
-----END DATA SET-----
```

5. Result Codes

The result codes are based on and not equal to the reply codes defined in the EPP result codes of [RFC5730]. Four decimal digits are used to describe the success or failure of the processing of the Data Set File (DSF) or the processing of an individual record. Each digit of the result code has specific significance.

The first digit denotes success or failure. The second digit denotes the result category, such as request syntax or security. The third and fourth digits provide explicit response detail within each result category.

There are two values for the first digit of the result code:

1yzz Positive result. The file or record was processed by the system successfully. Partial successful results MAY be included.
2yzz Negative result. The file or record was not processed successfully. Partial successful results MAY NOT be included.

The second digit groups responses into one of five specific categories:

x0zz File or record syntax
x1zz Implementation-specific rules
x2zz Security
x3zz Data Management
x4zz Server System

Successful result codes:

1000 "Success"
This is the result for a successfully processed file when all records are processed successfully or the result for an individual record that is successfully processed.

1001 "Success with failures"
This is the result for a successfully processed file when some records are processed successfully and some records failed.

1002 "Success with all failures"
This is the result for a successfully processed file when all of the records failed.

Error result codes:

2000 "File syntax error"
This result code occurs when the overall file is syntactically incorrect.

2001 "Header syntax error"
This result code occurs when the header is syntactically incorrect.

2002 "Body syntax error"
This result code occurs when the body is syntactically incorrect.

2003 "Required parameter missing"
This result code occurs when a server receives a request for which a required parameter has not been provided.

2004 "Parameter value range error"
This result code occurs when a server receives a request whose value is outside the range of values specified in the protocol.

2005 "Parameter value syntax error"
This result code occurs when a server receives a request whose value is improperly formed.

2100 "Unimplemented protocol version"
This result code occurs when a server receives a request version, as defined by the XML namespace of the header, that is not implemented by the server.

2102 "Unimplemented option"
This result code occurs when a server receives a request that contains a protocol option that is not implemented by the server.

2103 "Unimplemented extension"
This result code occurs when a server receives a request that contains an extension, as defined by the XML namespace used by a field element extension, that is not implemented by the server.

2104 "Billing failure"
This result code occurs when a server receives a request that cannot be completed due to a client-billing failure.

2201 "Authorization error"
This result code occurs when a server receives a request cannot be execute due to a client-authorization error.

2202 "Invalid authorization information"
This result code occurs when a server receives invalid authorization information to execute a request.

2302 "Object exists"
This result code occurs when a server receives a request to create an object that already exists in the repository.

2303 "Object does not exist"
This result code occurs when a server receives a request to transform an object that does not exist in the repository.

2304 "Object status prohibits operation"
This result code occurs when a server receives a request to transform an object that cannot be completed due to server policy or business practices.

2305 "Object association prohibits operation"
This result code occurs when a server receives a request to transform an object that cannot be completed due to dependencies to other objects that are associated with the target object.

2306 "Parameter value policy error"
This result code occurs when a server receives a request that contains a parameter value that is syntactically valid but semantically invalid due to local policy.

2307 "Unimplemented object service"
This result code occurs when a server receives a request to operate on an object that is not supported by the server.

2308 "Data management policy violation"
This result code occurs when a server receives a request whose execution results in a violation of server data management policies.

2400 "Request failed"
This result code occurs when a server receives a request that cannot be executed due to an internal server error that is not related to the protocol.

The error result codes include some overlap that will require the server to decide which one to return based on server policy. For example, a "Header syntax error", represented by the error result code 2001, may have occured for multiple reasons including a 2003 "Required parameter missing", a 2004 "Parameter value range error", or a 2005 "Parameter value syntax error". The overlapping reasons also apply at the record level, where an object may be syntactically incorrect with a 2005 "Parameter value syntax error", and also be against server policy with a 2306 "Parameter value policy error". The server SHOULD attempt to provide the most accurate error result code to match the error. For example, a syntax error defined by the protocol SHOULD take precedence over a server policy error.

6. Example Data Set File (DSF) Result Files

This section includes a set of Data Set File (DSF) result examples.

6.1. Example Data Set File (DSF) with Result Data

This section includes example Data Set File (DSF) files that reference the <dataSet:resultData> element, described in Section 3.1.3, for providing the meta-data about the result file,

along with a set of fields that reference the <dataSet:fResultCode> field element, described in Section 2.4.2, for including the result code in processing the domain name record in the request.

Example Data Set File (DSF) for a succesful result by the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:resultData code="1000">
    <dataSet:fields>
       <dsfDomain:fName/>
       <dataSet:fResultCode/>
       <dataSet:fResultMsg/>
       <dataSet:fResultReason/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
    <dataSet:msg>Code Set processed successfully.</dataSet:msg>
    <dataSet:records>
      <dataSet:total>2</dataSet:total>
      <dataSet:success>2</dataSet:success>
      <dataSet:failed>0</dataSet:failed>
    </dataSet:records>
  </dataSet:resultData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,1000,,
domain2.example,1000,,
-----END DATA SET-----
```

Example Data Set File (DSF) for a partially successful result by the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:resultData code="1001">
    <dataSet:fields>
      <dsfDomain:fName/>
      <dataSet:fResultCode/>
      <dataSet:fResultMsg/>
      <dataSet:fResultReason/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
    <dataSet:msg>Success with failures</dataSet:msg>
    <dataSet:records>
      <dataSet:total>4</dataSet:total>
      <dataSet:success>1</dataSet:success>
      <dataSet:failed>3</dataSet:failed>
    </dataSet:records>
  </dataSet:resultData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,1000,Success,
domain2.example,2303,Object does not exist,
domain3.example,2201,Authorization error,
domain4.example,2302,Object exists,domain code already set
-----END DATA SET-----
```

Example Data Set File (DSF) for a failure result by the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
  "urn:ietf:params:xml:ns:dataSet-1.0">
  <dataSet:resultData code="2000">
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
    <dataSet:msg>File syntax error</dataSet:msg>
    <dataSet:reason lang="en">Malformed request file
    </dataSet:reason>
  </dataSet:resultData>
</dataSet:definition>
-----BEGIN DATA SET-----
-----END DATA SET-----
```

7. Formal Syntax

Each schema is included in a sub-section, starting with the Data Set schema and followed by each object schema.

The formal syntax presented here is a complete schema representation of the object mapping suitable for automated validation of EPP XML instances. The BEGIN and END tags are not part of the schema; they are used to note the beginning and ending of the schema for URI registration purposes.

7.1. Data Set Schema

Data Set schema that represents the base XML schema for the Data Set File (DSF).

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Data Set XML Schema
    </documentation>
```

```
    </annotation>

    <import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="xmldsig-core-schema.xsd"/>

    <element name="definition"
      type="dataSet:definitionType"/>

    <complexType name="definitionType">
      <sequence>
         <choice>
          <element name="encodedSignedDefData"
            type="dataSet:encodedSignedDefDataType"/>
          <element name="defData"
            type="dataSet:defDataType"/>
          <element name="resultData"
            type="dataSet:resultDataType"/>
        </choice>
      </sequence>
    </complexType>

    <!-- field seperator must be a single character -->
    <simpleType name="sepType">
      <restriction base="string">
        <minLength value="1"/>
        <maxLength value="1"/>
      </restriction>
    </simpleType>

    <!-- Ordered list of field definitions for the csv -->
    <complexType name="fieldsType">
      <sequence maxOccurs="unbounded">
        <element ref="dataSet:field"/>
      </sequence>
      <attribute name="sep" type="dataSet:sepType" default=","/>
    </complexType>

    <!-- defDataType -->
    <complexType name="defDataType">
      <sequence>
        <element name="type"
          type="dataSet:typeType"/>
        <element name="fields"
          type="dataSet:fieldsType"/>
        <element name="dataSetId"
          type="dataSet:IdType"/>
        <element name="crDate"
          type="dateTime"/>
```

```
      </sequence>
    </complexType>

    <!-- typeType -->
    <complexType name="typeType">
      <simpleContent>
        <extension base="token">
          <attribute name="subType" type="token"/>
        </extension>
      </simpleContent>
    </complexType>

    <!-- signedData -->
    <element name="signedDefData"
      type="dataSet:signedDefDataType"/>

    <complexType name="signedDefDataType">
      <complexContent>
        <extension base="dataSet:defDataType">
          <sequence>
            <element name="cksum"
              type="token"/>
            <element ref="dsig:Signature"/>
          </sequence>
          <attribute name="id" type="ID" use="required"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- resultDataType -->
    <complexType name="resultDataType">
      <sequence>
        <element name="type"
          type="dataSet:typeType"
          minOccurs="0"/>
        <element name="fields"
          type="dataSet:fieldsType"
          minOccurs="0"/>
        <element name="dataSetId"
          type="dataSet:IdType"
          minOccurs="0"/>
        <element name="svTRID"
          type="dataSet:IdType"/>
        <element name="msg"
          type="dataSet:msgType"/>
        <element name="reason"
          type="dataSet:msgType"
          minOccurs="0"/>
```

```
        <element name="records"
          type="dataSet:recordsType"
          minOccurs="0"/>
      </sequence>
      <attribute name="code" type="dataSet:resultCodeType"
        use="required"/>
    </complexType>

    <complexType name="recordsType">
      <sequence>
        <element name="total" type="unsignedInt"/>
        <element name="success" type="unsignedInt"/>
        <element name="failed" type="unsignedInt"/>
      </sequence>
    </complexType>

    <simpleType name="resultCodeType">
      <restriction base="unsignedShort">
        <!-- Success -->
        <enumeration value="1000"/>
        <!-- Success with failures -->
        <enumeration value="1001"/>
        <!-- Success with all failures -->
        <enumeration value="1002"/>
        <!-- File syntax error -->
        <enumeration value="2000"/>
        <!-- Invalid header -->
        <enumeration value="2001"/>
        <!-- Invalid body -->
        <enumeration value="2002"/>
        <!-- Required parameter missing -->
        <enumeration value="2003"/>
        <!-- Parameter value range error -->
        <enumeration value="2004"/>
        <!-- Parameter value syntax error -->
        <enumeration value="2005"/>
        <!-- Unimplemented protocol version -->
        <enumeration value="2100"/>
        <!-- Unimplemented option -->
        <enumeration value="2102"/>
        <!-- Unimplemented extension -->
        <enumeration value="2103"/>
        <!-- Billing failure -->
        <enumeration value="2104"/>
        <!-- Authorization error -->
        <enumeration value="2201"/>
        <!-- Invalid authorization information -->
        <enumeration value="2202"/>
```

```
        <!-- Object exists -->
        <enumeration value="2302"/>
        <!-- Object does not exist -->
        <enumeration value="2303"/>
        <!-- Object status prohibits operation -->
        <enumeration value="2304"/>
        <!-- Object association prohibits operation -->
        <enumeration value="2305"/>
        <!-- Parameter value policy error -->
        <enumeration value="2306"/>
        <!-- Unimplemented object service -->
        <enumeration value="2307"/>
        <!-- Data management policy violation -->
        <enumeration value="2308"/>
        <!-- Request failed -->
        <enumeration value="2400"/>
      </restriction>
    </simpleType>

    <simpleType name="IdType">
      <restriction base="token">
        <minLength value="3"/>
        <maxLength value="64"/>
      </restriction>
    </simpleType>

    <complexType name="msgType">
      <simpleContent>
        <extension base="normalizedString">
          <attribute name="lang" type="language"
            default="en"/>
        </extension>
      </simpleContent>
    </complexType>


    <!-- encodedSignedData -->
    <element name="encodedSignedDefData"
      type="dataSet:encodedSignedDefDataType"/>

    <complexType name="encodedSignedDefDataType">
      <simpleContent>
        <extension base="token">
          <attribute name="encoding"
            type="token" default="base64"/>
        </extension>
      </simpleContent>
    </complexType>
```

```
     <!-- Abstract field type -->
     <element name="field" type="dataSet:fieldType"
       abstract="true"/>

     <complexType name="fieldType">
       <sequence/>
     </complexType>

     <!-- fieldType with optional value (isRequired=false) -->
     <complexType name="fieldOptionalType">
       <complexContent>
         <extension base="dataSet:fieldType">
           <sequence/>
           <attribute name="isRequired" type="boolean"
             default="false"/>
           <attribute name="isPrimaryKey" type="boolean"
             default="false"/>
         </extension>
       </complexContent>
     </complexType>

    <!-- fieldType with required value (isRequired=false) -->
    <complexType name="fieldRequiredType">
     <complexContent>
       <extension base="dataSet:fieldType">
         <sequence/>
         <attribute name="isRequired" type="boolean"
           default="true"/>
         <attribute name="isPrimaryKey" type="boolean"
           default="false"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- fieldType as primary key field (isPrimaryKey=true) -->
    <complexType name="fieldPrimaryKeyType">
      <complexContent>
        <extension base="dataSet:fieldType">
          <sequence/>
          <attribute name="isRequired" type="boolean"
            default="true"/>
          <attribute name="isPrimaryKey" type="boolean"
            default="true"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- fieldType as list item -->
```

```
    <complexType name="fListItemType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="op"
            type="dataSet:listItemOpType"
            default="replace"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- Enumerated list item "op" attribute values -->
    <simpleType name="listItemOpType">
      <restriction base="token">
        <enumeration value="replace"/>
        <enumeration value="add"/>
        <enumeration value="remove"/>
      </restriction>
    </simpleType>

    <complexType name="fNameRequiredType">
      <complexContent>
        <extension base="dataSet:fieldRequiredType">
          <sequence/>
          <attribute name="type" type="token"
            default="eppcom\:labelType"/>
        </extension>
      </complexContent>
    </complexType>

    <element name="fName" type="dataSet:fNameType"
      substitutionGroup="dataSet:field"/>

    <complexType name="fNameType">
      <complexContent>
        <extension base="dataSet:fieldPrimaryKeyType">
          <sequence/>
          <attribute name="type" type="token"
            default="dataSet\:labelType"/>
          <attribute name="class" type="token"
            use="required"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- Authorization Information field -->
    <element name="fAuthInfo" type="dataSet:fAuthInfoType"
      substitutionGroup="dataSet:field"/>
```

```
    <complexType name="fAuthInfoType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="eppcom\:pwAuthInfoType"/>
        </extension>
      </complexContent>
    </complexType>


    <!-- General token type -->
    <complexType name="fTokenType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
           default="token"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- boolean type -->
    <complexType name="fBooleanType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
           default="boolean"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- fResultCode field -->
    <element name="fResultCode"
      type="dataSet:fResultCodeType"
      substitutionGroup="dataSet:field"/>
    <complexType name="fResultCodeType">
      <complexContent>
        <extension base="dataSet:fieldRequiredType">
          <sequence/>
          <attribute name="type" type="token"
            default="dataSet\:resultCodeType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- fResultMsg field -->
```

```
    <element name="fResultMsg"
      type="dataSet:fResultMsgType"
      substitutionGroup="dataSet:field"/>
    <complexType name="fResultMsgType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="normalizedString"/>
          <attribute name="lang" type="language"
            default="en"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- fResultReason field -->
    <element name="fResultReason"
      type="dataSet:fResultReasonType"
      substitutionGroup="dataSet:field"/>
    <complexType name="fResultReasonType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
           default="normalizedString"/>
         <attribute name="lang" type="language"
           default="en"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- unsignedByte type -->
    <complexType name="fUnsignedByteType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="unsignedByte"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- unsignedShort type -->
    <complexType name="fUnsignedShortType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
```

```
            default="unsignedShort"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- hexBinary type -->
    <complexType name="fHexBinaryType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="hexBinary"/>
        </extension>
      </complexContent>
    </complexType>

   </schema>
   END
```

7.2. Domain Name Schema

Domain Name Object XML schema that defines the fields specific to the Domain Name Object.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="urn:ietf:params:xml:ns:dsfDomain-1.0"
       xmlns:dsfDomain="urn:ietf:params:xml:ns:dsfDomain-1.0"
       xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
       xmlns:secDNS="urn:ietf:params:xml:ns:secDNS-1.1"
       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
       xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
       xmlns="http://www.w3.org/2001/XMLSchema"
       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:domain-1.0"
    schemaLocation="domain-1.0.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:secDNS-1.1"
    schemaLocation="secDNS-1.1.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
    schemaLocation="dataSet-1.0.xsd"/>
```

```
     <annotation>
      <documentation>
       Domain Name Data Set File (DSF) Object
      </documentation>
     </annotation>

    <!-- Domain name field -->
    <element name="fName" type="dataSet:fNameRequiredType"
      substitutionGroup="dataSet:field"/>

    <!-- Registration Period -->
    <element name="fPeriod" type="dsfDomain:fPeriodType"
      substitutionGroup="dataSet:field"/>

    <complexType name="fPeriodType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="domain\:pLimitType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- Registration Period Unit -->
    <element name="fPeriodUnit" type="dsfDomain:fPeriodUnitType"
      substitutionGroup="dataSet:field"/>

    <complexType name="fPeriodUnitType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="domain\:pUnitType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- Contact field -->
    <element name="fContact" type="dsfDomain:fContactsType"
      substitutionGroup="dataSet:field"/>

    <complexType name="fContactsType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="dataSet\:contactRoleType"/>
```

```
          <attribute name="role" type="dsfDomain:contactRoleType"
            use="required"/>
        </extension>
      </complexContent>
    </complexType>

    <simpleType name="contactRoleType">
      <restriction base="token">
        <enumeration value="registrant"/>
        <enumeration value="admin"/>
        <enumeration value="tech"/>
        <enumeration value="billing"/>
      </restriction>
    </simpleType>

    <!-- Domain name server -->
    <element name="fNs" type="dsfDomain:fNsType"
      substitutionGroup="dataSet:field"/>

    <complexType name="fNsType">
      <complexContent>
        <extension base="dataSet:fListItemType">
          <sequence/>
          <attribute name="type" type="token"
            default="domain\:pLimitType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- DNSSEC field types -->

    <!-- Maximum signature lifetime field -->
    <element name="fMaxSigLife" type="dsfDomain:fMaxSigLifeType"
      substitutionGroup="dataSet:field"/>
    <complexType name="fMaxSigLifeType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="secDNS\:maxSigLifeType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- Key tag field -->
    <element name="fKeyTag" type="dataSet:fUnsignedShortType"
      substitutionGroup="dataSet:field"/>
```

```
    <!-- DS Algorithm field -->
    <element name="fDsAlg" type="dataSet:fUnsignedByteType"
      substitutionGroup="dataSet:field"/>

    <!-- Digest type field -->
    <element name="fDigestType" type="dataSet:fUnsignedByteType"
      substitutionGroup="dataSet:field"/>

    <!-- Digest field -->
    <element name="fDigest" type="dataSet:fHexBinaryType"
      substitutionGroup="dataSet:field"/>

    <!-- Flags field -->
    <element name="fFlags" type="dataSet:fUnsignedShortType"
      substitutionGroup="dataSet:field"/>

    <!-- Protocol field -->
    <element name="fProtocol" type="dataSet:fUnsignedByteType"
      substitutionGroup="dataSet:field"/>

    <!-- Key Algorithm field -->
    <element name="fKeyAlg" type="dataSet:fUnsignedByteType"
      substitutionGroup="dataSet:field"/>

    <!-- Public Key field -->
    <element name="fPubKey" type="dsfDomain:fPubKeyType"
      substitutionGroup="dataSet:field"/>
    <complexType name="fPubKeyType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="secDNS\:keyType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- Domain status field -->
    <element name="fStatus" type="dsfDomain:fStatusType"
      substitutionGroup="dataSet:field"/>

    <!-- Domain status based on domain-1.0.xsd  -->
    <complexType name="fStatusType">
      <complexContent>
        <extension base="dataSet:fListItemType">
          <sequence/>
          <attribute name="type" type="token"
            default="domain\:statusValueType"/>
```

```
        </extension>
      </complexContent>
    </complexType>

     <!--
     End of schema.
     -->
   </schema>
   END
```

7.3. Host Schema

Host Object XML schema that defines the fields specific to the Host Object.

```
   BEGIN
   <?xml version="1.0" encoding="UTF-8"?>

   <schema targetNamespace="urn:ietf:params:xml:ns:dsfHost-1.0"
          xmlns:dsfHost="urn:ietf:params:xml:ns:dsfHost-1.0"
          xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
          xmlns:host="urn:ietf:params:xml:ns:host-1.0"
          xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
          xmlns="http://www.w3.org/2001/XMLSchema"
          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:host-1.0"
       schemaLocation="host-1.0.xsd"/>
     <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
       schemaLocation="dataSet-1.0.xsd"/>

     <annotation>
      <documentation>
       Host Name Data Set File (DSF) Object
      </documentation>
     </annotation>

    <!-- Host name field -->
    <element name="fName" type="dataSet:fNameRequiredType"
     substitutionGroup="dataSet:field"/>

    <!-- Host new name field -->
    <element name="fNewName" type="dsfHost:fNewNameType"
```

```
     substitutionGroup="dataSet:field"/>

    <complexType name="fNewNameType">
      <complexContent>
        <extension base="dataSet:fieldOptionalType">
          <sequence/>
          <attribute name="type" type="token"
            default="eppcom\:labelType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- IP address field -->
    <element name="fAddr" type="dsfHost:fAddrType"
      substitutionGroup="dataSet:field"/>

    <complexType name="fAddrType">
      <complexContent>
        <extension base="dataSet:fListItemType">
          <sequence/>
          <attribute name="type" type="token"
            default="host\:addrStringType"/>
        </extension>
      </complexContent>
    </complexType>

    <!-- IP address version field linked to the fAddr field -->
    <element name="fAddrVersion" type="dsfHost:fAddrVersionType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fAddrVersionType">
     <complexContent>
       <extension base="dataSet:fListItemType">
         <sequence/>
         <attribute name="type" type="token"
           default="host\:ipType"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Host status field -->
    <element name="fStatus" type="dsfHost:fStatusType"
      substitutionGroup="dataSet:field"/>

    <!-- Host status based on host-1.0.xsd  -->
    <complexType name="fStatusType">
      <complexContent>
        <extension base="dataSet:fListItemType">
          <sequence/>
```

```
          <attribute name="type" type="token"
            default="host\:statusValueType"/>
        </extension>
      </complexContent>
    </complexType>

     <!--
     End of schema.
     -->
   </schema>
   END
```

7.4. Contact Schema

Contact Object XML schema that defines the fields specific to the Contact Object.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="urn:ietf:params:xml:ns:dsfContact-1.0"
    xmlns:dsfContact="urn:ietf:params:xml:ns:dsfContact-1.0"
    xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
    xmlns:contact="urn:ietf:params:xml:ns:contact-1.0"
    xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
    xmlns="http://www.w3.org/2001/XMLSchema"
    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:contact-1.0"
   schemaLocation="contact-1.0.xsd"/>
 <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
   schemaLocation="dataSet-1.0.xsd"/>

 <annotation>
   <documentation>
     Contact Data Set File (DSF) Object
   </documentation>
 </annotation>


 <!-- Server-unique contact identifier field -->
 <element name="fId" type="dsfContact:fIdType"
  substitutionGroup="dataSet:field"/>
```

```
    <complexType name="fIdType">
     <complexContent>
       <extension base="dataSet:fieldRequiredType">
         <sequence/>
         <attribute name="type" type="token"
         default="eppcom\:clIDType"/>
       </extension>
     </complexContent>
    </complexType>


    <!-- Is Registrar Contact field -->
    <element name="fIsRegistrarContact"
     type="dataSet:fBooleanType"
     substitutionGroup="dataSet:field"/>


    <!-- voice and fax telephone number fields -->
    <element name="fVoice" type="dsfContact:fE164StringType"
     substitutionGroup="dataSet:field"/>
    <element name="fFax" type="dsfContact:fE164StringType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fE164StringType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
         default="contact\:e164StringType"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- voice and fax telephone extension fields -->
    <element name="fVoiceExt" type="dataSet:fTokenType"
     substitutionGroup="dataSet:field"/>
    <element name="fFaxExt" type="dataSet:fTokenType"
     substitutionGroup="dataSet:field"/>

    <!-- contact email address field -->
    <element name="fEmail" type="dsfContact:fEmailType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fEmailType">
     <complexContent>
       <extension base="dataSet:fieldRequiredType">
         <sequence/>
         <attribute name="type" type="token"
         default="eppcom\:minTokenType"/>
       </extension>
```

```
     </complexContent>
    </complexType>



    <!--
      Postal type field
      ("loc" = localized, "int" = internationalized)
    -->
    <element name="fPostalType" type="dsfContact:fPostalTypeType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fPostalTypeType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
         default="contact\:postalInfoEnumType"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Standard postal line field -->
    <complexType name="fPostalLineType">
     <complexContent>
       <extension base="dataSet:fieldRequiredType">
         <sequence/>
         <attribute name="type" type="token"
         default="contact\:postalLineType"/>
         <attribute name="isLoc" type="boolean"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Standard optional postal line field -->
    <complexType name="fOptPostalLineType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
         default="contact\:optPostalLineType"/>
         <attribute name="isLoc" type="boolean"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Name of the individual or role field -->
    <element name="fName" type="dsfContact:fPostalLineType"
     substitutionGroup="dataSet:field"/>
```

```
    <!-- Name organization field -->
    <element name="fOrg" type="dsfContact:fOptPostalLineType"
     substitutionGroup="dataSet:field"/>


    <!-- Street address line field with required index attribute  -->
    <!-- starting with index 0.  -->
    <element name="fStreet" type="dsfContact:fStreetType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fStreetType">
     <complexContent>
       <extension base="dsfContact:fOptPostalLineType">
         <sequence/>
         <attribute name="index" type="int"
         use="required"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Contact's city field -->
    <element name="fCity" type="dsfContact:fPostalLineType"
     substitutionGroup="dataSet:field"/>

    <!-- Contact's state or province field -->
    <element name="fSp" type="dsfContact:fOptPostalLineType"
     substitutionGroup="dataSet:field"/>

    <!-- Contact's postal code field -->
    <element name="fPc" type="dsfContact:fPcType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fPcType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
         default="contact\:pcType"/>
         <attribute name="isLoc" type="boolean"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Contact's country code field -->
    <element name="fCc" type="dsfContact:fCcType"
     substitutionGroup="dataSet:field"/>
    <complexType name="fCcType">
     <complexContent>
       <extension base="dataSet:fieldRequiredType">
         <sequence/>
```

```
        <attribute name="type" type="token"
        default="contact\:ccType"/>
        <attribute name="isLoc" type="boolean"/>
      </extension>
    </complexContent>
   </complexType>

   <!-- Disclosure element fields -->
   <!-- Flag of "1" to allow disclosure
        and "0" to disallow disclosure -->
   <element name="fDiscloseFlag" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure of localized name
        based on fDiscloseFlag? -->
   <element name="fDiscloseNameLoc" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure of internationalized name
        based on fDiscloseFlag? -->
   <element name="fDiscloseNameInt" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure of localized org
        based on fDiscloseFlag? -->
   <element name="fDiscloseOrgLoc" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure of internationalized org
        based on fDiscloseFlag? -->
   <element name="fDiscloseOrgInt" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure of localized address
        based on fDiscloseFlag? -->
   <element name="fDiscloseAddrLoc" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure of internationalized address
        based on fDiscloseFlag? -->
   <element name="fDiscloseAddrInt" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure voice telephone number
        based on fDiscloseFlag? -->
   <element name="fDiscloseVoice" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure facsimile telephone number
        based on fDiscloseFlag? -->
   <element name="fDiscloseFax" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
   <!-- Disclosure email address
        based on fDiscloseFlag? -->
   <element name="fDiscloseEmail" type="dsfContact:fBoolean"
    substitutionGroup="dataSet:field"/>
```

```
    <complexType name="fBoolean">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
         default="boolean"/>
       </extension>
     </complexContent>
    </complexType>

    <!-- Contact status field -->
    <element name="fStatus" type="dsfContact:fStatusType"
     substitutionGroup="dataSet:field"/>

    <!-- Host status based on contact-1.0.xsd  -->
    <complexType name="fStatusType">
     <complexContent>
       <extension base="dataSet:fListItemType">
         <sequence/>
         <attribute name="type" type="token"
           default="contact\:statusValueType"/>
       </extension>
     </complexContent>
    </complexType>

   <!--
   End of schema.
   -->
   </schema>
   END
```

7.5. Verification Code Schema

Verification Code Object XML schema that defines the fields specific to the Verification Code Object.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
  xmlns:dsfVerificationCode=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <annotation>
```

```
      <documentation>
        Verification Code Data Set File (DSF) Object
      </documentation>
    </annotation>

    <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
      schemaLocation="dataSet-1.0.xsd"/>

   <!-- fCode field -->
   <element name="fCode" type="dsfVerificationCode:fCodeType"
     substitutionGroup="dataSet:field"/>
   <complexType name="fCodeType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
           default="dsfVerificationCode\:verificationCodeValueType"/>
         <attribute name="codeType" type="token"/>
       </extension>
     </complexContent>
   </complexType>

   <!-- Format of verification code value -->
   <simpleType name="verificationCodeValueType">
     <restriction base="token">
       <pattern value="\d+-[A-Za-z0-9]+"/>
     </restriction>
   </simpleType>

   <!-- fEncodedSignedCode field -->
   <element name="fEncodedSignedCode"
     type="dsfVerificationCode:fEncodedSignedCodeType"
     substitutionGroup="dataSet:field"/>
   <complexType name="fEncodedSignedCodeType">
     <complexContent>
       <extension base="dataSet:fieldOptionalType">
         <sequence/>
         <attribute name="type" type="token"
           default="token"/>
         <attribute name="encoding" type="token"
           default="base64"/>
       </extension>
     </complexContent>
    </complexType>

  </schema>
  END
```

7.6. Routing Schema

Routing XML schema that defines the fields specific to routing.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace=
    "urn:ietf:params:xml:ns:dsfRouting-1.0"
  xmlns:dsfRouting=
    "urn:ietf:params:xml:ns:dsfRouting-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Routing Data Set File (DSF) Fields
    </documentation>
  </annotation>

  <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
    schemaLocation="dataSet-1.0.xsd"/>

 <!-- fSubProduct field -->
 <element name="fSubProduct" type="dsfRouting:fSubProductType"
   substitutionGroup="dataSet:field"/>
 <complexType name="fSubProductType">
   <complexContent>
     <extension base="dataSet:fieldOptionalType">
       <sequence/>
       <attribute name="type" type="token"
         default="token"/>
     </extension>
   </complexContent>
 </complexType>

</schema>
END
```

8. IANA Considerations

8.1. XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688].

Registration request for the dataSet namespace:

URI: ietf:params:xml:ns:dataSet-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dataSet XML schema:

URI: ietf:params:xml:ns:dataSet-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfDomain namespace:

URI: ietf:params:xml:ns:dsfDomain-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfDomain XML schema:

URI: ietf:params:xml:ns:dsfDomain-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfHost namespace:

URI: ietf:params:xml:ns:dsfHost-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfHost XML schema:

URI: ietf:params:xml:ns:dsfHost-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfContact namespace:

URI: ietf:params:xml:ns:dsfContact-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfContact XML schema:

URI: ietf:params:xml:ns:dsfContact-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfVerificationCode namespace:

URI: ietf:params:xml:ns:dsfVerificationCode-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfVerificationCode XML schema:

URI: ietf:params:xml:ns:dsfVerificationCode-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfRouting namespace:

URI: ietf:params:xml:ns:dsfRouting-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfRouting XML schema:

URI: ietf:params:xml:ns:dsfRouting-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

9. Security Considerations

The data supported by the Data Set File (DSF) format MAY include information that needs to be protected with the use of a secure transport. For example, the <dataSet:fAuthInfo> field provides object authorization information, and as described in [RFC5731], both the client and the server MUST ensure that it is stored and exchanged with high-grade encryption mechanisms. The transport leveraged to exchange the DSF containing sensitive or secure data MUST protect it from inadvertent disclosure.

XML Signature [W3C.CR-xmldsig-core2-20120124] is used in this document to verify that the Data Set originated from a trusted source and that it wasn't tampered with in transit from the source to the client to the server. To support multiple source keys, the source

certificate chain MUST be included in the <X509Certificate> elements of the Signed Def Data (Section 3.1.2.1) and MUST chain up and be verified by the server against a set of trusted certificates.

It is RECOMMENDED that signed definition data does not include white-spaces between the XML elements in order to mitigate risks of invalidating the digital signature when transferring of signed codes between applications takes place.

Use of XML canonicalization SHOULD be used when generating the signed code. SHA256/RSA-SHA256 SHOULD be used for digesting and signing. The size of the RSA key SHOULD be at least 2048 bits.

10. Normative References

[I-D.gould-eppext-verificationcode]
Gould, J., "Verification Code Extension for the Extensible Provisioning Protocol (EPP)", draft-gould-eppext-verificationcode-04 (work in progress), September 2016.

[iso13239]
the International Organization for Standardization, "ISO 13239", December 2007, <http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=37010>.

[itu-t-V.42]
International Telecommunication Union, "ITU-T V.42 Series V: Data Communication Over the Telephone Network", March 2002, <https://www.itu.int/rec/T-REC-V.42-200203-I>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5733>.

[RFC5910] Gould, J. and S. Hollenbeck, "Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)", RFC 5910, DOI 10.17487/RFC5910, May 2010, <http://www.rfc-editor.org/info/rfc5910>.

[W3C.CR-xmldsig-core2-20120124] Cantor, S., Roessler, T., Eastlake, D., Yiu, K., Reagle, J., Solo, D., Datta, P., and F. Hirsch, "XML Signature Syntax and Processing Version 2.0", World Wide Web Consortium CR CR-xmldsig-core2-20120124, January 2012, <http://www.w3.org/TR/2012/CR-xmldsig-core2-20120124>.

Appendix A. Acknowledgements

The author wishes to acknowledge the work done by Scott Hollenbeck in the EPP RFC 5730 for providing some of the concepts and text used in this document.

Special suggestions that have been incorporated into this document were provided by John Colosi, Deepak Deshpande, Scott Hollenbeck, Suzy Strier, and Srikanth Veeramachaneni.

Appendix B. Change History

B.1. Change from 00 to 01

1. Fixed the ABNF.
2. Added support for the 1002 "Success with all failures" result code to indicate that the file was successfully processed but that all of the records failed.
3. Updated the description for the 1000 "Success" and the 1001 "Success with failures" result codes.

Author’s Address

James Gould
VeriSign, Inc.
12061 Bluemont Way
Reston, VA 20190
US

Email: jgould@verisign.com
URI: http://www.verisign.com

Registration Protocols Extensions
Internet-Draft
Updates: 7484 (if approved)
Intended status: Best Current Practice
Expires: October 20, 2017

S. Hollenbeck
Verisign Labs
A. Newton
ARIN

April 18, 2017

Registration Data Access Protocol (RDAP) Object Tagging
draft-hollenbeck-regext-rdap-object-tag-03

Abstract

The Registration Data Access Protocol (RDAP) includes a method that can be used to identify the authoritative server for processing domain name, IP address, and autonomous system number queries. The method does not describe how to identify the authoritative server for processing other RDAP query types, such as entity queries. This limitation exists because the identifiers associated with these query types are typically unstructured. This document describes an operational practice that can be used to add structure to RDAP identifiers that makes it possible to identify the authoritative server for additional RDAP queries.

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 20, 2017.

Table of Contents

1. Introduction

The Registration Data Access Protocol (RDAP) includes a method ([RFC7484]) that can be used to identify the authoritative server for processing domain name, IP address, and autonomous system number (ASN) queries. This method works because each of these data elements is structured in a way that facilitates automated parsing of the element and association of the data element with a particular RDAP service provider. For example, domain names include labels (such as "com", "net", and "org") that are associated with specific service providers.

As noted in Section 9 of RFC 7484 [RFC7484], the method does not describe how to identify the authoritative server for processing entity queries, name server queries, help queries, or queries using certain search patterns. This limitation exists because the identifiers bound to these queries are typically not structured in a way that makes it easy to associate an identifier with a specific service provider. This document describes an operational practice that can be used to add structure to RDAP identifiers that makes it possible to identify the authoritative server for additional RDAP queries.

2. Object Naming Practice

Tagging object identifiers with a service provider tag makes it possible to identify the authoritative server for processing an RDAP query using the method described in RFC 7484 [RFC7484]. A service provider tag is constructed by prepending the Unicode TILDE character "~" (U+007E, described as an "unreserved" character in RFC 3986 [RFC3986]) to an IANA-registered value that represents the service provider. For example, a tag for a service provider identified by the string value "ARIN" is represented as "~ARIN".

Service provider tags are concatenated to the end of RDAP query object identifiers to unambiguously identify the authoritative server for processing an RDAP query. Building on the example from Section 3.1.5 of RFC 7482 [RFC7482], an RDAP entity handle can be constructed that allows an RDAP client to bootstrap an entity query. The following identifier is used to find information for the entity associated with handle "XXXX" at service provider "ARIN":

XXXX~ARIN

Clients that wish to bootstrap an entity query can parse this identifier into distinct handle and service provider identifier elements. Handles can themselves contain TILDE characters; the service provider identifier is found following the last TILDE character in the tagged identifier. The service provider identifier is used to retrieve a base RDAP URL from an IANA registry. The base URL and entity handle are then used to form a complete RDAP query path segment. For example, if the base RDAP URL "https://example.com/rdap/" is associated with service provider "YYYY" in an IANA registry, an RDAP client will parse a tagged entity identifier "XXXX~YYYY" into distinct handle ("XXXX") and service provider ("YYYY") identifiers. The service provider identifier "YYYY" is used to query an IANA registry to retrieve the base RDAP URL "https://example.com/rdap/". The base RDAP URL is concatenated to the entity handle to create a complete RDAP query path segment of "https://example.com/rdap/entity/XXXX~YYYY".

Implementation of this practice requires tagging of unstructured potential query identifiers in RDAP responses. Consider these elided examples from Section 5.3 of RFC 7483 [RFC7483] in which the handle identifiers have been tagged with a service provider tag:

```
{
  "objectClassName" : "domain",
  "handle" : "XXXX~RIR",
  "ldhName" : "0.2.192.in-addr.arpa",
  "nameservers" :
```

```
    [
      ...
    ],
    "secureDNS":
    {
      ...
    },
    "remarks" :
    [
      ...
    ],
    "links" :
    [
      ...
    ],
    "events" :
    [
      ...
    ],
    "entities" :
    [
      {
        "objectClassName" : "entity",
        "handle" : "XXXX~RIR",
        "vcardArray":
        [
          ...
        ],
        "roles" : [ "registrant" ],
        "remarks" :
        [
          ...
        ],
        "links" :
        [
          ...
        ],
        "events" :
        [
          ...
        ]
      }
    ],
    "network" :
    {
      "objectClassName" : "ip network",
      "handle" : "XXXX~RIR",
      "startAddress" : "192.0.2.0",
```

```
      "endAddress" : "192.0.2.255",
      "ipVersion" : "v4",
      "name": "NET-RTR-1",
      "type" : "DIRECT ALLOCATION",
      "country" : "AU",
      "parentHandle" : "YYYY~RIR",
      "status" : [ "active" ]
    }
  }
```

Figure 1

```
  {
    "objectClassName" : "domain",
    "handle" : "XXXX~DNR",
    "ldhName" : "xn--fo-5ja.example",
    "unicodeName" : "foo.example",
    "variants" :
    [
      ...
    ],
    "status" : [ "locked", "transfer prohibited" ],
    "publicIds":
    [
      ...
    ],
    "nameservers" :
    [
      {
        "objectClassName" : "nameserver",
        "handle" : "XXXX~DNR",
        "ldhName" : "ns1.example.com",
        "status" : [ "active" ],
        "ipAddresses" :
        {
          ...
        },
        "remarks" :
        [
          ...
        ],
        "links" :
        [
          ...
        ],
        "events" :
        [
          ...
```

```
        ]
      },
      {
        "objectClassName" : "nameserver",
        "handle" : "XXXX~DNR",
        "ldhName" : "ns2.example.com",
        "status" : [ "active" ],
        "ipAddresses" :
        {
          ...
        },
        "remarks" :
        [
          ...
        ],
        "links" :
        [
          ...
        ],
        "events" :
        [
          ...
        ]
      }
    ],
    "secureDNS":
    {
      ...
    },
    "remarks" :
    [
      ...
    ],
    "links" :
    [
      ...
    ],
    "port43" : "whois.example.net",
    "events" :
    [
      ...
    ],
    "entities" :
    [
      {
        "objectClassName" : "entity",
        "handle" : "XXXX~ABC",
        "vcardArray":
```

```
          [
            ...
          ],
          "status" : [ "validated", "locked" ],
          "roles" : [ "registrant" ],
          "remarks" :
          [
            ...
          ],
          "links" :
          [
            ...
          ],
          "events" :
          [
            ...
          ]
        }
      ]
  }
```

Figure 2

As described in Section 5 of RFC 7483 [RFC7483], RDAP responses can contain "self" links. Service provider tags and self references SHOULD be consistent. If they are inconsistent, the service provider tag is processed with higher priority when using these values to identify a service provider.

There is a risk of unpredictable processing behavior if the TILDE character is used for naturally occurring, non-separator purposes in an entity handle. This could lead to a client mistakenly assuming that a TILDE character represents a separator and the text that follows TILDE is a service provider identifier. A client that queries the IANA registry for what they assume is a valid service provider will likely receive an unexpected invalid result. As a consequence, the TILDE character MUST NOT be used in an entity handle for any purpose other than to separate an object identifier from a service provider tag.

3. Bootstrap Service Registry for RDAP Service Providers

The bootstrap service registry for the RDAP service provider space is represented using the structure specified in Section 3 of RFC 7484 [RFC7484]. The JSON output of this registry contains alphanumeric identifiers that identify RDAP service providers, grouped by base RDAP URLs, as shown in this example.

```
{
  "version": "1.0",
  "publication": "YYYY-MM-DDTHH:MM:SSZ",
  "description": "RDAP service provider bootstrap values",
  "services": [
    [
      ["YYYY"],
      [
        "https://example.com/rdap/"
      ]
    ],
    [
      ["ZZ54"],
      [
        "http://rdap.example.org/"
      ]
    ],
    [
      ["1754"],
      [
        "https://example.net/rdap/",
        "http://example.net/rdap/"
      ]
    ]
  ]
 }
```

Figure 3

Alphanumeric service provider identifiers conform to the syntax specified in the IANA registry of Extensible Provisioning Protocol (EPP) Repository Identifiers [1].

3.1. Registration Procedure

The service provider registry is populated using the "First Come First Served" policy defined in RFC 5226 [RFC5226]. Provider identifier values can be derived and assigned by IANA on request. Registration requests include the requested service provider identifier (or an indication that IANA should assign an identifier) and one or more base RDAP URLs to be associated with the service provider identifier.

4. IANA Considerations

IANA is requested to create the RDAP Bootstrap Services Registry listed below and make it available as JSON objects. The contents of

this registry is described in Section 3, with the formal syntax specified in Section 10 of RFC 7484 [RFC7484].

4.1. Bootstrap Service Registry for RDAP Service Providers

Entries in this registry contain at least the following:

- An alphanumeric value that identifies the RDAP service provider being registered.
- One or more URLs that provide the RDAP service regarding this registration.

5. Implementation Status

NOTE: Please remove this section and the reference to RFC 7942 prior to publication as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942 [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

5.1. Verisign Labs

Responsible Organization: Verisign Labs
Location: https://rdap.verisignlabs.com/
Description: This implementation includes support for domain registry RDAP queries using live data from the .cc and .tv country code top-level domains. Client authentication is required to receive entity information in query responses.
Level of Maturity: This is a "proof of concept" research implementation.

Coverage: This implementation includes all of the features described in this specification.
Contact Information: Scott Hollenbeck, shollenbeck@verisign.com

6. Security Considerations

This practice helps to ensure that end users will get RDAP data from an authoritative source using a bootstrap method to find authoritative RDAP servers, reducing the risk of sending queries to non-authoritative sources. The method has the same security properties as the RDAP protocols themselves. The transport used to access the IANA registries can be more secure by using TLS [RFC5246], which IANA supports. Additional considerations associated with RDAP are described in RFC 7481 [RFC7481].

7. Acknowledgements

The author would like to acknowledge the following individuals for their contributions to the development of this document: Tom Harrison, and Marcos Sanz. In addition, the authors would like to recognize the Regional Internet Registry (RIR) operators (AFRINIC, APNIC, ARIN, LACNIC, and RIPE) that have been implementing and using the practice of tagging handle identifiers for several years. Their experience provided significant inspiration for the development of this document.

8. References

8.1. Normative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC7484] Blanchet, M., "Finding the Authoritative Registration Data (RDAP) Service", RFC 7484, DOI 10.17487/RFC7484, March 2015, <http://www.rfc-editor.org/info/rfc7484>.

8.2. Informative References

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

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

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", RFC 7481, DOI 10.17487/RFC7481, March 2015, <http://www.rfc-editor.org/info/rfc7481>.

[RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", RFC 7482, DOI 10.17487/RFC7482, March 2015, <http://www.rfc-editor.org/info/rfc7482>.

[RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, DOI 10.17487/RFC7483, March 2015, <http://www.rfc-editor.org/info/rfc7483>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <http://www.rfc-editor.org/info/rfc7942>.

8.3. URIs

[1] http://www.iana.org/assignments/epp-repository-ids/epp-repository-ids.xhtml#epp-repository-ids-1

Appendix A. Change Log

00: Initial version.

01: Changed separator character from HYPHEN MINUS to COMMERCIAL AT. Added a recommendation to maintain consistency between service provider tags and "self" links (suggestion received from Tom Harrison). Fixed a spelling error, and corrected the network example in Section 2 (editorial erratum reported for RFC 7483 by Marcos Sanz). Added acknowledgements.

02: Changed separator character from COMMERCIAL AT to TILDE. Clarity updates and fixed an example handle. Added text to describe the risk of separator characters appearing naturally in entity handles and being misinterpreted as separator characters.

03: Added Implementation Status section (Section 5).

Authors' Addresses

Scott Hollenbeck
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
USA

Email: shollenbeck@verisign.com
URI: http://www.verisignlabs.com/

Andrew Lee Newton
American Registry for Internet Numbers
PO Box 232290
Centreville, VA 20120
US

Email: andy@arin.net
URI: http://www.arin.net

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: November 25, 2017

S. Hollenbeck
Verisign Labs

May 24, 2017

Federated Authentication for the Registration Data Access Protocol (RDAP) using OpenID Connect
draft-hollenbeck-regext-rdap-openid-03

Abstract

The Registration Data Access Protocol (RDAP) provides "RESTful" web services to retrieve registration metadata from domain name and regional internet registries. RDAP allows a server to make access control decisions based on client identity, and as such it includes support for client identification features provided by the Hypertext Transfer Protocol (HTTP). Identification methods that require clients to obtain and manage credentials from every RDAP server operator present management challenges for both clients and servers, whereas a federated authentication system would make it easier to operate and use RDAP without the need to maintain server-specific client credentials. This document describes a federated authentication system for RDAP based on OpenID Connect.

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 November 25, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Registration Data Access Protocol (RDAP) provides "RESTful" web services to retrieve registration metadata from domain name and regional internet registries. RDAP allows a server to make access control decisions based on client identity, and as such it includes support for client identification features provided by the Hypertext Transfer Protocol (HTTP) [RFC7230].

RDAP is specified in multiple documents, including "HTTP Usage in the Registration Data Access Protocol (RDAP)" [RFC7480], "Security Services for the Registration Data Access Protocol (RDAP)" [RFC7481], "Registration Data Access Protocol Query Format" [RFC7482], and "JSON Responses for the Registration Data Access Protocol (RDAP)" [RFC7483]. RFC 7481 describes client identification and authentication services that can be used with RDAP, but it does not specify how any of these services can (or should) be used with RDAP.

1.1. Problem Statement

The traditional "user name and password" authentication method does not scale well in the RDAP ecosystem. Assuming that all domain name and address registries will eventually provide RDAP service, it is impractical and inefficient for users to secure login credentials from the hundreds of different server operators. Authentication methods based on user names and passwords do not provide information that describes the user in sufficient detail (while protecting the personal privacy of the user) for server operators to make fine-grained access control decisions based on the user's identity. The authentication system used for RDAP needs to address all of these needs.

1.2. Proposal

A basic level of RDAP service can be provided to users who possess an identifier issued by a recognized provider who is able to authenticate and validate the user. The identifiers issued by social media services, for example, can be used. Users who require higher levels of service (and who are willing to share more information about them self to gain access to that service) can secure identifiers from specialized providers who are or will be able to provide more detailed information about the user. Server operators can then make access control decisions based on the identification information provided by the user.

A federated authentication system would make it easier to operate and use RDAP by re-using existing identifiers to provide a basic level of access. It can also provide the ability to collect additional user identification information, and that information can be shared with the consent of the user. This document describes a federated authentication system for RDAP based on OpenID Connect [OIDC] that meets all of these needs.

2. 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 RFC 2119 [RFC2119].

3. Federated Authentication for RDAP

RDAP itself does not include native security services. Instead, RDAP relies on features that are available in other protocol layers to provide needed security services including access control, authentication, authorization, availability, data confidentiality, data integrity, and identification. A description of each of these security services can be found in "Internet Security Glossary, Version 2" [RFC4949]. This document focuses on a federated authentication system for RDAP that provides services for authentication, authorization, and identification, allowing a server operator to make access control decisions. Section 3 of RFC 7481 [RFC7481] describes general considerations for RDAP access control, authentication, and authorization.

The traditional client-server authentication model requires clients to maintain distinct credentials for every RDAP server. This situation can become unwieldy as the number of RDAP servers increases. Federated authentication mechanisms allow clients to use one credential to access multiple RDAP servers and reduce client credential management complexity.

3.1. RDAP and OpenID Connect

OpenID Connect 1.0 [OIDCC] is a decentralized, single sign-on (SSO) federated authentication system that allows users to access multiple web resources with one identifier instead of having to create multiple server-specific identifiers. Users acquire identifiers from OpenID Providers, or OPs. Relying Parties, or RPs, are applications (such as RDAP) that outsource their user authentication function to an OP. OpenID Connect is built on top of the authorization framework provided by the OAuth 2.0 [RFC6749] protocol.

The OAuth authorization framework describes a method for users to access protected web resources without having to hand out their credentials. Instead, clients are issued Access Tokens by authorization servers with the permission of the resource owners. Using OpenID Connect and OAuth, multiple RDAP servers can form a federation and clients can access any server in the federation by providing one credential registered with any OP in that federation. The OAuth authorization framework is designed for use with HTTP and thus can be used with RDAP.

3.1.1. Terminology

This document uses the terms "client" and "server" defined by RDAP [RFC7480]. An RDAP client performs the role of an OpenID Connect Core [OIDCC] Entity or End-User. An RDAP server performs the role of an OpenID Connect Core Relying Party (RP). Additional terms from Section 1.2 of the OpenID Connect Core specification are incorporated by reference.

3.1.2. Overview

At a high level, RDAP authentication of a browser-based client using OpenID Connect requires completion of the following steps:

1. An RDAP client (acting as an OpenID End-User) sends an HTTP (or HTTPS) query containing OAuth 2.0 request parameters to an RDAP server.
2. The RDAP server (acting as an OpenID Relying Party (RP)) prepares an Authentication Request containing the desired request parameters.
3. The RDAP server sends the RDAP client and Authentication Request to an Authorization Server operated by an OpenID Provider (OP) using an HTTP redirect.
4. The Authorization Server authenticates the RDAP Client.
5. The Authorization Server obtains RDAP Client consent/ authorization.
6. The Authorization Server sends the RDAP Client back to the RDAP server with an Authorization Code using an HTTP redirect.
7. The RDAP server requests a response using the Authorization Code at the Token Endpoint.
8. The RDAP server receives a response that contains an ID Token and Access Token in the response body.
9. The RDAP server validates the ID Token and retrieves the RDAP client’s Subject Identifier.

The RDAP server can then make identification, authorization, and access control decisions based on local policies, the ID Token received from the OP, and the received Claims. Note that OpenID

Connect describes different process flows for other types of clients, such as script-based or command line clients.

3.1.3. RDAP Authentication and Authorization Steps

End-Users MUST possess an identifier (an OpenID) issued by an OP to use OpenID Connect with RDAP. The OpenID Foundation maintains a list of OPs on its web site [1]. Additional OPs are almost certainly needed to fully realize the potential for federated authentication with RDAP because RDAP has authorization and access control requirements that go beyond the end-user authentication requirements of a typical web site.

OpenID Connect requires RPs to register with OPs to use OpenID Connect services for an End-User. That process is described by the "OpenID Connect Dynamic Client Registration" protocol [OIDCR].

3.1.3.1. Provider Discovery

An RDAP server/RP needs to receive an identifier from an End-User that can be used to discover the End-User's OP. That process is required and is documented in the "OpenID Connect Discovery" protocol [OIDCD].

3.1.3.2. Authentication Request

Once the OP is known, an RP MUST form an Authentication Request and send it to the OP as described in Section 3 of the OpenID Connect Core protocol [OIDCC]. The authentication path followed (authorization, implicit, or hybrid) will depend on the Authentication Request response_type set by the RP. The remainder of the processing steps described here assume that the Authorization Code Flow is being used by setting "response_type=code" in the Authentication Request.

The benefits of using the Authorization Code Flow for authenticating a human user are described in Section 3.1 of the OpenID Connect Core protocol. The Implicit Flow is more commonly used by clients implemented in a web browser using a scripting language; it is described in Section 3.2 of the OpenID Connect Core protocol. The Hybrid Flow (described in Section 3.3 of the OpenID Connect Core protocol) combines elements of the Authorization and Implicit Flows by returning some tokens from the Authorization Endpoint and others from the Token Endpoint.

An Authentication Request can contain several parameters. REQUIRED parameters are specified in Section 3.1.2.1 of the OpenID Connect Core protocol [OIDCC]. Other parameters MAY be included.

The OP receives the Authentication Request and attempts to validate it as described in Section 3.1.2.2 of the OpenID Connect Core protocol [OIDCC]. If the request is valid, the OP attempts to authenticate the End-User as described in Section 3.1.2.3 of the OpenID Connect Core protocol [OIDCC]. The OP returns an error response if the request is not valid or if any error is encountered.

3.1.3.3. End-User Authorization

After the End-User is authenticated, the OP MUST obtain authorization information from the End-User before releasing information to the RDAP Server/RP. This process is described in Section 3.1.2.4 of the OpenID Connect Core protocol [OIDCC].

3.1.3.4. Authorization Response and Validation

After the End-User is authenticated, the OP will send a response to the RP that describes the result of the authorization process in the form of an Authorization Grant. The RP MUST validate the response. This process is described in Sections 3.1.2.5 - 3.1.2.7 of the OpenID Connect Core protocol [OIDCC].

3.1.3.5. Token Processing

The RP sends a Token Request using the Authorization Grant to a Token Endpoint to obtain a Token Response containing an Access Token, ID Token, and an OPTIONAL Refresh Token. The RP MUST validate the Token Response. This process is described in Section 3.1.3 of the OpenID Connect Core protocol [OIDCC].

3.1.3.6. Delivery of User Information

The set of Claims can be retrieved by sending a request to a UserInfo Endpoint using the Access Token. The Claims MAY be returned in the ID Token. The process of retrieving Claims from a UserInfo Endpoint is described in Section 5.3 of the OpenID Connect Core protocol [OIDCC].

OpenID Connect specified a set of standard Claims in Section 5.1. Additional Claims for RDAP are described in Section 3.1.4.1.

3.1.4. Specialized Parameters for RDAP

3.1.4.1. Claims

OpenID Connect claims are pieces of information used to make assertions about an entity. Section 5 of the OpenID Connect Core protocol [OIDCC] describes a set of standard claims that can be used

to identify a person. Section 5.1.2 notes that additional claims MAY be used, and it describes a method to create them.

3.1.4.1.1. Stated Purpose

There are communities of RDAP users and operators who wish to make and validate claims about a user's "need to know" when it comes to requesting access to a resource. For example, a law enforcement agent or a trademark attorney may wish to be able to assert that they have a legal right to access a protected resource, and a server operator will need to be able to receive and validate that claim. These needs can be met by defining and using an additional "purpose" claim.

The "purpose" claim identifies the purpose for which access to a protected resource is being requested. Use of the "purpose" claim is OPTIONAL; processing of this claim is subject to server acceptance of the purpose and successful authentication of the End-User. Unrecognized purpose values MUST be ignored and the associated query MUST be processed as if the unrecognized purpose value was not present at all.

The "purpose" value is a case-sensitive string containing a StringOrURI value as specified in Section 2 of the JSON Web Token (JWT) specification ([RFC7519]). An example:

```
{"purpose" : "domainNameControl"}
```

Purpose values are themselves registered with IANA. Each entry in the registry contains the following fields:

Value: the purpose string value being registered. Value strings can contain upper case characters from "A" to "Z", lower case ASCII characters from "a" to "z", and the underscore ("_") character. Value strings contain at least one character and no more than 64 characters.

Description: a one- or two-sentence description of the meaning of the purpose value, how it might be used, and/or how it should be interpreted by clients and servers.

This registry is operated under the "Specification Required" policy defined in RFC 5226 ([RFC5226]). The set of initial values used to populate the registry as described in Section 6.2 are taken from the final report [2] produced by the Expert Working Group on gTLD Directory Services chartered by the Internet Corporation for Assigned Names and Numbers (ICANN).

4. Protocol Parameters

This specification adds the following protocol parameters to RDAP:

1. A query parameter to request authentication for a specific end-user identity.
2. A path segment to request an ID Token and an Access Token for a specific end-user identity.
3. A query parameter to deliver an ID Token and an Access Token for use with an RDAP query.

4.1. Client Authentication Request and Response

Client authentication is requested by adding a query component to an RDAP request URI using the syntax described in Section 3.4 of RFC 3986 [RFC3986]. The query used to request client authentication is represented as a "key=value" pair using a key value of "id" and a value component that contains the client identifier issued by an OP. An example:

https://example.com/rdap/domain/example.com?id=user.idp.example

The response to an authenticated query MUST use the response structures specified in RFC 7483 [RFC7483]. Information that the end-user is not authorized to receive MUST be omitted from the response.

4.2. Token Request and Response

Clients MAY send a request to an RDAP server to authenticate an end-user and return an ID Token and an Access Token from an OP that can be then be passed to the RP/RDAP server to authenticate and process subsequent queries. Identity provider authentication is requested using a "tokens" path segment and a query parameter with key value of "id" and a value component that contains the client identifier issued by an OP. An example:

https://example.com/rdap/tokens?id=user.idp.example

In addition to any core RDAP response elements, the response to this query MUST contain four name-value pairs, in any order, representing the returned ID Token and Access Token. The ID Token is represented using a key value of "id_token". The Access Token is represented using a key value of "access_token". The access token type is represented using a key value of "token_type" and a value of "bearer" as described in Sections 4.2.2 and 7.1 of RFC 6749 [RFC6749]. The lifetime of the access token is represented using a key value of "expires_in" and a numerical value that describes the lifetime in

seconds of the access token as described in Section 4.2.2 of RFC 6749 [RFC6749]. The token values returned in the RDAP server response MUST be Base64url encoded as described in RFCs 7515 [RFC7515] and 7519 [RFC7519].

An example (the encoded tokens have been abbreviated for clarity):

```
{
 "access_token" : "eyJ0...NiJ9",
 "id_token" : "eyJ0...EjXk",
 "token_type" : "bearer",
 "expires_in" : "3600"
}
```

Figure 1

An RDAP server that processes this type of query MUST determine if the identifier is associated with an OP that is recognized and supported by the server. Servers MUST reject queries that include an identifier associated with an unsupported OP with an HTTP 501 (Not Implemented) response. An RDAP server that receives a query containing an identifier associated with a recognized OP MUST perform the steps required to authenticate the user with the OP using a browser or browser-like client and return encoded tokens to the client. Note that tokens are typically valid for a limited period of time and new tokens will be required when an existing token’s validity period has expired.

The tokens can then be passed to the server for use with an RDAP query using a query parameter with key values of "id_token" and "access_token" and values that represent the encoded tokens. An example (the encoded tokens have been abbreviated and the URI split across multiple lines for clarity):

```
https://example.com/rdap/domain/example.com
?id_token=eyJ0...EjXk
&access_token=eyJ0...NiJ9
```

The response to an authenticated query MUST use the response structures specified in RFC 7483 [RFC7483]. Information that the end-user is not authorized to receive MUST be omitted from the response.

4.3. Token Refresh and Revocation

An access token can be refreshed as described in Section 12 of the OpenID Connect Core protocol [OIDCC] and Section 6 of OAuth 2.0

[RFC6749]. Clients can take advantage of this functionality if it is supported by the OP and accepted by the RDAP server.

A refresh token is requested using a "tokens" path segment and two query parameters. The first query parameter includes a key value of "id" and a value component that contains the client identifier issued by an OP. The second query parameter includes a key value of "refresh" and a value component of "true". A value component of "false" MUST be processed to return a result that is consistent with not including a "refresh" parameter at all as described in Section 4.2. An example using "refresh=true":

https://example.com/rdap/tokens?id=user.idp.example
&refresh=true

The response to this query MUST contain all of the response elements described in Section 4.2. In addition, the response MUST contain a name-value pair that represents a refresh token. The name-value pair includes a key value of "refresh_token" and a Base64url-encoded value that represents the refresh token.

Example refresh token request response (the encoded tokens have been abbreviated for clarity):

```
{
 "access_token" : "eyJ0...NiJ9",
 "id_token" : "eyJ0...EjXk",
 "token_type" : "bearer",
 "expires_in" : "3600",
 "refresh_token" : "eyJ0...c8da"
}
```

Figure 2

Once acquired, a refresh token can be used to refresh an access token. An access token is refreshed using a "tokens" path segment and two query parameters. The first query parameter includes a key value of "id" and a value component that contains the client identifier issued by an OP. The second query parameter includes a key value of "refresh_token" and a Base64url-encoded value that represents the refresh token. An example:

https://example.com/rdap/tokens?id=user.idp.example
&refresh_token=eyJ0...f3jE

In addition to any core RDAP response elements, the response to this query MUST contain four name-value pairs, in any order, representing a returned Refresh Token and Access Token. The Refresh Token is

represented using a key value of "refresh_token". The Access Token is represented using a key value of "access_token". The access token type is represented using a key value of "token_type" and a value of "bearer" as described in Sections 4.2.2 and 7.1 of RFC 6749 [RFC6749]. The lifetime of the access token is represented using a key value of "expires_in" and a numerical value that describes the lifetime in seconds of the access token as described in Section 4.2.2 of RFC 6749 [RFC6749]. The token values returned in the RDAP server response MUST be Base64url encoded as described in RFCs 7515 [RFC7515] and 7519 [RFC7519].

Example access token refresh response (the encoded tokens have been abbreviated for clarity):

```
{
 "access_token" : "0dac...13b0",
 "refresh_token" : "f735...d30c",
 "token_type" : "bearer",
 "expires_in" : "3600"
}
```

Figure 3

Access and refresh tokens can be revoked as described in RFC 7009 [RFC7009] by sending a request to an RDAP server that contains a "tokens/revoke" path segment and two query parameters. The first query parameter includes a key value of "id" and a value component that contains the client identifier issued by an OP. The second query parameter includes a key value of "token" and a Base64url-encoded value that represents either the current refresh token or the associated access token. An example:

https://example.com/rdap/tokens/revoke?id=user.idp.example
&token=f735...d30c

Note that this command will revoke both access and refresh tokens at the same time. In addition to any core RDAP response elements, the response to this query MUST contain a description of the result of processing the revocation request within the RDAP "notices" data structure.

Example token revocation success:

```
"notices" :
[
  {
    "title" : "Token Revocation Result",
    "description" : "Token revocation succeeded.",
  }
],
"lang" : "en-US"
```

Figure 4

Example token revocation failure:

```
"notices" :
[
  {
    "title" : "Token Revocation Result",
    "description" : "Token revocation failed.",
  }
],
"errorCode" : 400,
"lang" : "en-US"
```

Figure 5

4.4. Parameter Processing

Unrecognized query parameters MUST be ignored. An RDAP request that does not include an "id" query component MUST be processed as an unauthenticated query. An RDAP server that processes an authenticated query MUST determine if the identifier is associated with an OP that is recognized and supported by the server. Servers MUST reject queries that include an identifier associated with an unsupported OP with an HTTP 501 (Not Implemented) response. An RDAP server that receives a query containing an identifier associated with a recognized OP MUST perform the steps required to authenticate the user with the OP, process the query, and return an RDAP response that is appropriate for the end user’s level of authorization and access.

An RDAP server that receives a query containing tokens associated with a recognized OP and authenticated end user MUST process the query and return an RDAP response that is appropriate for the end user’s level of authorization and access. Errors based on processing either the ID Token or the Access Token MUST be signaled with an appropriate HTTP status code as described in Section 3.1 of RFC 6750 [RFC6750].

On receiving a query containing tokens, the RDAP server MUST validate the ID Token. It can do this independently of the OP, because the ID Token is a JWT that contains all the data necessary for validation. The Access Token, however, is an opaque value, and can only be validated by sending a request using it to the UserInfo Endpoint and confirming that a successful response is received. This is different from the OpenID Connect Authorization Code and Implicit flows, where the Access Token can be validated against the at_hash claim from the ID Token. With a query containing tokens, the Access Token might not validate against the at_hash claim because the Access Token may have been refreshed since the ID Token was issued.

An RDAP server that processes requests without needing the UserInfo claims does not need to retrieve the claims merely in order to validate the Access Token. Similarly, an RDAP server that has cached the UserInfo claims for an end user, in accordance with the HTTP headers of a previous UserInfo Endpoint response, does not need to retrieve those claims again in order to revalidate the Access Token.

5. Non-Browser Clients

The flow described in Section 3.1.3 requires a client to interact with a server using a web browser. This will not work well in situations where the client is automated or an end-user is using a command line client such as curl [3] or wget [4]. This is a known issue with OpenID Connect, and is typically addressed using a two-step process:

1. Authenticate with the OP using a browser or browser-like client and store the ID Token and Access Token locally.
2. Send a request to the content provider/RP along with the ID Token and Access Token received from the OP.

The Access Token MAY be passed to the RP in an HTTP "Authorization" header [RFC7235] or as a query parameter. The Access Token MUST be specified using the "Bearer" authentication scheme [RFC6750] if it is passed in an "Authorization" header. The ID Token MUST be passed to the RP as a query parameter.

Here are two examples using the curl and wget utilities. Start by authenticating with the OP:

https://example.com/rdap/tokens?id=user.idp.example

Save the token information and pass it to the RP along with the URI representing the RDAP query. Using curl (encoded tokens have been abbreviated for clarity:

```
curl -H "Authorization: Bearer eyJ0...NiJ9"\
-k https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk

curl -k https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk&access_token=eyJ0...NiJ9
```

Using wget:

```
wget --header="Authorization: Bearer eyJ0...NiJ9"\
https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk

wget https://example.com/rdap/domain/example.com\
?id_token=eyJ0...EjXk&access_token=eyJ0...NiJ9
```

Refresh tokens can be useful to automated or command line clients who wish to continue a session without explicitly re-authenticating an end user. See Section 4.3 for more information.

6. IANA Considerations

6.1. JSON Web Token Claims Registry

IANA is requested to register the following value in the JSON Web Token Claims Registry:

> Claim Name: "purpose"
> Claim Description: The stated purpose for submitting a request to access a protected RDAP resource.
> Change Controller: IESG
> Specification Document(s): Section 3.1.4.1.1 of this document.

6.2. RDAP Query Purpose Registry

IANA is requested to create a new protocol registry to manage RDAP query purpose values. This registry should appear under its own heading on IANA's protocol listings, using the same title as the name of the registry. The information to be registered and the procedures to be followed in populating the registry are described in Section 3.1.4.1.1.

Name of registry: Registration Data Access Protocol (RDAP) Query Purpose Values

Section at http://www.iana.org/protocols:

Registry Title: Registration Data Access Protocol (RDAP) Query Purpose Values

Registry Name: Registration Data Access Protocol (RDAP) Query Purpose Values

Registration Procedure: Specification Required

Reference: This draft

Required information: See Section 3.1.4.1.1.

Review process: "Specification Required" as described in RFC 5226 [RFC5226].

Size, format, and syntax of registry entries: See Section 3.1.4.1.1.

Initial assignments and reservations:

-----BEGIN FORM-----
Value: domainNameControl
Description: Tasks within the scope of this purpose include creating and managing and monitoring a registrant’s own domain name, including creating the domain name, updating information about the domain name, transferring the domain name, renewing the domain name, deleting the domain name, maintaining a domain name portfolio, and detecting fraudulent use of the Registrant’s own contact information.
-----END FORM-----

-----BEGIN FORM-----
Value: personalDataProtection
Description: Tasks within the scope of this purpose include identifying the accredited privacy/proxy provider associated with a domain name and reporting abuse, requesting reveal, or otherwise contacting the provider.
-----END FORM-----

-----BEGIN FORM-----
Value: technicalIssueResolution
Description: Tasks within the scope of this purpose include (but are not limited to) working to resolve technical issues, including email delivery issues, DNS resolution failures, and web site functional issues.
-----END FORM-----

-----BEGIN FORM-----
Value: domainNameCertification

Description: Tasks within the scope of this purpose include a Certification Authority (CA) issuing an X.509 certificate to a subject identified by a domain name.
-----END FORM-----

-----BEGIN FORM-----
Value: individualInternetUse
Description: Tasks within the scope of this purpose include identifying the organization using a domain name to instill consumer trust, or contacting that organization to raise a customer complaint to them or file a complaint about them.
-----END FORM-----

-----BEGIN FORM-----
Value: businessDomainNamePurchaseOrSale
Description: Tasks within the scope of this purpose include making purchase queries about a domain name, acquiring a domain name from a registrant, and enabling due diligence research.
-----END FORM-----

-----BEGIN FORM-----
Value: academicPublicInterestDNSRResearch
Description: Tasks within the scope of this purpose include academic public interest research studies about domain names published in the registration data service, including public information about the registrant and designated contacts, the domain name’s history and status, and domain names registered by a given registrant (reverse query).
-----END FORM-----

-----BEGIN FORM-----
Value: legalActions
Description: Tasks within the scope of this purpose include investigating possible fraudulent use of a registrant’s name or address by other domain names, investigating possible trademark infringement, contacting a registrant/licensee’s legal representative prior to taking legal action and then taking a legal action if the concern is not satisfactorily addressed.
-----END FORM-----

-----BEGIN FORM-----
Value: regulatoryAndContractEnforcement
Description: Tasks within the scope of this purpose include tax authority investigation of businesses with online presence, Uniform Dispute Resolution Policy (UDRP) investigation, contractual compliance investigation, and registration data escrow audits.
-----END FORM-----

```
-----BEGIN FORM-----
Value: criminalInvestigationAndDNSAbuseMitigation
Description: Tasks within the scope of this purpose include reporting
abuse to someone who can investigate and address that abuse, or
contacting entities associated with a domain name during an offline
criminal investigation.
-----END FORM-----

-----BEGIN FORM-----
Value: dnsTransparency
Description: Tasks within the scope of this purpose involve querying
the registration data made public by registrants to satisfy a wide
variety of use cases around informing the general public.
-----END FORM-----
```

7. Implementation Status

NOTE: Please remove this section and the reference to RFC 7942 prior to publication as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942 [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

7.1. Verisign Labs

Responsible Organization: Verisign Labs
Location: https://rdap.verisignlabs.com/
Description: This implementation includes support for domain registry RDAP queries using live data from the .cc and .tv country code top-level domains. Three access levels are provided based on the authenticated identity of the client:

1. Unauthenticated: Limited information is returned in response to queries from unauthenticated clients.
2. Basic: Clients who authenticate using a publicly available identity provider like Google Gmail or Microsoft Hotmail will receive all of the information available to an unauthenticated client plus additional registration metadata, but no personally identifiable information associated with entities.
3. Advanced: Clients who authenticate using a more restrictive identity provider will receive all of the information available to a Basic client plus whatever information the server operator deems appropriate for a fully authorized client. Currently supported identity providers include those developed by Verisign Labs (https://testprovider.rdap.verisignlabs.com/) and CZ.NIC (https://www.mojeid.cz/).

Level of Maturity: This is a "proof of concept" research implementation.
Coverage: This implementation includes all of the features described in this specification.
Contact Information: Scott Hollenbeck, shollenbeck@verisign.com

8. Security Considerations

Security considerations for RDAP can be found in RFC 7481 [RFC7481]. Security considerations for OpenID Connect Core [OIDCC] and OAuth 2.0 [RFC6749] can be found in their reference specifications. OpenID Connect defines optional mechanisms for robust signing and encryption that can be used to provide data integrity and data confidentiality services as needed. Security services for ID Tokens and Access Tokens (with references to the JWT specification) are described in the OpenID Connect Core protocol.

8.1. Authentication and Access Control

Having completed the client identification, authorization, and validation process, an RDAP server can make access control decisions based on a comparison of client-provided information and local policy. For example, a client who provides an email address (and nothing more) might be entitled to receive a subset of the information that would be available to a client who provides an email address, a full name, and a stated purpose. Development of these access control policies is beyond the scope of this document.

9. Acknowledgements

The author would like to acknowledge the following individuals for their contributions to the development of this document: Tom Harrison, Russ Housley, Rhys Smith, Jaromir Talir, and Alessandro

Vesely. In addition, the Verisign Registry Services Lab development team of Andrew Kaizer, Sai Mogali, Anurag Saxena, Swapneel Sheth, Nitin Singh, and Zhao Zhao provided critical "proof of concept" implementation experience that helped demonstrate the validity of the concepts described in this document.

10. References

10.1. Normative References

[OIDC] OpenID Foundation, "OpenID Connect", <http://openid.net/connect/>.

[OIDCC] OpenID Foundation, "OpenID Connect Core incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[OIDCD] OpenID Foundation, "OpenID Connect Discovery 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-discovery-1_0.html>.

[OIDCR] OpenID Foundation, "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-registration-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <http://www.rfc-editor.org/info/rfc7009>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", RFC 7480, DOI 10.17487/RFC7480, March 2015, <http://www.rfc-editor.org/info/rfc7480>.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", RFC 7481, DOI 10.17487/RFC7481, March 2015, <http://www.rfc-editor.org/info/rfc7481>.

[RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", RFC 7482, DOI 10.17487/RFC7482, March 2015, <http://www.rfc-editor.org/info/rfc7482>.

[RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, DOI 10.17487/RFC7483, March 2015, <http://www.rfc-editor.org/info/rfc7483>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

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

10.2. Informative References

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <http://www.rfc-editor.org/info/rfc7942>.

10.3. URIs

[1] http://openid.net/get-an-openid/

[2] https://www.icann.org/en/system/files/files/final-report-06jun14-en.pdf

[3] http://curl.haxx.se/

[4] https://www.gnu.org/software/wget/

Appendix A. Change Log

00: Initial version.
01: Updated flow description (Section 3.1.2) and description of the registration process (Section 3.1.3). Thanks to Jaromir Talir.
02: Updated flow description.
03: Added description of query parameters and non-browser clients. Updated security considerations to note issues associated with access control.
04: Updated references for JSON Web Token, OpenID Connect Core, and OpenID Connect Discovery. Added acknowledgement to the Verisign Labs developers. Changed intended status to Standards Track. Added text to describe protocol parameters and processing. Other minor edits.
05: Added examples for curl and wget. Added a reference to RFC 7235.
00: Changed WG reference in file name from weirds to regext. Described support for refresh tokens. Editorial updates. Corrected several examples. Added registry of purpose values.
01: Added Implementation Status section (Section 7).
02: Updated section Section 4.4 to clarify ID Token validation requirements.
03: Keepalive refresh.

Author's Address

Scott Hollenbeck
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
USA

Email: shollenbeck@verisign.com
URI: http://www.verisignlabs.com/

eppext
Internet-Draft
Intended status: Standards Track
Expires: December 2, 2016

H.W. Ribbers
M.W. Groeneweg
SIDN
R. Gieben

A.L.J Verschuren

May 31, 2016

Key Relay Mapping for the Extensible Provisioning Protocol
draft-ietf-eppext-keyrelay-12

Abstract

This document describes an Extensible Provisioning Protocol (EPP) mapping for a key relay object that relays DNSSEC key material between EPP clients using the poll queue defined in RFC5730.

This key relay mapping will help facilitate changing the DNS operator of a domain while keeping the DNSSEC chain of trust intact.

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 December 2, 2016.

Copyright Notice

Table of Contents

1. Introduction

There are certain transactions initiated by a DNS-operator that require an authenticated exchange of information between DNS-operators. Often, there is no direct channel between these parties or it is non-scalable and insecure.

One such transaction is the exchange of DNSSEC key material when changing the DNS operator for DNSSEC signed zones. We suggest that DNS-operators use the administrative EPP channel to bootstrap the delegation by relaying DNSSEC key material for the zone.

In this document we define an EPP extension to sent DNSSEC key material between EPP clients. This allows DNS operators to bootstrap automatically, reliable and securely the transfer of a domain name while keeping the DNSSEC chain of trust intact.

1.1. 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 BCP 14, RFC 2119 [RFC2119].

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

In examples, "C:" represents lines sent by a protocol client, and "S:" represents lines returned by a protocol server. Indentation and white space in examples is provided only to illustrate element relationships and is not a mandatory feature of this protocol.

1.2. Secure Transfer of DNSSEC Key Material

Exchanging DNSSEC key material in preparation of a domain name transfer is one of the phases in the lifecycle of a domain name [I-D.koch-dnsop-dnssec-operator-change].

DNS-operators need to exchange DNSSEC key material before the registration data can be changed to keep the DNSSEC chain of trust intact. This exchange is normally initiated through the gaining registrar.

The gaining and losing DNS operators could talk directly to each other (the ~ arrow in Figure 1) to exchange the DNSKEY, but often there is no trusted path between the two. As both can securely interact with the registry over the administrative channel through the registrar, the registry can act as a relay for the key material exchange.

The registry is merely used as a relay channel. Therefore it is up to the losing DNS-operator to complete the intended transaction. The registry SHOULD have certain policies in place that require the losing DNS operator to cooperate with this transaction, however this is beyond this document. This document focuses on the EPP protocol syntax.

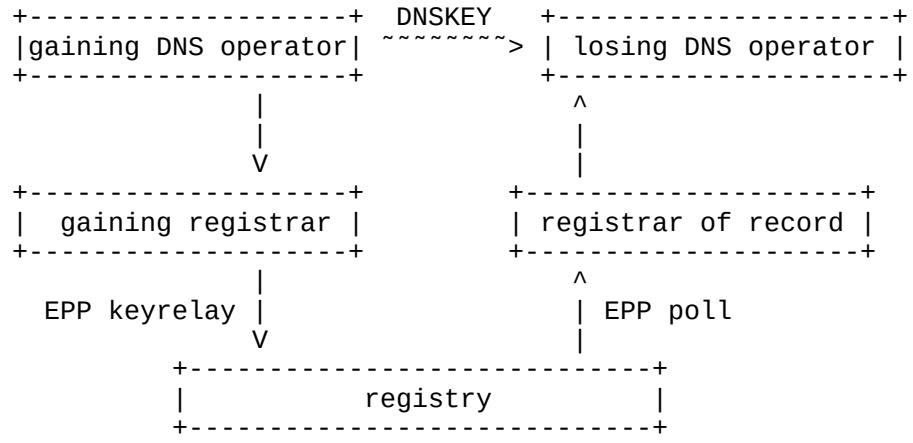

Figure 1: Transfer of DNSSEC key material.

There is no distinction in the EPP protocol between Registrars and DNS-operators, there is only mention of an EPP client and EPP server. Therefore the term EPP client will be used for the interaction with the EPP server for relaying DNSSEC key material.

2. Object Attributes

2.1. DNSSEC Key Material

The DNSSEC key material is represented in EPP by a <keyRelayData> element.

2.1.1. <keyRelayData> element

The <keyRelayData> contains the following elements:

- One REQUIRED <keyData> element that contains the DNSSEC key material as described in [RFC5910], Section 4

- An OPTIONAL <expiry> element that describes the expected lifetime of the relayed key(s) in the zone. When the <expiry> element is provided the losing DNS operator SHOULD remove the inserted key material from the zone after the expire time. This may be because the transaction that needed the insertion should either be completed or abandoned by that time. If a client receives a key relay object that has been sent previously it MUST update the expire time of the key material. This enables the clients to update the lifetime of the key material when a transfer is delayed.

The <expiry> element MUST contain exactly one of the following child elements:

* <absolute>: The DNSSEC key material is valid from the current date and time until it expires on the specified date and time. If a date in the past is provided this MUST be interpreted as a revocation of a previously sent key relay object.

* <relative>: The DNSSEC key material is valid from the current date and time until the end of the specified duration. If a period of zero is provided this MUST be interpreted as a revocation of a previously sent key relay object.

3. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in the EPP core protocol specification [RFC5730]. The command mapping described here is specifically for use in this key relay mapping.

3.1. EPP Query Commands

EPP provides three commands to retrieve object information: <check> to determine if an object is known to the server, <info> to retrieve

detailed information associated with an object, and <transfer> to retrieve object transfer status information.

3.1.1. EPP <check> Command

Check semantics do not apply to key relay objects, so there is no mapping defined for the EPP <check> command and the EPP <check> response.

3.1.2. EPP <info> Command

Info command semantics do not apply to the key relay objects, so there is no mapping defined for the EPP <info> Command.

The EPP <info> response for key relay objects is used in the EPP poll response, as described in [RFC5730]. The key relay object created with the <create> command, described in Section 3.2.1 is inserted into the receiving client's poll queue. The receiving client will receive the key relay object using the EPP <poll> command, as described in [RFC5730].

When a <poll> command has been processed successfully for a key relay poll message, the EPP <resData> element MUST contain a child <keyrelay:infData> element that is identified by the keyrelay namespace. The <keyrelay:infData> element contains the following child elements:

o A REQUIRED <name> element containing the domain name for which the DNSSEC key material is relayed.

o A REQUIRED <authInfo> element that contains authorization information associated with the domain object ([RFC5731], Section 3.2.1).

o One or more REQUIRED <keyRelayData> elements containing data to be relayed, as defined in Section 2.1. A server MAY apply a server policy that specifies the number of <keyRelayData> elements that can be incorporated. When a server policy is violated, a server MUST respond with an EPP result code 2308 "Data management policy violation".

o An OPTIONAL <crDate> element that contains the date and time of the submitted <create> command.

o An OPTIONAL <reID> element that contains the identifier of the client that requested the key relay.

o An OPTIONAL <acID> element that contains the identifier of the client that SHOULD act upon the key relay.

Example <poll> response:

```
S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
S:    xmlns:keyrelay="urn:ietf:params:xml:ns:keyrelay-1.0"
S:  xmlns:s="urn:ietf:params:xml:ns:secDNS-1.1"
S:  xmlns:d="urn:ietf:params:xml:ns:domain-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>1999-04-04T22:01:00.0Z</qDate>
S:      <msg>Keyrelay action completed successfully.</msg>
S:    </msgQ>
S:    <resData>
S:      <keyrelay:infData>
S:        <keyrelay:name>example.org</keyrelay:name>
S:        <keyrelay:authInfo>
S:          <d:pw>JnSdBAZSxxzJ</d:pw>
S:        </keyrelay:authInfo>
S:        <keyrelay:keyRelayData>
S:          <keyrelay:keyData>
S:            <s:flags>256</s:flags>
S:            <s:protocol>3</s:protocol>
S:            <s:alg>8</s:alg>
S:            <s:pubKey>cmlraXN0aGViZXN0</s:pubKey>
S:          </keyrelay:keyData>
S:          <keyrelay:expiry>
S:            <keyrelay:relative>P1M13D</keyrelay:relative>
S:          </keyrelay:expiry>
S:        </keyrelay:keyRelayData>
S:        <keyrelay:crDate>
S:          1999-04-04T22:01:00.0Z
S:        </keyrelay:crDate>
S:        <keyrelay:reID>
S:          ClientX
S:        </keyrelay:reID>
S:        <keyrelay:acID>
S:          ClientY
S:        </keyrelay:acID>
S:      </keyrelay:infData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-ZYX</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

3.1.3. EPP <transfer> Command

Transfer semantics do not apply to key relay objects, so there is no mapping defined for the EPP <transfer> command.

3.2. EPP Transform Commands

EPP provides five commands to transform objects: <create> to create an instance of an object, <delete> to delete an instance of an object, <renew> to extend the validity period of an object, <transfer> to manage object sponsorship changes, and <update> to change information associated with an object.

3.2.1. EPP <create> Command

The EPP <create> command provides a transform operation that allows a client to create a key relay object that includes the domain name and DNSSEC key material to be relayed. When the <create> command is validated, the server MUST insert an EPP <poll> message, using the key relay info response (See Section 3.1.2), in the receiving client's poll queue that belongs to the registrar on record of the provided domain name.

In addition to the standard EPP command elements, the <create> command MUST contain a <keyrelay:create> element that is identified by the keyrelay namespace. The <keyrelay:create> element contains the following child elements:

o A REQUIRED <keyrelay:name> element containing the domain name for which the DNSSEC key material is relayed.

o A REQUIRED <authInfo> element that contains authorization information associated with the domain object ([RFC5731], Section 3.2.1).

o One or more REQUIRED <keyrelay:keyRelayData> element containing data to be relayed, as defined in Section 2.1

Example <create> commands:

Note that in the provided example the second <keyrelay:keyRelayData> element has a period of zero and thus represents the revocation of a previously sent key relay object (see Section 2.1.1).

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0"
C:    xmlns:keyrelay="urn:ietf:params:xml:ns:keyrelay-1.0"
C:  xmlns:s="urn:ietf:params:xml:ns:secDNS-1.1"
C:  xmlns:d="urn:ietf:params:xml:ns:domain-1.0">
C:  <command>
C:    <create>
C:      <keyrelay:create>
C:        <keyrelay:name>example.org</keyrelay:name>
C:        <keyrelay:authInfo>
C:          <d:pw>JnSdBAZSxxzJ</d:pw>
C:        </keyrelay:authInfo>
C:        <keyrelay:keyRelayData>
C:          <keyrelay:keyData>
C:            <s:flags>256</s:flags>
C:            <s:protocol>3</s:protocol>
C:            <s:alg>8</s:alg>
C:            <s:pubKey>cmlraXN0aGViZXN0</s:pubKey>
C:          </keyrelay:keyData>
C:          <keyrelay:expiry>
C:            <keyrelay:relative>P1M13D</keyrelay:relative>
C:          </keyrelay:expiry>
C:        </keyrelay:keyRelayData>
C:        <keyrelay:keyRelayData>
C:          <keyrelay:keyData>
C:            <s:flags>256</s:flags>
C:            <s:protocol>3</s:protocol>
C:            <s:alg>8</s:alg>
C:            <s:pubKey>bWFyY2lzdGhlYmVzdA==</s:pubKey>
C:          </keyrelay:keyData>
C:          <keyrelay:expiry>
C:            <keyrelay:relative>P0D</keyrelay:relative>
C:          </keyrelay:expiry>
C:        </keyrelay:keyRelayData>
C:      </keyrelay:create>
C:    </create>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a server has succesfully processed the <create> command it MUST respond with a standard EPP response. See [RFC5730], Section 2.6.

Example <create> response:

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

When a server cannot process the <create> command due to the server policy it MUST return an EPP 2308 error message. This might be the case when the server knows that the receiving client does not support keyrelay transactions. See [RFC5730], Section 2.6.

Example <create> response:

```
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="2308">
S:      <msg>Data management policy violation</msg>
S:    </result>
S:    <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54321-ZYX</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

3.2.2. EPP <delete> Command

Delete semantics do not apply to key relay objects, so there is no mapping defined for the EPP <delete> command and the EPP <delete> response.

3.2.3. EPP <renew> Command

Renew semantics do not apply to key relay objects, so there is no mapping defined for the EPP <renew> command and the EPP <renew> response.

3.2.4. EPP <transfer> Command

Transfer semantics do not apply to key relay objects, so there is no mapping defined for the EPP <transfer> command and the EPP <transfer> response.

3.2.5. EPP <update> Command

Update semantics do not apply to key relay objects, so there is no mapping defined for the EPP <update> command and the EPP <update> response.

4. Formal Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:ietf:params:xml:ns:keyrelay-1.0"
  xmlns:keyrelay="urn:ietf:params:xml:ns:keyrelay-1.0"
  xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
  xmlns:secDNS="urn:ietf:params:xml:ns:secDNS-1.1"
  xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Extensible Provisioning Protocol v1.0 protocol
      extension schema for relaying DNSSEC key material.
    </documentation>
  </annotation>

  <import namespace="urn:ietf:params:xml:ns:eppcom-1.0" />
  <import namespace="urn:ietf:params:xml:ns:secDNS-1.1" />
  <import namespace="urn:ietf:params:xml:ns:domain-1.0" />

  <element name="keyRelayData" type="keyrelay:keyRelayDataType" />
  <element name="infData" type="keyrelay:infDataType" />
  <element name="create" type="keyrelay:createType" />

  <complexType name="createType">
    <sequence>
      <element name="name" type="eppcom:labelType" />
      <element name="authInfo" type="domain:authInfoType" />
      <element name="keyRelayData" type="keyrelay:keyRelayDataType"
          maxOccurs="unbounded"/>
    </sequence>
  </complexType>

 <complexType name="infDataType">
```

```
      <sequence>
        <element name="name" type="eppcom:labelType" />
        <element name="authInfo" type="domain:authInfoType" />
        <element name="keyRelayData" type="keyrelay:keyRelayDataType"
            maxOccurs="unbounded"/>
        <element name="crDate" type="dateTime"/>
        <element name="reID" type="eppcom:clIDType" />
        <element name="acID" type="eppcom:clIDType" />
      </sequence>
    </complexType>

    <complexType name="keyRelayDataType">
      <sequence>
        <element name="keyData" type="secDNS:keyDataType" />
        <element name="expiry" type="keyrelay:keyRelayExpiryType"
            minOccurs="0" />
      </sequence>
    </complexType>

    <complexType name="keyRelayExpiryType">
      <choice>
        <element name="absolute" type="dateTime" />
        <element name="relative" type="duration" />
      </choice>
    </complexType>
  </schema>
```

5. IANA Considerations

5.1. XML Namespace

This document uses URNs to describe a XML namespace conforming to the registry mechanism described in [RFC3688]. The following URI assignment is requested of IANA:

URI: urn:ietf:params:xml:ns:keyrelay-1.0

Registrant Contact: See the "Author’s Address" section of this document.

XML: See the "Formal Syntax" section of this document.

5.2. XML Schema

This document uses URNs to describe a XML schema conforming to the registry mechanism described in [RFC3688]. The following URI assignment is requested of IANA:

URI: urn:ietf:params:xml:schema:keyrelay-1.0

XML: See the "Formal Syntax" section of this document.

5.3. EPP Extension Registry

The EPP extension described in this document should be registered by the IANA in the EPP Extension Registry described in [RFC7451]. The details of the registration are as follows:

Name of Extension: "Key Relay Mapping for the Extensible Provisioning Protocol"

Document status: Standards Track

Reference: (insert reference to RFC version of this document)

Registrant Name and Email Address: IESG, iesg@ietf.org

TLDs: Any

IPR Disclosure: https://datatracker.ietf.org/ipr/2393/

Status: Active

Notes: None

6. Security Considerations

A server SHOULD NOT perform any transformation on data under server management when processing a <keyrelay:create> command. The intent of this command is to put DNSSEC key material on the poll queue of another client. To make sure that this EPP extension is interoperable with the different server policies that already have implemented EPP this extension it is not classified as must not.

Any EPP client can use this mechanism to put data on the message queue of another EPP client, allowing for the potential of a denial of service attack. However this can, and should be detected by the server. A server MAY set a server policy which limits or rejects a <keyrelay:create> command if it detects the mechanism is being abused.

For the <keyrelay:keyRelayData> data a correct <domain:authInfo> element should be used as an indication that putting the key material on the receiving EPP clients poll queue is authorized by the _registrant_ of that domain name. The authorization of EPP clients

to perform DNS changes is not covered in this document as it depends on registry specific policy.

A client that uses this mechanism to send DNSSEC key material to another client could verify through DNS that the DNSSEC key material is added to the authoritive zone of the domain. This check can be used to verify that the DNSSEC key material has traveled end-to-end from the gaining DNS operator to the losing DNS operator. This check does not tell anything about the DNSSEC chain of trust and can merely be used as a verification of a succesful transfer of the DNSSEC key material.

7. Acknowledgements

We like to thank the following individuals for their valuable input, review, constructive criticism in earlier revisions or support for the concepts described in this document:

Maarten Wullink, Marco Davids, Ed Lewis, James Mitchell, David Peal, Patrik Faltstrom, Klaus Malorny, James Gould, Patrick Mevzek, Seth Goldman, Maarten Bosteels, Ulrich Wisser, Kees Monshouwer and Scott Hollenbeck.

8. References

8.1. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[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.

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

8.2. Informative References

[I-D.koch-dnsop-dnssec-operator-change]
Koch, P., Sanz, M., and A. Verschuren, "Changing DNS Operators for DNSSEC signed Zones", draft-koch-dnsop-dnssec-operator-change-06 (work in progress), February 2014.

[RFC7451] Hollenbeck, S., "Extension Registry for the Extensible Provisioning Protocol", RFC 7451, February 2015.

Appendix A. Changelog

[This section should be removed by the RFC editor before publishing]

A.1. draft-gieben-epp-keyrelay-00

1. Initial document.

A.2. draft-gieben-epp-keyrelay-01

1. Style and grammar changes;
2. Added an expire element as per suggestion by Klaus Malorny;
3. Make the authInfo element mandatory and make the registry check it as per feedback by Klaus Malorny and James Gould.

A.3. draft-gieben-epp-keyrelay-02

1. Added element to identify the relaying EPP client as suggested by Klaus Malorny;
2. Corrected XML for missing and excess clTRID as noted by Patrick Mevzek;
3. Added clarifications for the examples based on feedback by Patrick Mevzeck;
4. Reviewed the consistency of using DNS operator versus registrar after review comments by Patrick Faltstrom and Ed Lewis.

A.4. draft-gieben-epp-keyrelay-03

1. Style and grammar changes
2. Corrected acknowledgement section
3. Corrected XML for Expire element to not be mandatory but only occur once.

A.5. draft-ietf-eppext-keyrelay-00

1. Added feedback from Seth Goldman and put him in the acknowledgement section.

2. IDnits formatting ajustments

A.6. draft-ietf-eppext-keyrelay-01

1. Introducing the <relay> command, and thus separating the data and the command.

2. Updated the Introduction, describing the general use of relay vs the intended use-case of relaying DNSSEC key data.

3. Restructuring the document to make it more inline with existing EPP extensions.

A.7. draft-ietf-eppext-keyrelay-02

1. Updated the XML structure by removing the <relay> command based on WG feedback

2. Updated the wording

A.8. draft-ietf-eppext-keyrelay-03

1. Updated the document title in the EPP Extension Registry section

2. Restored Acknowledgement section, thanks to Marco Davids

3. Incorperated feedback from Patrick Mevzek

A.9. draft-ietf-eppext-keyrelay-04

1. Incorperated feedback from James Gould

2. Added additional text when server is aware that receiving clients do not support keyrelay transactions or DNSSEC as suggested by Kees Monshouwer.

3. Added additional text for supporting key revocation as suggested by Kees Monshouwer

4. Updated some of the wording

5. Fix the usage of multiple keys in a create message

A.10. draft-ietf-eppext-keyrelay-05

1. Review comments after WG last call

A.11. draft-ietf-eppext-keyrelay-06

1. Review comments by Ulrich Wisser during IESG writeup

A.12. draft-ietf-eppext-keyrelay-07

1. fixed changelog

A.13. draft-ietf-eppext-keyrelay-08

1. fixed issue with authinfo

2. fixed issue with relative period in example xml

A.14. draft-ietf-eppext-keyrelay-09

1. fixed issue with naming

A.15. draft-ietf-eppext-keyrelay-10

1. removed 4 spaces

A.16. draft-ietf-eppext-keyrelay-11

1. Processed editorial changes from AD review

2. Processed comments made during IETF last call

A.17. draft-ietf-regext-keyrelay-00

1. Processed comments made during IESG review

Authors' Addresses

Rik Ribbers
SIDN
Meander 501
Arnhem 6825 MD
NL

Email: rik.ribbers@sidn.nl
URI: https://www.sidn.nl/

Marc Groeneweg
SIDN
Meander 501
Arnhem 6825 MD
NL

Email: marc.groeneweg@sidn.nl
URI: https://www.sidn.nl/

Miek Gieben

Email: miek@miek.nl

Antoin Verschuren

Email: ietf@antoin.nl

Registration Protocols Extensions
Internet-Draft
Intended status: Standards Track
Expires: December 25, 2017

R. Carney
GoDaddy Inc.
G. Brown
CentralNic Group plc
J. Frakes

June 23, 2017

Registry Fee Extension for the Extensible Provisioning Protocol (EPP)
draft-ietf-regext-epp-fees-05

Abstract

This document describes an Extensible Provisioning Protocol (EPP) extension mapping for registry fees.

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 December 25, 2017.

Copyright Notice

Table of Contents

1. Introduction

Historically, domain name registries have applied a simple fee structure for billable transactions, namely a basic unit price applied to domain <create>, <renew>, <transfer> and RGP [RFC3915] restore commands. Given the relatively small number of EPP servers to which EPP clients have been required to connect, it has generally been the case that client operators have been able to obtain details of these fees out-of-band by contacting the server operators.

Given the recent expansion of the DNS namespace, and the proliferation of novel business models, it is now desirable to provide a method for EPP clients to query EPP servers for the fees and credits associated with certain commands and specific objects.

This document describes an extension mapping for version 1.0 of the Extensible Provisioning Protocol (EPP) [RFC5730]. This EPP mapping provides a mechanism by which EPP clients may query the fees and credits associated with various billable transactions, and obtain their current account balance.

1.1. 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 RFC 2119 [RFC2119].

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

"fee" is used as an abbreviation for "urn:ietf:params:xml:ns:fee-0.21". The XML namespace prefix "fee" is used, but implementations MUST NOT depend on it and instead employ a proper namespace-aware XML parser and serializer to interpret and output the XML documents.

In examples, "C:" represents lines sent by a protocol client and "S:" represents lines returned by a protocol server. Indentation and white space in examples are provided only to illustrate element relationships and are not a REQUIRED feature of this protocol.

(Note to RFC Editor: remove the following paragraph before publication as an RFC.)

The XML namespace prefix above contains a version number, specifically "0.21". This version number will increment with successive versions of this document, and will reach 1.0 if and when this document is published as an RFC. This permits clients to distinguish which version of the extension a server has implemented.

2. Migrating to Newer Versions of This Extension

(Note to RFC Editor: remove this section before publication as an RFC.)

Servers which implement this extension SHOULD provide a way for clients to progressively update their implementations when a new version of the extension is deployed.

Servers SHOULD (for a temporary migration period) provide support for older versions of the extension in parallel to the newest version, and allow clients to select their preferred version via the <svcExtension> element of the <login> command.

If a client requests multiple versions of the extension at login, then, when preparing responses to commands which do not include extension elements, the server SHOULD only include extension elements in the namespace of the newest version of the extension requested by the client.

When preparing responses to commands which do include extension elements, the server SHOULD only include extension elements for the extension versions present in the command.

3. Extension Elements

3.1. Client Commands

The <fee:command> element is used in the EPP <check> command to determine the fee which is applicable to the given command.

The list of values include:

- "create" indicating a <create> command as defined in [RFC5730];
- "delete" indicating a <delete> command as defined in [RFC5730];
- "renew" indicating a <renew> command as defined in [RFC5730];
- "update" indicating a <update> command as defined in [RFC5730];
- "transfer" indicating a <transfer> command as defined in [RFC5730];
- If the server supports the Registry Grace Period Mapping [RFC3915], then the server MUST also support the "restore" value as defined in [RFC3915];

- o "custom" indicating a custom command that uses the customName attribute to define the custom operation.

The <fee:command> element MAY have an OPTIONAL "phase" attribute specifying a launch phase as described in [draft-ietf-eppext-launchphase]. It may also contain an OPTIONAL "subphase" attribute identifying the custom or sub-phase as described in [draft-ietf-eppext-launchphase].

3.2. Currency Codes

The <fee:currency> element is used to indicate which currency fees are charged in. This value of this element MUST be a three-character currency code from [ISO4217].

Note that ISO 4217 provides the special "XXX" code, which MAY be used if the server uses a non-currency based system for assessing fees, such as a system of credits.

The use of <fee:currency> elements in client commands is OPTIONAL: if a <fee:currency> element is not present in a command, the server MUST determine the currency based on the client's account settings which MUST be agreed by the client and server via an out-of-band channel. However, the <fee:currency> element MUST be present in responses.

Servers SHOULD NOT perform a currency conversion if a client uses an incorrect currency code. Servers SHOULD return a 2004 "Parameter value range" error instead.

3.3. Validity Periods

When querying for fee information using the <check> command, the <fee:period> element is used to indicate the units to be added to the registration period of objects by the <create>, <renew> and <transfer> commands. This element is derived from the <domain:period> element described in [RFC5731].

The <fee:period> element is OPTIONAL in <check> commands, if omitted, the server MUST determine the fee(s) using the server default period. The <fee:period> element MUST be present in <check> responses.

3.4. Fees and Credits

Servers which implement this extension will include elements in responses which provide information about the fees and/or credits associated with a given billable transaction.

The <fee:fee> and <fee:credit> elements are used to provide this information. The presence of a <fee:fee> element in a response indicates a debit against the client's account balance; a <fee:credit> element indicates a credit. A <fee:fee> element MUST have a non-negative value. A <fee:credit> element MUST have a negative value.

A server MAY respond with multiple <fee:fee> and <fee:credit> elements in the same response. In such cases, the net fee or credit applicable to the transaction is the arithmetic sum of the values of each of the <fee:fee> and/or <fee:credit> elements. This amount applies to the total additional validity period applied to the object (where applicable) rather than to any incremental unit.

The following attributes are defined for the <fee:fee> element. These are described in detail below:

description: an OPTIONAL attribute which provides a human-readable description of the fee. Servers should provide documentation on the possible values of this attribute, and their meanings.

refundable: an OPTIONAL boolean attribute indicating whether the fee is refundable if the object is deleted.

grace-period: an OPTIONAL attribute which provides the time period during which the fee is refundable.

applied: an OPTIONAL attribute indicating when the fee will be deducted from the client's account.

The <fee:credit> element can take a "description" attribute as described above. No other attributes are defined for this element.

3.4.1. Refunds

<fee:fee> elements MAY have an OPTIONAL "refundable" attribute which takes a boolean value. Fees may be refunded under certain circumstances, such as when a domain application is rejected (as described in [draft-ietf-eppext-launchphase]) or when an object is deleted during the relevant Grace Period (see below).

If the "refundable" attribute is omitted, then clients SHOULD NOT make any assumption about the refundability of the fee.

3.4.2. Grace Periods

[RFC3915] describes a system of "grace periods", which are time periods following a billable transaction during which, if an object is deleted, the client receives a refund.

The "grace-period" attribute MAY be used to indicate the relevant grace period for a fee. If a server implements the Registry Grace Period extension, it MUST specify the grace period for all relevant transactions.

If the "grace-period" attribute is omitted, then clients SHOULD NOT make any assumption about the grace period of the fee.

3.4.3. Correlation between Refundability and Grace Periods

If a <fee:fee> element has a "grace-period" attribute then it MUST also be refundable. If the "refundable" attribute of a <fee:fee> element is false then it MUST NOT have a "grace-period" attribute.

3.4.4. Applicability

Fees may be applied immediately upon receipt of a command from a client, or may only be applied once an out-of-band process (such as the processing of applications at the end of a launch phase) has taken place.

The "applied" attribute of the <fee:fee> element allows servers to indicate whether a fee will be applied immediately, or whether it will be applied at some point in the future. This attribute takes two possible values: "immediate" or "delayed".

3.5. Account Balance

The <fee:balance> element is an OPTIONAL element which MAY be included in server responses to transform commands. If present, it can be used by the client to determine the remaining credit at the server.

Whether or not the <fee:balance> is included in responses is a matter of server policy. However, if a server chooses to offer support for this element, it MUST be included in responses to all "transform" or billable commands (ie <create>, <renew>, <update>, <delete>, <transfer op="request">).

The value of the <fee:balance> MAY be negative. A negative balance indicates that the server has extended a line of credit to the client (see below).

If a server includes a <fee:balance> element in response to transform commands, the value of the element MUST reflect the client's account balance after any fees or credits associated with that command have been applied.

3.6. Credit Limit

As described above, if a server returns a response containing a <fee:balance> with a negative value, then the server has extended a line of credit to the client. A server MAY also include a <fee:creditLimit> element in responses which indicates the maximum credit available to a client. A server MAY reject certain transactions if the absolute value of the <fee:balance> is equal to or exceeds the value of the <fee:creditLimit> element.

Whether or not the <fee:creditLimit> is included in responses is a matter of server policy. However, if a server chooses to offer support for this element, it MUST be included in responses to all "transform" commands (ie <create>, <renew>, <update>, <delete>, <transfer op="request">).

3.7. Classification of Objects

Objects may be assigned to a particular class, category, or tier, each of which has a particular fee or set of fees associated with it. The <fee:class> element, which MAY appear in <check> and transform responses, is used to indicate the classification of an object.

If a server makes use of this element, it should provide clients with a list of all the values that the element may take via an out-of-band channel. Servers MUST NOT use values which do not appear on this list.

Servers which make use of this element MUST use a <fee:class> element with the value "standard" for all objects that are subject to the standard or default fee.

3.8. Phase and Subphase Attributes

The <fee:command> element has two attributes, phase and subphase, that provide additional information related to a specific launch phase as described in [draft-ietf-eppext-launchphase].

If the client <fee:command> contains no phase attribute the server SHOULD return data (including the phase/subphase attribute(s)) for all valid server combinations of currently active phase(s) and active subphase(s).

If the client <fee:command> contains a server supported combination of phase/subphase the server MUST return fee data (including the phase/subphase attribute(s)) for the specific combination(s).

If the client <fee:command> contains a phase attribute with no subphase, the server SHOULD return data (including the phase/subphase attribute(s)) for all valid server combinations of provided phase and available subphase(s).

If the client <fee:command> contains a subphase with no phase attribute the server MUST respond with a 2003 "Required parameter missing" error.

If the client <fee:command> contains a phase attribute not defined in [draft-ietf-eppext-launchphase] or not supported by server the server MUST respond with a 2004 "Parameter value range" error.

If the client <fee:command> contains a subphase attribute (or phase/subphase combination) not supported by server the server MUST respond with a 2004 "Parameter value range" error.

3.9. Reason

The <fee:reason> element is used to provide server specific text in an effort to better explain why an operation did not complete as the client expected. An OPTIONAL "lang" attribute MAY be present to identify the language of the returned text.

4. Server Handling of Fee Information

Depending on server policy, a client MAY be required to include the extension elements described in this document for certain transform commands. Servers must provide clear documentation to clients about the circumstances in which this extension must be used.

The server MUST return avail="0" in its response to a <check> command for any domain name in the <check> command that does not include the <fee:check> extension for which the server would likewise fail a domain <create> command when no <fee> extension is provided for that same domain name.

If a server receives a command from a client which does not include the fee extension data elements required by the server for that command, then the server MUST respond with a 2003 "Required parameter missing" error.

If the currency or total fee provided by the client is less than the server's own calculation of the fee for that command, then the server MUST reject the command with a 2004 "Parameter value range" error.

5. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in [RFC5730].

5.1. EPP Query Commands

This extension does not add any elements to the EPP <poll> or <info> commands or responses.

5.1.1. EPP <check> Command

This extension defines a new command called the Fee Check Command that defines additional elements for the EPP <check> command to provide fee information along with the availability information of the EPP <check> command.

The command MAY contain an <extension> element which MAY contain a <fee:check> element. The <fee:check> element MAY contain one <fee:currency> element and MUST contain one or more <fee:command> elements.

The <fee:command> element(s) contain(s) a "name" attribute, an OPTIONAL "phase" attribute, and an OPTIONAL "subphase" attribute. The <fee:command> element(s) MAY have the following child elements:

- An OPTIONAL <fee:period> element (as described in Section 3.3).

Example <check> command:

```
C: <?xml version="1.0" encoding="utf-8" standalone="no"?>
C: <epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:   <command>
C:     <check>
C:       <domain:check
C:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:         <domain:name>example.com</domain:name>
C:         <domain:name>example.net</domain:name>
C:         <domain:name>example.xyz</domain:name>
C:       </domain:check>
C:     </check>
C:     <extension>
C:       <fee:check xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
C:         <fee:currency>USD</fee:currency>
C:         <fee:command name="create">
C:           <fee:period unit="y">2</fee:period>
C:         </fee:command>
C:         <fee:command name="renew"/>
C:         <fee:command name="transfer"/>
C:         <fee:command name="restore"/>
C:       </fee:check>
C:     </extension>
C:     <clTRID>ABC-12345</clTRID>
C:   </command>
C: </epp>
```

When the server receives a <check> command that includes the extension elements described above, its response MUST contain an <extension> element, which MUST contain a child <fee:chkData> element. The <fee:chkData> element MUST contain a <fee:currency> element and a <fee:cd> for each element referenced in the client <check> command.

Each <fee:cd> element MUST contain the following child elements:

- A <fee:objID> element, which MUST match an element referenced in the client <check> command.
- A <fee:command> element matching each <fee:command> (unless the "avail" attribute of the <fee:cd> if false) that appeared in the corresponding <fee:check> of the client command. This element MAY have the OPTIONAL "phase" and "subphase" attributes, which MUST match the same attributes in the corresponding <fee:command> element of the client command.

The <fee:cd> element also has an OPTIONAL "avail" attribute which is a boolean. If the value of this attribute evaluates to false, this

indicates that the server cannot calculate the relevant fees, because the object, command, currency, period, class or some combination is invalid per server policy. If "avail" is false then the <fee:cd> element MUST contain a <fee:reason> element (as described in Section 3.9) and the server MAY eliminate some or all of the <fee:command> element(s).

The <fee:command> element(s) MAY have the following child elements:

- An OPTIONAL <fee:period> element (as described in Section 3.3), which contains the same unit that appeared in the <fee:period> element of the command. If the value of the preceding <fee:command> element is "restore", this element MUST NOT be included, otherwise it MUST be included. If no <fee:period> appeared in the client command (and the command is not "restore") then the server MUST return its default period value.
- Zero or more <fee:fee> elements (as described in Section 3.4).
- Zero or more <fee:credit> elements (as described in Section 3.4).
- An OPTIONAL <fee:class> element (as described in Section 3.7).
- An OPTIONAL <fee:reason> element (as described in Section 3.9).

If the "avail" attribute of the <fee:cd> element is true and if no <fee:fee> elements are present in a <fee:command> element, this indicates that no fee will be assessed by the server for this command.

If the "avail" attribute is true, then the <fee:command> element MUST NOT contain a <fee:reason> element.

Example <check> response:

```
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:chkData
S:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:         <domain:cd>
S:           <domain:name avail="1">example.com</domain:name>
S:         </domain:cd>
S:         <domain:cd>
S:           <domain:name avail="1">example.net</domain:name>
S:         </domain:cd>
S:         <domain:cd>
S:           <domain:name avail="1">example.xyz</domain:name>
```

```
S:         </domain:cd>
S:       </domain:chkData>
S:     </resData>
S:     <extension>
S:       <fee:chkData
S:           xmlns:fee="urn:ietf:params:xml:ns:fee-0.21"
S:           xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:         <fee:currency>USD</fee:currency>
S:         <fee:cd avail="1">
S:           <fee:objID>example.com</fee:objID>
S:           <fee:command name="create">
S:             <fee:period unit="y">2</fee:period>
S:             <fee:fee
S:               description="Registration Fee"
S:               refundable="1"
S:               grace-period="P5D">10.00</fee:fee>
S:           </fee:command>
S:           <fee:command name="renew">
S:             <fee:period unit="y">1</fee:period>
S:             <fee:fee
S:               description="Renewal Fee"
S:               refundable="1"
S:               grace-period="P5D">5.00</fee:fee>
S:           </fee:command>
S:           <fee:command name="transfer">
S:             <fee:period unit="y">1</fee:period>
S:             <fee:fee
S:               description="Transfer Fee"
S:               refundable="1"
S:               grace-period="P5D">5.00</fee:fee>
S:           </fee:command>
S:           <fee:command name="restore">
S:             <fee:fee
S:               description="Redemption Fee">5.00</fee:fee>
S:           </fee:command>
S:         </fee:cd>
S:         <fee:cd avail="1">
S:           <fee:objID>example.net</fee:objID>
S:           <fee:command name="create">
S:             <fee:period unit="y">2</fee:period>
S:             <fee:fee
S:               description="Registration Fee"
S:               refundable="1"
S:               grace-period="P5D">10.00</fee:fee>
S:           </fee:command>
S:           <fee:command name="renew">
S:             <fee:period unit="y">1</fee:period>
S:             <fee:fee
```

```
S:                description="Renewal Fee"
S:                refundable="1"
S:                grace-period="P5D">5.00</fee:fee>
S:            </fee:command>
S:            <fee:command name="transfer">
S:              <fee:period unit="y">1</fee:period>
S:              <fee:fee
S:                description="Transfer Fee"
S:                refundable="1"
S:                grace-period="P5D">5.00</fee:fee>
S:            </fee:command>
S:            <fee:command name="restore">
S:              <fee:fee
S:                description="Redemption Fee">5.00</fee:fee>
S:            </fee:command>
S:          </fee:cd>
S:          <fee:cd avail="0">
S:            <fee:objID>example.xyz</fee:objID>
S:            <fee:command name="create">
S:              <fee:period unit="y">2</fee:period>
S:              <fee:reason>Only 1 year registration periods are
S:                vaild.</fee:reason>
S:            </fee:command>
S:          </fee:cd>
S:        </fee:chkData>
S:      </extension>
S:      <trID>
S:        <clTRID>ABC-12345</clTRID>
S:        <svTRID>54322-XYZ</svTRID>
S:      </trID>
S:    </response>
S: </epp>
```

5.1.1.1. Server Handling of Elements

Clients MAY include a <fee:class> in the <fee:command> element. There are two ways in which servers may handle this element:

1. If the server supports the concept of tiers or classes of objects, then the value of this element MUST be validated. If incorrect for the specified object, the "avail" attribute of the corresponding <fee:cd> element MUST be false.
2. If the server supports different "types" of object registrations (such as a "blocking" registration which does not resolve, or where a registry provides a value-added service that requires an opt-out to disable), then, as with the first model, the server MUST validate the value of the element. If the value is incorrect, the "avail" attribute of the corresponding <fee:cd>

element MUST be false, and a <fee:reason> element (as described in Section 3.9) MUST be provided.

Server operators must provide clear documentation to client operators which of the above models it supports.

5.1.2. EPP Transfer Query Command

This extension does not add any elements to the EPP <transfer> query command, but does include elements in the response, when the extension has been selected during a <login> command.

When the <transfer> query command has been processed successfully, the client selected the extension when it logged in, and the client is authorized by the server to view information about the transfer, the server MAY include in the <extension> section of the EPP response a <fee:trnData> element, which contains the following child elements:

- A <fee:currency> element (as described in Section 3.2).
- A <fee:period> element (as described in Section 3.3).
- Zero or more <fee:fee> elements (as described in Section 3.4) containing the fees that will be charged to the gaining client.
- Zero or more <fee:credit> elements (as described in Section 3.4) containing the credits that will be refunded to the losing client.

Servers SHOULD omit <fee:credit> when returning a response to the gaining client, and omit <fee:fee> elements when returning a response to the losing client.

If no <fee:trnData> element is included in the response, then no fee will be assessed by the server for the transfer.

Example <transfer> query response:

```
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:       <domain:trnData
S:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:         <domain:name>example.com</domain:name>
S:         <domain:trStatus>pending</domain:trStatus>
S:         <domain:reID>ClientX</domain:reID>
S:         <domain:reDate>2000-06-08T22:00:00.0Z</domain:reDate>
S:         <domain:acID>ClientY</domain:acID>
S:         <domain:acDate>2000-06-13T22:00:00.0Z</domain:acDate>
S:         <domain:exDate>2002-09-08T22:00:00.0Z</domain:exDate>
S:       </domain:trnData>
S:     </resData>
S:     <extension>
S:       <fee:trnData xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
S:         <fee:currency>USD</fee:currency>
S:         <fee:period unit="y">1</fee:period>
S:         <fee:fee>5.00</fee:fee>
S:       </fee:trnData>
S:     </extension>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54322-XYZ</svTRID>
S:     </trID>
S:   </response>
S: </epp>
```

5.2. EPP Transform Commands

5.2.1. EPP <create> Command

This extension adds elements to both the EPP <create> command and response, when the extension has been selected during a <login> command.

If a <create> command is received with no <fee:create> extension and the server requires a <fee:create> extension for the <domain:name> the server MUST respond with a 2003 "Required parameter missing" error.

When submitting a <create> command to the server, the client MAY include in the <extension> element a <fee:create> element which includes the following child elements:

- An OPTIONAL <fee:currency> element (as described in Section 3.2);
- One or more <fee:fee> elements (as described in Section 3.4).

The server MUST fail the <create> command if the <fee:fee> provided by the client is less than the server fee.

When the <create> command has been processed successfully, and the client selected the extension when it logged in, and a fee was assessed by the server for the transaction, the server MUST include in the <extension> section of the EPP response a <fee:creData> element, which contains the following child elements:

- A <fee:currency> element (as described in Section 3.2);
- Zero or more <fee:fee> elements (as described in Section 3.4);
- Zero or more <fee:credit> elements (as described in Section 3.4);
- An OPTIONAL <fee:balance> element (as described in Section 3.5);
- An OPTIONAL <fee:creditLimit> element (as described in Section 3.6).

Example <create> command:

```
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.com</domain:name>
C:         <domain:period unit="y">2</domain:period>
C:         <domain:ns>
C:           <domain:hostObj>ns1.example.net</domain:hostObj>
C:           <domain:hostObj>ns2.example.net</domain:hostObj>
C:         </domain:ns>
C:         <domain:registrant>jd1234</domain:registrant>
C:         <domain:contact type="admin">sh8013</domain:contact>
C:         <domain:contact type="tech">sh8013</domain:contact>
C:         <domain:authInfo>
C:           <domain:pw>2fooBAR</domain:pw>
C:         </domain:authInfo>
C:       </domain:create>
C:     </create>
C:     <extension>
C:       <fee:create xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
C:         <fee:currency>USD</fee:currency>
C:         <fee:fee>5.00</fee:fee>
C:       </fee:create>
C:     </extension>
C:     <clTRID>ABC-12345</clTRID>
C:   </command>
C: </epp>
```

Example <create> response:

```
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:creData
S:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:         <domain:name>example.com</domain:name>
S:         <domain:crDate>1999-04-03T22:00:00.0Z</domain:crDate>
S:         <domain:exDate>2001-04-03T22:00:00.0Z</domain:exDate>
S:       </domain:creData>
S:     </resData>
S:     <extension>
S:       <fee:creData xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
S:         <fee:currency>USD</fee:currency>
S:         <fee:fee
S:           description="Registration Fee"
S:           refundable="1"
S:           grace-period="P5D">5.00</fee:fee>
S:         <fee:balance>-5.00</fee:balance>
S:         <fee:creditLimit>1000.00</fee:creditLimit>
S:       </fee:creData>
S:     </extension>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54321-XYZ</svTRID>
S:     </trID>
S:   </response>
S: </epp>
```

5.2.2. EPP <delete> Command

This extension does not add any elements to the EPP <delete> command, but does include elements in the response, when the extension has been selected during the <login> command.

When the <delete> command has been processed successfully, and the client selected the extension when it logged in, the server MAY include in the <extension> section of the EPP response a <fee:delData> element, which contains the following child elements:

- A <fee:currency> element (as described in Section 3.2);
- Zero or more <fee:fee> elements (as described in Section 3.4);
- Zero or more <fee:credit> elements (as described in Section 3.4);

- An OPTIONAL <fee:balance> element (as described in Section 3.5);
- An OPTIONAL <fee:creditLimit> element (as described in Section 3.6).

Example <delete> response:

```
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:     <extension>
S:       <fee:delData
S:         xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
S:         <fee:currency>USD</fee:currency>
S:         <fee:credit description="AGP Credit">-5.00</fee:credit>
S:         <fee:balance>1005.00</fee:balance>
S:       </fee:delData>
S:     </extension>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54321-XYZ</svTRID>
S:     </trID>
S:   </response>
S: </epp>
```

5.2.3. EPP <renew> Command

This extension adds elements to both the EPP <renew> command and response, when the extension has been selected during a <login> command.

When submitting a <renew> command to the server, the client MAY include in the <extension> element a <fee:renew> element which includes the following child elements:

- An OPTIONAL <fee:currency> element (as described in Section 3.2);
- One or more <fee:fee> elements (as described in Section 3.4).

When the <renew> command has been processed successfully, and the client selected the extension when it logged in, the server MAY include in the <extension> section of the EPP response a <fee:renData> element, which contains the following child elements:

- A <fee:currency> element (as described in Section 3.2);
- Zero or more <fee:fee> elements (as described in Section 3.4);
- Zero or more <fee:credit> elements (as described in Section 3.4);

- An OPTIONAL <fee:balance> element (as described in Section 3.5);
- An OPTIONAL <fee:creditLimit> element (as described in Section 3.6).

Example <renew> command:

```
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.com</domain:name>
C:         <domain:curExpDate>2000-04-03</domain:curExpDate>
C:         <domain:period unit="y">5</domain:period>
C:       </domain:renew>
C:     </renew>
C:     <extension>
C:       <fee:renew xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
C:         <fee:currency>USD</fee:currency>
C:         <fee:fee>5.00</fee:fee>
C:       </fee:renew>
C:     </extension>
C:     <clTRID>ABC-12345</clTRID>
C:   </command>
C: </epp>
```

Example <renew> response:

```
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:renData
S:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:         <domain:name>example.com</domain:name>
S:         <domain:exDate>2005-04-03T22:00:00.0Z</domain:exDate>
S:       </domain:renData>
S:     </resData>
S:     <extension>
S:       <fee:renData xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
S:         <fee:currency>USD</fee:currency>
S:         <fee:fee
S:           refundable="1"
S:           grace-period="P5D">5.00</fee:fee>
S:         <fee:balance>1000.00</fee:balance>
S:       </fee:renData>
S:     </extension>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54322-XYZ</svTRID>
S:     </trID>
S:   </response>
S: </epp>
```

5.2.4. EPP <transfer> Command

This extension adds elements to both the EPP <transfer> command and response, when the value of the "op" attribute of the <transfer> command element is "request", and the extension has been selected during the <login> command.

When submitting a <transfer> command to the server, the client MAY include in the <extension> element a <fee:transfer> element which includes the following child elements:

- An OPTIONAL <fee:currency> element (as described in Section 3.2);
- One or more <fee:fee> elements (as described in Section 3.4).

When the <transfer> command has been processed successfully, and the client selected the extension when it logged in, the server MAY

include in the <extension> section of the EPP response a <fee:trnData> element, which contains the following child elements:

- A <fee:currency> element (as described in Section 3.2);
- Zero or more <fee:fee> elements (as described in Section 3.4);
- Zero or more <fee:credit> elements (as described in Section 3.4);
- An OPTIONAL <fee:balance> element (as described in Section 3.5);
- An OPTIONAL <fee:creditLimit> element (as described in Section 3.6).

Example <transfer> command:

```
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.com</domain:name>
C:         <domain:period unit="y">1</domain:period>
C:         <domain:authInfo>
C:           <domain:pw roid="JD1234-REP">2fooBAR</domain:pw>
C:         </domain:authInfo>
C:       </domain:transfer>
C:     </transfer>
C:     <extension>
C:       <fee:transfer xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
C:         <fee:currency>USD</fee:currency>
C:         <fee:fee>5.00</fee:fee>
C:       </fee:transfer>
C:     </extension>
C:     <clTRID>ABC-12345</clTRID>
C:   </command>
C: </epp>
```

Example <transfer> response:

```
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:       <domain:trnData
S:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:         <domain:name>example.com</domain:name>
S:         <domain:trStatus>pending</domain:trStatus>
S:         <domain:reID>ClientX</domain:reID>
S:         <domain:reDate>2000-06-08T22:00:00.0Z</domain:reDate>
S:         <domain:acID>ClientY</domain:acID>
S:         <domain:acDate>2000-06-13T22:00:00.0Z</domain:acDate>
S:         <domain:exDate>2002-09-08T22:00:00.0Z</domain:exDate>
S:       </domain:trnData>
S:     </resData>
S:     <extension>
S:       <fee:trnData xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
S:         <fee:currency>USD</fee:currency>
S:         <fee:fee
S:           refundable="1"
S:           grace-period="P5D">5.00</fee:fee>
S:       </fee:trnData>
S:     </extension>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54322-XYZ</svTRID>
S:     </trID>
S:   </response>
S: </epp>
```

5.2.5. EPP <update> Command

This extension adds elements to both the EPP <update> command and response, when the extension has been selected during the <login> command.

When submitting a <update> command to the server, the client MAY include in the <extension> element a <fee:update> element which includes the following child elements:

- An OPTIONAL <fee:currency> element (as described in Section 3.2);
- One or more <fee:fee> elements (as described in Section 3.4).

When the <update> command has been processed successfully, and the client selected the extension when it logged in, the server MAY include in the <extension> section of the EPP response a <fee:updData> element, which contains the following child elements:

- A <fee:currency> element (as described in Section 3.2);
- Zero or more <fee:fee> elements (as described in Section 3.4);
- Zero or more <fee:credit> elements (as described in Section 3.4);
- An OPTIONAL <fee:balance> element (as described in Section 3.5);
- An OPTIONAL <fee:creditLimit> element (as described in Section 3.6).

Example <update> command:

```
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.com</domain:name>
C:         <domain:chg>
C:           <domain:registrant>sh8013</domain:registrant>
C:         </domain:chg>
C:       </domain:update>
C:     </update>
C:     <extension>
C:       <fee:update xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
C:         <fee:currency>USD</fee:currency>
C:         <fee:fee>5.00</fee:fee>
C:       </fee:update>
C:     </extension>
C:     <clTRID>ABC-12345</clTRID>
C:   </command>
C: </epp>
```

Example <update> response:

```
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:     <extension>
S:       <fee:updData xmlns:fee="urn:ietf:params:xml:ns:fee-0.21">
S:         <fee:currency>USD</fee:currency>
S:         <fee:fee>5.00</fee:fee>
S:       </fee:updData>
S:     </extension>
S:     <trID>
S:       <clTRID>ABC-12345</clTRID>
S:       <svTRID>54321-XYZ</svTRID>
S:     </trID>
S:   </response>
S: </epp>
```

6. Formal Syntax

One schema is presented here that is the EPP Fee Extension schema.

The formal syntax presented here is a complete schema representation of the object mapping suitable for automated validation of EPP XML instances. The BEGIN and END tags are not part of the schema; they are used to note the beginning and ending of the schema for URI registration purposes.

6.1. Fee Extension Schema

```
BEGIN
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:fee="urn:ietf:params:xml:ns:fee-0.21"
  xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
  xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
  targetNamespace="urn:ietf:params:xml:ns:fee-0.21"
  elementFormDefault="qualified">

  <import namespace="urn:ietf:params:xml:ns:eppcom-1.0" />
  <import namespace="urn:ietf:params:xml:ns:domain-1.0" />

  <annotation>
    <documentation>
      Extensible Provisioning Protocol v1.0 Fee Extension
```

```
      </documentation>
    </annotation>

    <!-- Child elements found in EPP commands and responses -->
    <element name="check" type="fee:checkType" />
    <element name="chkData" type="fee:chkDataType" />
    <element name="create" type="fee:transformCommandType" />
    <element name="creData" type="fee:transformResultType" />
    <element name="renew" type="fee:transformCommandType" />
    <element name="renData" type="fee:transformResultType" />
    <element name="transfer" type="fee:transformCommandType" />
    <element name="trnData" type="fee:transformResultType" />
    <element name="update" type="fee:transformCommandType" />
    <element name="updData" type="fee:transformResultType" />
    <element name="delData" type="fee:transformResultType" />

    <!-- client <check> command -->
    <complexType name="checkType">
      <sequence>
        <element name="currency" type="fee:currencyType"
          minOccurs="0" />
        <element name="command" type="fee:commandType"
          minOccurs="1" maxOccurs="unbounded" />
      </sequence>
    </complexType>

    <complexType name="objectIdentifierType">
      <simpleContent>
        <extension base="eppcom:labelType">
          <attribute name="element"
          type="NMTOKEN" default="name" />
        </extension>
      </simpleContent>
    </complexType>

    <!-- server <check> result -->
    <complexType name="chkDataType">
      <sequence>
        <element name="currency" type="fee:currencyType" />
        <element name="cd" type="fee:objectCDType"
          maxOccurs="unbounded" />
      </sequence>
    </complexType>

    <complexType name="objectCDType">
      <sequence>
        <element name="objID" type="fee:objectIdentifierType" />
        <element name="command" type="fee:commandType"
```

```
          maxOccurs="unbounded" />
        <element name="reason" type="fee:reasonType" minOccurs="0" />
      </sequence>
      <attribute name="avail" type="boolean" default="1" />
    </complexType>

    <!-- general transform (create, renew, update, transfer) command -->
    <complexType name="transformCommandType">
      <sequence>
        <element name="currency" type="fee:currencyType"
          minOccurs="0" />
        <element name="fee" type="fee:feeType"
          maxOccurs="unbounded" />
        <element name="credit" type="fee:creditType"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </complexType>

    <!-- general transform (create, renew, update) result -->
    <complexType name="transformResultType">
      <sequence>
        <element name="currency" type="fee:currencyType"
          minOccurs="0" />
        <element name="period" type="domain:periodType"
          minOccurs="0" />
        <element name="fee" type="fee:feeType"
          minOccurs="0" maxOccurs="unbounded" />
        <element name="credit" type="fee:creditType"
          minOccurs="0" maxOccurs="unbounded" />
        <element name="balance" type="fee:balanceType"
          minOccurs="0" />
        <element name="creditLimit" type="fee:creditLimitType"
          minOccurs="0" />
      </sequence>
    </complexType>

    <!-- common types -->
    <simpleType name="currencyType">
      <restriction base="string">
        <pattern value="[A-Z]{3}" />
      </restriction>
    </simpleType>

    <complexType name="commandType">
      <sequence>
        <element name="period" type="domain:periodType"
          minOccurs="0" maxOccurs="1" />
        <element name="fee" type="fee:feeType"
```

```
          minOccurs="0" maxOccurs="unbounded" />
        <element name="credit" type="fee:creditType"
          minOccurs="0" maxOccurs="unbounded" />
        <element name="class" type="token" minOccurs="0" />
        <element name="reason" type="fee:reasonType" minOccurs="0" />
      </sequence>
      <attribute name="name" type="fee:commandEnum"/>
      <attribute name="customName" type="token"/>
      <attribute name="phase" type="token" />
      <attribute name="subphase" type="token" />
    </complexType>

    <complexType name="reasonType">
      <simpleContent>
        <extension base="token">
          <attribute name="lang" type="language"/>
        </extension>
      </simpleContent>
    </complexType>

    <simpleType name="commandEnum">
      <restriction base="token">
        <enumeration value="create"/>
        <enumeration value="delete"/>
        <enumeration value="renew"/>
        <enumeration value="update"/>
        <enumeration value="transfer"/>
        <enumeration value="restore"/>
        <enumeration value="custom"/>
      </restriction>
    </simpleType>

    <simpleType name="nonNegativeDecimal">
      <restriction base="decimal">
        <minInclusive value="0" />
      </restriction>
    </simpleType>

    <simpleType name="negativeDecimal">
      <restriction base="decimal">
        <maxInclusive value="0" />
      </restriction>
    </simpleType>

    <complexType name="feeType">
      <simpleContent>
        <extension base="fee:nonNegativeDecimal">
          <attribute name="description"/>
```

```
          <attribute name="refundable" type="boolean" />
          <attribute name="grace-period" type="duration" />
          <attribute name="applied">
            <simpleType>
              <restriction base="token">
                <enumeration value="immediate" />
                <enumeration value="delayed" />
              </restriction>
            </simpleType>
          </attribute>
        </extension>
      </simpleContent>
    </complexType>

    <complexType name="creditType">
      <simpleContent>
        <extension base="fee:negativeDecimal">
          <attribute name="description"/>
        </extension>
      </simpleContent>
    </complexType>

    <simpleType name="balanceType">
      <restriction base="decimal" />
    </simpleType>

    <simpleType name="creditLimitType">
      <restriction base="decimal" />
    </simpleType>

  </schema>
  END
```

7. Security Considerations

The mapping extensions described in this document do not provide any security services beyond those described by EPP [RFC5730], the EPP domain name mapping [RFC5731], and protocol layers used by EPP. The security considerations described in these other specifications apply to this specification as well.

8. IANA Considerations

8.1. XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. The following URI assignment is requested of IANA:

URI: ietf:params:xml:ns:fee-0.21

Registrant Contact: See the "Author's Address" section of this document.

XML: See the "Formal Syntax" section of this document.

8.2. EPP Extension Registry

The EPP extension described in this document should be registered by the IANA in the EPP Extension Registry described in [RFC7451]. The details of the registration are as follows:

Name of Extension: EPP Fee Extension

Document status: Standards Track

Reference: (insert reference to RFC version of this document)

Registrant Name and Email Address: See the "Author's Address" section of this document.

TLDs: any

IPR Disclosure: none

Status: active

Notes: none

9. Implemntation Status

Note to RFC Editor: Please remove this section and the reference to [RFC6982] before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. RegistryEngine EPP Service

Organization: CentralNic

Name: RegistryEngine EPP Service

Description: Generic high-volume EPP service for gTLDs, ccTLDs and SLDs

Level of maturity: Deployed in CentralNic’s production environment as well as two other gTLD registry systems, and two ccTLD registry systems.

Coverage: All aspects of the protocol are implemented.

Licensing: Proprietary In-House software

Contact: epp@centralnic.com

URL: https://www.centralnic.com

10. Acknowledgements

The authors wish to thank the following persons for their feedback and suggestions:

- James Gould of Verisign Inc
- Luis Munoz of ISC
- Michael Young of Architelos
- Ben Levac and Jeff Eckhaus of Demand Media
- Seth Goldman of Google
- Klaus Malorny and Michael Bauland of Knipp
- Jody Kolker, Joe Snitker and Kevin Allendorf of Go Daddy
- Michael Holloway of Com Laude
- Santosh Kalsangrah of Impetus Infotech
- Alex Mayrhofer of Nic.at

11. Change History

11.1. Change from 04 to 05

Updated scheme to version 0.21 to support the lang attribute for the reason element of the objectCDType and the commandType types as well as to add the update command to the commandEnum type. Updated section 3.1 to include language for the custom command. Added section 3.9 to provide a description of the <fee:reason> element. Fixed typos and added clarification text on when client fee is less than server fee in section 4. Additionally, I added description pointers to appropriate Section 3 definitions for element clarity throughout the document.

11.2. Change from 03 to 04

Updated scheme to version 0.19 to correct typos and to replace the commandTypeValue type with the commandEnum type and customName attribute for stricter validation. Updated various text for grammar and clarity. Added text to section 4 clarifying the <check> response when the client provided no fee extension but the server was expecting the extension.

11.3. Change from 02 to 03

Updated scheme to version 0.17 to simplify the check command syntax. Moved fee avail to objectCDType to allow fast failing on error situations. Removed the objectCheckType as it was no longer being used. Updated examples to reflect these scheme changes. Added language for server failing a <create> if the <fee:fee> passed by the client is less than the server fee.

11.4. Change from 01 to 02

Updated scheme to version 0.15 to fix errors in CommandType, objectCDType, transformCommandType and transformResultType definitions.

11.5. Change from 00 to 01

Added Roger Carney as author to finish draft. Moved Formal Syntax section to main level numbering. Various grammar, typos, and administrative edits for clarity. Removed default value for the "applied" attribute of <fee:fee> so that it can truly be optional. Added support for the <delete> command to return a <fee:fee> element as well. Modified default response on the <check> command for the optional <fee:period> when it was not provided in the command, leaving it to the server to provide the default period value.

Extensive edits were done to the <check> command, the <check> response and to the fee extension schema (checkType, objectCheckType, objectIdentifierType, objectCDType, commandType) to support requesting and returning multiple transformation fees in a single call. Added section on Phase/Subphase to provide more context on the uses.

11.6. Change from draft-brown-00 to draft-ietf-regext-fees-00

Updated to be REGEXT WG document.

12. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC3915] Hollenbeck, S., "Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP)", RFC 3915, DOI 10.17487/RFC3915, September 2004, <http://www.rfc-editor.org/info/rfc3915>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <http://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, <http://www.rfc-editor.org/info/rfc5731>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <http://www.rfc-editor.org/info/rfc6982>.

[RFC7451] Hollenbeck, S., "Extension Registry for the Extensible Provisioning Protocol", RFC 7451, DOI 10.17487/RFC7451, February 2015, <http://www.rfc-editor.org/info/rfc7451>.

Authors' Addresses

Roger Carney
GoDaddy Inc.
14455 N. Hayden Rd. #219
Scottsdale, AZ 85260
US

Email: rcarney@godaddy.com
URI: http://www.godaddy.com

Gavin Brown
CentralNic Group plc
35-39 Moorgate
London, England EC2R 6AR
GB

Phone: +44 20 33 88 0600
Email: gavin.brown@centralnic.com
URI: http://www.centralnic.com

Jothan Frakes

Email: jothan@jothan.com
URI: http://jothan.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 1, 2017

J. Gould
VeriSign, Inc.

October 28, 2016

Extensible Provisioning Protocol (EPP) and Registration Data Access Protocol (RDAP) Status Mapping
draft-ietf-regext-epp-rdap-status-mapping-04

Abstract

This document describes the mapping of the Extensible Provisioning Protocol (EPP) statuses with the statuses registered for use in the Registration Data Access Protocol (RDAP). This document identifies gaps in the mapping, and registers RDAP statuses to fill the gaps to ensure that all of the EPP RFC statuses are supported in RDAP.

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 May 1, 2017.

Table of Contents

1. Introduction

This document maps the statuses defined in the Extensible Provisioning Protocol (EPP) RFCs to the list of statuses registered for use in the Registration Data Access Protocol (RDAP), in the RDAP JSON Values Registry [rdap-json-values].

The RDAP JSON Values Registry is described in section 10.2 of [RFC7483] and is available in the RDAP JSON Values Registry [rdap-json-values].

The EPP statuses used as the source of the mapping include section 2.3 of the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731], section 2.3 of the Extensible Provisioning Protocol (EPP) Host Mapping [RFC5732], section 2.2 of the Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733], and section 3.1 of Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915].

Each EPP status MUST map to a single RDAP status to ensure that data in the Domain Name Registries (DNRs) that use EPP can be accurately presented in RDAP.

1.1. 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 RFC 2119 [RFC2119].

2. EPP to RDAP Status Mapping

Below is an alphabetically sorted list of EPP statuses from the EPP RFCs ([RFC5731], [RFC5732], [RFC5733], and [RFC3915]) mapped to the RDAP statuses registered in the RDAP JSON Values Registry [rdap-json-values], with the format <EPP Status> '=' <RDAP Status>, where a blank <RDAP Status> indicates a gap in the mapping.

addPeriod =
autoRenewPeriod =
clientDeleteProhibited =
clientHold =
clientRenewProhibited =
clientTransferProhibited =
clientUpdateProhibited =
inactive = inactive
linked = associated
ok = active
pendingCreate = pending create
pendingDelete = pending delete
pendingRenew = pending renew
pendingRestore =
pendingTransfer = pending transfer
pendingUpdate = pending update
redemptionPeriod =
renewPeriod =
serverDeleteProhibited =
serverRenewProhibited =
serverTransferProhibited =
serverUpdateProhibited =
serverHold =
transferPeriod =

The RDAP JSON Values Registry [rdap-json-values] does have a set of prohibited statuses including "renew prohibited", "update prohibited", "transfer prohibited", and "delete prohibited", but these statuses do not directly map to the EPP prohibited statuses. EPP provides status codes that allow distinguishing the case that an action is prohibited because of server policy from the case that an action is prohibited because of a client request. The ability to make this distinction needs to be preserved in RDAP.

Each of the EPP status values that don't map directly to an RDAP status value is described below. Each EPP status value includes a proposed new RDAP status value and a description of the value. The RDAP status value is derived from the EPP status value by converting the EPP camel case representation to lower case with a space character inserted between word boundaries.

addPeriod = add period; This grace period is provided after the initial registration of the object. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the registration.

autoRenewPeriod = auto renew period; This grace period is provided after an object registration period expires and is extended (renewed) automatically by the server. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the auto renewal.

clientDeleteProhibited = client delete prohibited; The client requested that requests to delete the object MUST be rejected.

clientHold = client hold; The client requested that the DNS delegation information MUST NOT be published for the object.

clientRenewProhibited = client renew prohibited; The client requested that requests to renew the object MUST be rejected.

clientTransferProhibited = client transfer prohibited; The client requested that requests to transfer the object MUST be rejected.

clientUpdateProhibited = client update prohibited; The client requested that requests to update the object (other than to remove this status) MUST be rejected.

pendingRestore = pending restore; An object is in the process of being restored after being in the redemption period state.

redemptionPeriod = redemption period; A delete has been received, but the object has not yet been purged because an opportunity exists to restore the object and abort the deletion process.

renewPeriod = renew period; This grace period is provided after an object registration period is explicitly extended (renewed) by the client. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the renewal.

serverDeleteProhibited = server delete prohibited; The server set the status so that requests to delete the object MUST be rejected.

serverRenewProhibited = server renew prohibited; The server set the status so that requests to renew the object MUST be rejected.

serverTransferProhibited = server transfer prohibited; The server set the status so that requests to transfer the object MUST be rejected.

serverUpdateProhibited = server update prohibited; The server set the status so that requests to update the object (other than to remove this status) MUST be rejected.

serverHold = server hold; The server set the status so that DNS delegation information MUST NOT be published for the object.
transferPeriod = transfer period; This grace period is provided after the successful transfer of object registration sponsorship from one client to another client. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the transfer.

The resulting mapping after registering the new RDAP statuses is:

addPeriod = add period
autoRenewPeriod = auto renew period
clientDeleteProhibited = client delete prohibited
clientHold = client hold
clientRenewProhibited = client renew prohibited
clientTransferProhibited = client transfer prohibited
clientUpdateProhibited = client update prohibited
inactive = inactive
linked = associated
ok = active
pendingCreate = pending create
pendingDelete = pending delete
pendingRenew = pending renew
pendingRestore = pending restore
pendingTransfer = pending transfer
pendingUpdate = pending update
redemptionPeriod = redemption period
renewPeriod = renew period
serverDeleteProhibited = server delete prohibited
serverRenewProhibited = server renew prohibited
serverTransferProhibited = server transfer prohibited
serverUpdateProhibited = server update prohibited
serverHold = server hold
transferPeriod = transfer period

3. IANA Considerations

3.1. JSON Values Registry

The following values should be registered by the IANA in the RDAP JSON Values Registry described in [RFC7483]:

Value: add period

Type: status

Description: This grace period is provided after the initial registration of the object. If the object is deleted by the client

during this period, the server provides a credit to the client for the cost of the registration. This maps to the Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915] 'addPeriod' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: auto renew period

Type: status

Description: This grace period is provided after an object registration period expires and is extended (renewed) automatically by the server. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the auto renewal. This maps to the Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915] 'autoRenewPeriod' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: client delete prohibited

Type: status

Description: The client requested that requests to delete the object MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731], Extensible Provisioning Protocol (EPP) Host Mapping [RFC5732], and Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733] 'clientDeleteProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: client hold

Type: status

Description: The client requested that the DNS delegation information MUST NOT be published for the object. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731] 'clientHold' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: client renew prohibited

Type: status

Description: The client requested that requests to renew the object MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731] 'clientRenewProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: client transfer prohibited

Type: status

Description: The client requested that requests to transfer the object MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731] and Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733] 'clientTransferProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: client update prohibited

Type: status

Description: The client requested that requests to update the object (other than to remove this status) MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731], Extensible Provisioning Protocol (EPP) Host Mapping [RFC5732], and Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733] 'clientUpdateProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: pending restore

Type: status

Description: An object is in the process of being restored after being in the redemption period state. This maps to the Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915] 'pendingRestore' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: redemption period

Type: status

Description: A delete has been received, but the object has not yet been purged because an opportunity exists to restore the object and abort the deletion process. This maps to the Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915] 'redemptionPeriod' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: renew period

Type: status

Description: This grace period is provided after an object registration period is explicitly extended (renewed) by the client. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the renewal. This maps to the Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915] 'renewPeriod' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: server delete prohibited

Type: status

Description: The server set the status so that requests to delete the object MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731], Extensible Provisioning Protocol (EPP) Host Mapping [RFC5732], and Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733] 'serverDeleteProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: server renew prohibited

Type: status

Description: The server set the status so that requests to renew the object MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731] 'serverRenewProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: server transfer prohibited

Type: status

Description: The server set the status so that requests to transfer the object MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731] and Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733] 'serverTransferProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: server update prohibited

Type: status

Description: The server set the status so that requests to update the object (other than to remove this status) MUST be rejected. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731], Extensible Provisioning Protocol (EPP) Host Mapping [RFC5732], and Extensible Provisioning Protocol (EPP) Contact Mapping [RFC5733] 'serverUpdateProhibited' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: server hold

Type: status

Description: The server set the status so that DNS delegation information MUST NOT be published for the object. This maps to the Extensible Provisioning Protocol (EPP) Domain Name Mapping [RFC5731] 'serverHold' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

Value: transfer period

Type: status

Description: This grace period is provided after the successful transfer of object registration sponsorship from one client to another client. If the object is deleted by the client during this period, the server provides a credit to the client for the cost of the transfer. This maps to the Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP) [RFC3915] 'transferPeriod' status.

Registrant Name: IESG

Registrant Contact Information: iesg@ietf.org

4. Security Considerations

The status values described in this document can be subject to server-side information disclosure policies that restrict display of the values to authorized clients. Implementers may wish to review [RFC7481] for a description of the RDAP security services that can be used to implement information disclosure policies.

5. Normative References

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

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

[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.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", RFC 7481, DOI 10.17487/RFC7481, March 2015, <http://www.rfc-editor.org/info/rfc7481>.

[RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, March 2015.

[rdap-json-values] "RDAP JSON Values Registry", <https://www.iana.org/assignments/rdap-json-values/rdap-json-values.xhtml>.

Appendix A. Acknowledgements

Suggestions that have been incorporated into this document were provided by Andrew Newton, Scott Hollenbeck, Jim Galvin, Gustavo Lozano, and Robert Sparks.

Appendix B. Change History

B.1. Change from 00 to 01

1. Changed the mapping of "linked" to "associated" and removed the registration of "linked", based on feedback from Andrew Newton on the weirds mailing list.

B.2. Change from 01 to 02

1. Ping update.

B.3. Change from 02 to 03

1. Ping update.

B.4. Change from 03 to REGEXT 00

1. Changed to regext working group draft by changing draft-gould-epp-rdap-status-mapping to draft-ietf-regext-epp-rdap-status-mapping.

B.5. Change from REGEXT 00 to REGEXT 01

1. Updated based on regext mailing feedback from Scott Hollenbeck that included updating the registrant for the registration of the new statuses to IESG and iesg@ietf.org, and revising the security section. Changed to standards track based on suggestion by Jim Galvin and support from Gustavo Lozano on the regext mailing list.

B.6. Change from REGEXT 01 to REGEXT 02

1. Updated the text associated with distinguishing client and server prohibited statuses in RDAP based on feedback by Robert Sparks on the regext mailing list.
2. Removed the "For DNR that indicates" text from the description of the statuses based on feedback by Robert Sparks on the regext mailing list.
3. Made a few editorial changes to the status descriptions including referring to "redemption period" instead of "redemptionPeriod" and referring to "object" instead of "domain name".
4. Changed all references of "registrar" to "client" and "registry" to "server" in the status descriptions to be consistent.

B.7. Change from REGEXT 02 to REGEXT 03

1. Updated descriptions of the add period, auto renew period, renew period, and transfer period statuses to better reflect what the status is in RFC 3915, based on feedback by Robert Sparks on the regext mailing list.

B.8. Change from REGEXT 03 to REGEXT 04

1. Updated the descriptions of the JSON Values Registry entries to include a reference back to the appropriate EPP RFC status, based on feedback by Sabrina Tanamal from IANA.

Author's Address

James Gould
VeriSign, Inc.
12061 Bluemont Way
Reston, VA 20190
US

Email: jgould@verisign.com
URI: http://www.verisigninc.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 24, 2017

J. Gould
VeriSign, Inc.
W. Tan
Cloud Registry
G. Brown
CentralNic Ltd

June 22, 2017

Launch Phase Mapping for the Extensible Provisioning Protocol (EPP)
draft-ietf-regext-launchphase-05

Abstract

This document describes an Extensible Provisioning Protocol (EPP) extension mapping for the provisioning and management of domain name registrations and applications during the launch of a domain name registry.

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 December 24, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document describes an extension mapping for version 1.0 of the Extensible Provisioning Protocol (EPP) [RFC5730]. This EPP mapping specifies a flexible schema that can be used to implement several common use cases related to the provisioning and management of domain name registrations and applications during the launch of a domain name registry.

It is typical for domain registries to operate in special modes during their initial launch to facilitate allocation of domain names, often according to special rules. This document uses the term

"launch phase" and the shorter form "launch" to refer to such a period.

The EPP domain name mapping [RFC5731] is designed for the steady-state operation of a registry. During a launch period, the model in place may be different from what is defined in the EPP domain name mapping [RFC5731]. For example, registries often accept multiple applications for the same domain name during the "Sunrise" launch phase, referred to as a Launch Application. A Launch Registration refers to a registration made during a launch phase when the server uses a "first-come, first-served" model. Even in a "first-come, first-served" model, additional steps and information might be required, such as trademark information. In addition, RFC 7848 [RFC7848] defines a registry interface for the Trademark Claims or "claims" launch phase that includes support for presenting a Trademark Claims Notice to the Registrant. This document proposes an extension to the domain name mapping in order to provide a uniform interface for the management of Launch Applications and Launch Registrations in launch phases.

1.1. 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 RFC 2119 [RFC2119].

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

In examples, "C:" represents lines sent by a protocol client and "S:" represents lines returned by a protocol server. Indentation and white space in examples are provided only to illustrate element relationships and are not a REQUIRED feature of this protocol.

"launch-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:launch-1.0". The XML namespace prefix "launch" is used, but implementations MUST NOT depend on it and instead employ a proper namespace-aware XML parser and serializer to interpret and output the XML documents.

"signedMark-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:signedMark-1.0" that is defined in [RFC7848]. The XML namespace prefix "smd" is used, but implementations MUST NOT depend on it and instead employ a proper namespace-aware XML parser and serializer to interpret and output the XML documents.

"mark-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:mark-1.0" that is defined in [RFC7848]. The XML namespace prefix "mark" is used, but implementations MUST NOT depend on it and instead employ a proper namespace-aware XML parser and serializer to interpret and output the XML documents.

2. Object Attributes

This extension adds additional elements to the EPP domain name mapping [RFC5731]. Only those new elements are described here.

2.1. Application Identifier

Servers MAY allow multiple applications, referred to as a Launch Application, of the same domain name during its launch phase operations. Upon receiving a valid <domain:create> command to create a Launch Application, the server MUST create an application object corresponding to the request, assign an application identifier for the Launch Application, set the [RFC5731] pendingCreate status, and return the application identifier to the client with the <launch:applicationID> element. In order to facilitate correlation, all subsequent launch operations on the Launch Application MUST be qualified by the previously assigned application identifier using the <launch:applicationID> element.

2.2. Validator Identifier

The Validator Identifier is the unique identifier for a Trademark Validator that validates marks and has a repository of validated marks. The OPTIONAL "validatorID" attribute is used to define the Validator Identifier of the Trademark Validator. Registries MAY support more than one Third Party Trademark Validator. The Internet Corporation for Assigned Names and Numbers (ICANN) Trademark Clearinghouse (TMCH) is the default Trademark Validator and is reserved the Validator Identifier of "tmch". If the ICANN TMCH is not used or multiple Trademark Validators are used, the Validator Identifier MUST be defined using the "validatorID" attribute.

The Validator Identifier MAY be related to one or more issuer identifiers of the <mark:id> element and the <smd:id> element defined in [RFC7848]. Both the Validator Identifier and the Issuer Identifier used MUST be unique. If the ICANN TMCH is not used or multiple Trademark Validators are used, the server MUST define the list of supported validator identifiers and MUST make this information available to clients using a mutually acceptable, out-of-band mechanism.

The Validator Identifier MAY define a non-Trademark Validator that supports a form of claims.

2.3. Launch Phases

The server MAY support multiple launch phases sequentially or simultaneously. The <launch:phase> element MUST be included by the client to define the target launch phase of the command. The server SHOULD validate the phase and MAY validate the sub-phase of the <launch:phase> element against the active phase and OPTIONAL sub-phase of the server on a create command, and return an EPP error result code of 2306 if there is a mismatch.

The following launch phase values are defined:

sunrise The phase during which trademark holders can submit registrations or applications with trademark information that can be validated by the server.

landrush A post-Sunrise phase when non-trademark holders are allowed to register domain names with steps taken to address a large volume of initial registrations.

claims The phase, as defined in the Section 2.3.1, in which a Claims Notice MUST be displayed to a prospective registrant of a domain name that matches trademarks.

open A post-launch phase that is also referred to as "steady state". Servers MAY require additional trademark protection during this phase.

custom A custom server launch phase that is defined using the "name" attribute.

For extensibility, the <launch:phase> element includes an OPTIONAL "name" attribute that can define a sub-phase, or the full name of the phase when the <launch:phase> element has the "custom" value. For example, the "claims" launch phase could have two sub-phases that include "landrush" and "open".

Launch phases MAY overlap to support the "claims" launch phase, defined in the Section 2.3.1, and to support a traditional "landrush" launch phase. The overlap of the "claims" and "landrush" launch phases SHOULD be handled by setting "claims" as the <launch:phase> value and setting "landrush" as the sub-phase with the "name" attribute. For example, the <launch:phase> element SHOULD be <launch:phase name="landrush">claims</launch:phase>.

2.3.1. Trademark Claims Phase

The Trademark Claims Phase is when a Claims Notice MUST be displayed to a prospective registrant of a domain name that matches trademarks. The source of the trademarks is a Trademark Validator and the source of the Claims Notice information is a Claim Notice Information Service (CNIS), which MAY be directly linked to a Trademark Validator. The client interfaces with the server to determine if a trademark exists for a domain name, interfaces with a CNIS to get the Claims Notice information, and interfaces with the server to pass the Claims Notice acceptance information in a create command. This document supports the Trademark Claims Phase in two ways including:

Claims Check Form Claims Check Form (Section 3.1.1) is used to determine whether or not there are any matching trademarks for a domain name. If there is at least one matching trademark that exists for the domain name, a claims key is returned. The mapping of domain names and the claims keys is based on an out-of-band interface between the server and the Trademark Validator. The CNIS associated with the claims key Validator Identifier (Section 2.2) MUST accept the claims key as the basis for retrieving the claims information.

Claims Create Form Claims Create Form (Section 3.3.2) is used to pass the Claims Notice acceptance information in a create command. The notice identifier (<launch:noticeID>) format, validation rules, and server processing is up to the interface between the server and the Trademark Validator. The CNIS associated with the Validator Identifier (Section 2.2) MUST generate a notice identifier compliant with the <launch:noticeID> element.

The following shows the Trademark Claims Phase registration flow:

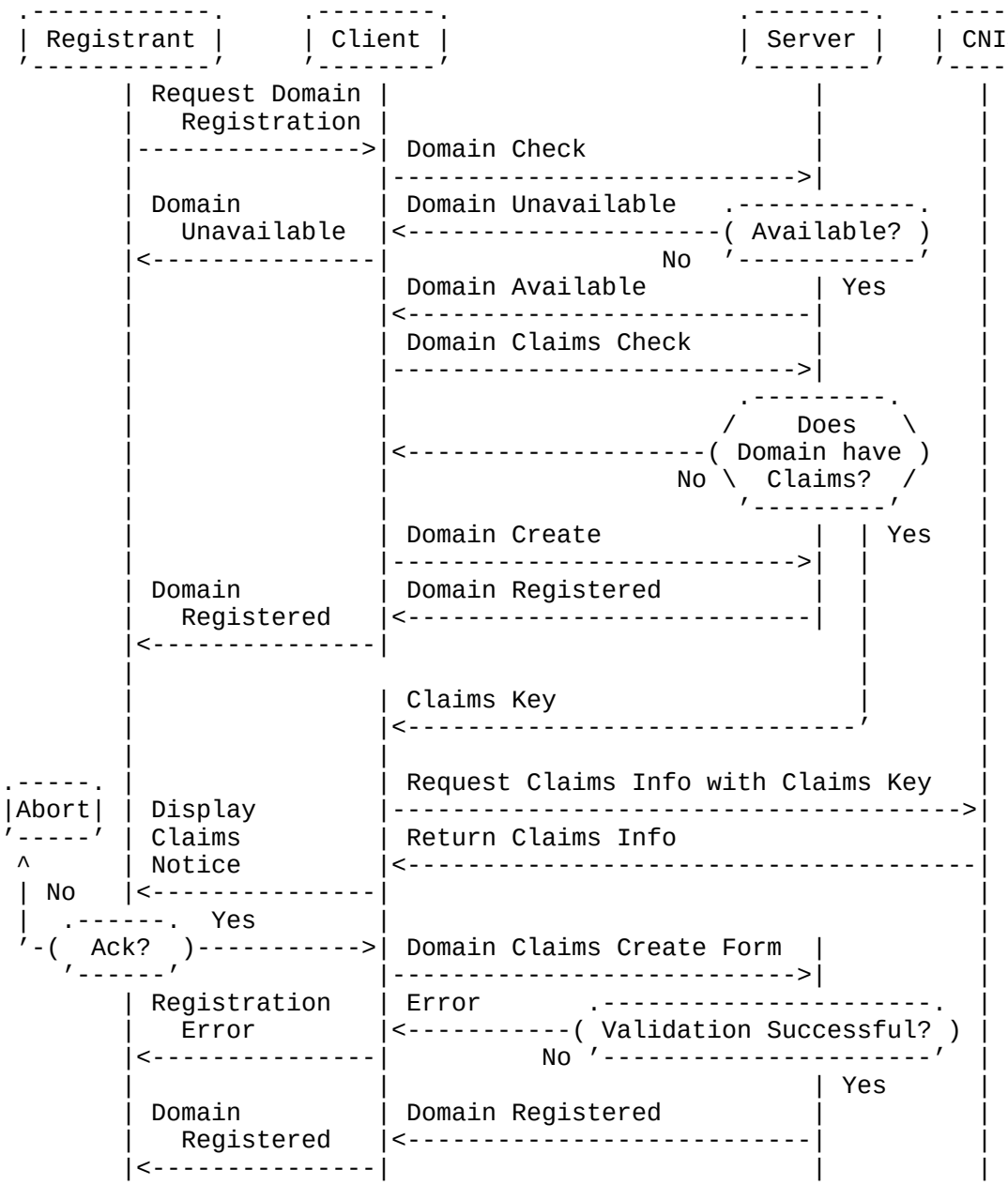

Figure 1

2.4. Status Values

A Launch Application or Launch Registration object MAY have a launch status value. The <launch:status> element is used to convey the launch status pertaining to the object, beyond what is specified in the object mapping. A Launch Application or Launch Registration MUST set the [RFC5731] "pendingCreate" status if a launch status is supported and the launch status is not one of the final statuses, including the "allocated" and "rejected" statuses.

The following status values are defined using the required "s" attribute:

pendingValidation: The initial state of a newly-created application or registration object. The application or registration requires validation, but the validation process has not yet completed.
validated: The application or registration meets relevant registry rules.
invalid: The application or registration does not validate according to registry rules. Server policies permitting, it may transition back into "pendingValidation" for revalidation, after modifications are made to ostensibly correct attributes that caused the validation failure.
pendingAllocation: The allocation of the application or registration is pending based on the results of some out-of-band process (for example, an auction).
allocated: The object corresponding to the application or registration has been provisioned. Is a possible end state of an application or registration object.
rejected: The application or registration object was not provisioned. Is a possible end state of an application or registration object.
custom: A custom status that is defined using the "name" attribute.

Each status value MAY be accompanied by a string of human-readable text that describes the rationale for the status applied to the object. The OPTIONAL "lang" attribute MAY be present to identify the language if the negotiated value is something other than the default value of "en" (English).

For extensibility the <launch:status> element includes an OPTIONAL "name" attribute that can define a sub-status or the full name of the status when the status value is "custom". The server SHOULD NOT use the "custom" status value.

Status values MAY be skipped. For example, an application or registration MAY immediately start at the "allocated" status or an application or registration MAY skip the "pendingAllocation" status.

If the launch phase does not require validation of a request, an application or registration MAY immediately skip to "pendingAllocation".

2.4.1. State Transition

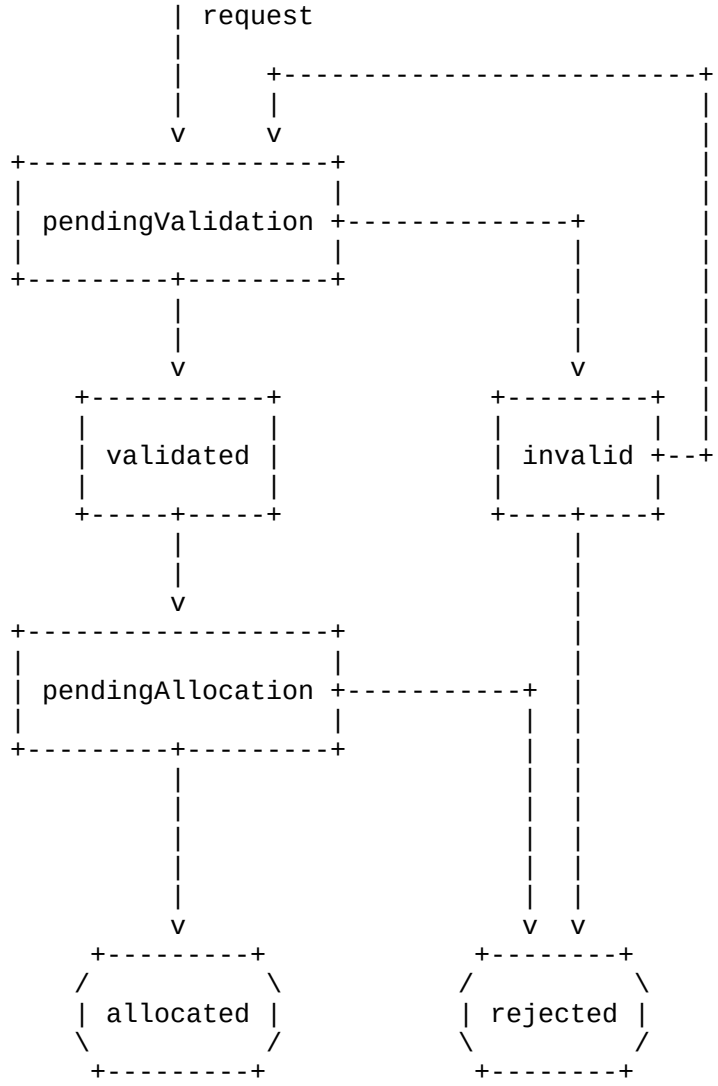

Figure 2

2.5. Poll Messaging

A Launch Application MUST and a Launch Registration MAY be handled as an EPP domain name object as specified in RFC 5731 [RFC5731] in "pendingCreate" status, with the launch status values defined in Section 2.4. As a Launch Application or Launch Registration transitions between the status values defined in Section 2.4, the server SHOULD insert poll messages, per [RFC5730], for the applicable intermediate statuses, including the "pendingValidation", "validated", "pendingAllocation, and "invalid" statuses, using the <domain:infData> element with the <launch:infData> extension. The <domain:infData> element MAY contain non-mandatory information, like contact and name server information. Also, further extensions that would normally be included in the response of a <domain:info> command, per [RFC5731], MAY be included. For the final statuses, including the "allocated" and "rejected" statuses, the server MUST insert a <domain:panData> poll message, per [RFC5731], with the <launch:infData> extension.

The following is an example poll message for a Launch Application that has transitioned to the "pendingAllocation" state.

```
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>2013-04-04T22:01:00.0Z</qDate>
S:      <msg>Application pendingAllocation.</msg>
S:    </msgQ>
S:    <resData>
S:      <domain:infData
S:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        <domain:name>domain.example</domain:name>
S:        ...
S:      </domain:infData>
S:    </resData>
S:    <extension>
S:      <launch:infData
S:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:        <launch:phase>sunrise</launch:phase>
S:          <launch:applicationID>abc123</launch:applicationID>
S:          <launch:status s="pendingAllocation"/>
S:      </launch:infData>
S:    </extension>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The following is an example <domain:panData> poll message for an "allocated" Launch Application.

```
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>2013-04-04T22:01:00.0Z</qDate>
S:      <msg>Application successfully allocated.</msg>
S:    </msgQ>
S:    <resData>
S:      <domain:panData
S:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        <domain:name paResult="1">domain.example</domain:name>
S:        <domain:paTRID>
S:          <clTRID>ABC-12345</clTRID>
S:          <svTRID>54321-XYZ</svTRID>
S:        </domain:paTRID>
S:        <domain:paDate>2013-04-04T22:00:00.0Z</domain:paDate>
S:      </domain:panData>
S:    </resData>
S:    <extension>
S:      <launch:infData
S:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:        <launch:phase>sunrise</launch:phase>
S:          <launch:applicationID>abc123</launch:applicationID>
S:          <launch:status s="allocated"/>
S:      </launch:infData>
S:    </extension>
S:    <trID>
S:      <clTRID>BCD-23456</clTRID>
S:      <svTRID>65432-WXY</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The following is an example <domain:panData> poll message for an "allocated" Launch Registration.

```
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>2013-04-04T22:01:00.0Z</qDate>
S:      <msg>Registration successfully allocated.</msg>
S:    </msgQ>
S:    <resData>
S:      <domain:panData
S:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        <domain:name paResult="1">domain.example</domain:name>
S:        <domain:paTRID>
S:          <clTRID>ABC-12345</clTRID>
S:          <svTRID>54321-XYZ</svTRID>
S:        </domain:paTRID>
S:        <domain:paDate>2013-04-04T22:00:00.0Z</domain:paDate>
S:      </domain:panData>
S:    </resData>
S:    <extension>
S:      <launch:infData
S:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:        <launch:phase>sunrise</launch:phase>
S:          <launch:status s="allocated"/>
S:      </launch:infData>
S:    </extension>
S:    <trID>
S:      <clTRID>BCD-23456</clTRID>
S:      <svTRID>65432-WXY</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

2.6. Mark Validation Models

A server MUST support at least one of the following models for validating trademark information:

code Use of a mark code by itself to validate that the mark matches the domain name. This model is supported using the <launch:codeMark> element with just the <launch:code> element.

mark The mark information is passed without any other validation element. The server will use some custom form of validation to

validate that the mark information is authentic. This model is supported using the <launch:codeMark> element with just the <mark:mark> (Section 2.6.2) element.

code with mark: A code is used along with the mark information by the server to validate the mark utilizing an external party. The code represents some form of secret that matches the mark information passed. This model is supported using the <launch:codeMark> element that contains both the <launch:code> and the <mark:mark> (Section 2.6.2) elements.

signed mark: The mark information is digitally signed as described in the Digital Signature (Section 2.6.3) section. The digital signature can be directly validated by the server using the public key of the external party that created the signed mark using its private key. This model is supported using the <smd:signedMark> (Section 2.6.3.1) and <smd:encodedSignedMark> (Section 2.6.3.2) elements.

More than one <launch:codeMark>, <smd:signedMark> (Section 2.6.3.1), or <smd:encodedSignedMark> (Section 2.6.3.2) element MAY be specified. The maximum number of marks per domain name is up to server policy.

2.6.1. <launch:codeMark> element

The <launch:codeMark> element that is used by the "code", "mark", and "code with mark" validation models, has the following child elements:

<launch:code>: OPTIONAL mark code used to validate the <mark:mark> (Section 2.6.2) information. The mark code is be a mark-specific secret that the server can verify against a third party. The OPTIONAL "validatorID" attribute is the Validator Identifier (Section 2.2) whose value indicates which Trademark Validator that the code originated from, with no default value.

<mark:mark>: OPTIONAL mark information with child elements defined in the Mark (Section 2.6.2) section.

The following is an example <launch:codeMark> element with both a <launch:code> and <mark:mark> (Section 2.6.2) element.

```
<launch:codeMark>
  <launch:code validatorID="sample">
    49FD46E6C4B45C55D4AC</launch:code>
  <mark:mark xmlns:mark="urn:ietf:params:xml:ns:mark-1.0">
    ...
  </mark:mark>
</launch:codeMark>
```

2.6.2. <mark:mark> element

A <mark:mark> element describes an applicant's prior right to a given domain name that is used with the "mark", "mark with code", and the "signed mark" validation models. The <mark:mark> element is defined in [RFC7848]. A new mark format can be supported by creating a new XML schema for the mark that has an element that substitutes for the <mark:abstractMark> element from [RFC7848].

2.6.3. Digital Signature

Digital signatures MAY be used by the server to validate either the mark information, when using the "signed mark" validation model with the <smd:signedMark> (Section 2.6.3.1) element or the <smd:encodedSignedMark> (Section 2.6.3.2) element.

2.6.3.1. <smd:signedMark> element

The <smd:signedMark> element contains the digitally signed mark information. The <smd:signedMark> element is defined in [RFC7848]. A new signed mark format can be supported by creating a new XML schema for the signed mark that has an element that substitutes for the <smd:abstractSignedMark> element from [RFC7848].

2.6.3.2. <smd:encodedSignedMark> element

The <smd:encodedSignedMark> element contains an encoded form of the digitally signed <smd:signedMark> (Section 2.6.3.1) element. The <smd:encodedSignedMark> element is defined in [RFC7848]. A new encoded signed mark format can be supported by creating a new XML schema for the encoded signed mark that has an element that substitutes for the <smd:encodedSignedMark> element from [RFC7848].

3. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in the EPP core protocol specification [RFC5730]. The command mappings described here are specifically for use in the Launch Phase Extension.

This mapping is designed to be flexible, requiring only a minimum set of required elements.

While it is meant to serve several use cases, it does not prescribe any interpretation by the client or server. Such processing is typically highly policy-dependent and therefore specific to implementations.

Operations on application objects are done via one or more of the existing EPP verbs defined in the EPP domain name mapping [RFC5731]. Registries MAY choose to support a subset of the operations.

3.1. EPP <check> Command

There are three forms of the extension to the EPP <check> command: the Claims Check Form (Section 3.1.1), the Availability Check Form (Section 3.1.2), and the Trademark Check Form (Section 3.1.3). The <launch:check> element "type" attribute defines the form, with the value of "claims" for the Claims Check Form (Section 3.1.1), with the value of "avail" for the Availability Check Form (Section 3.1.2), and with the value of "trademark" for the Trademark Check Form (Section 3.1.3). The default value of the "type" attribute is "claims". The forms supported by the server is determined by server policy. The server MUST return an EPP error result code of 2307 if it receives a check form that is not supported.

3.1.1. Claims Check Form

The Claims Check Form defines a new command called the Claims Check Command that is used to determine whether or not there are any matching trademarks, in the specified launch phase, for each domain name passed in the command, that requires the use of the "Claims Create Form" on a Domain Create Command. The availability check information defined in the EPP domain name mapping [RFC5731] MUST NOT be returned for the Claims Check Command. This form is the default form and MAY be explicitly identified by setting the <launch:check> "type" attribute to "claims".

Instead of returning whether the domain name is available, the Claims Check Command will return whether or not at least one matching trademark exists for the domain name, that requires the use of the "Claims Create Form" on a Domain Create Command. If there is at least one matching trademark that exists for the domain name, a <launch:claimKey> element is returned. The client MAY then use the value of the <launch:claimKey> element to obtain information needed to generate the Trademark Claims Notice from Trademark Validator based on the Validator Identifier (Section 2.2). The unique notice identifier of the Trademark Claims Notice MUST be passed in the <launch:noticeID> element of the extension to the Create Command (Section 3.3).

The <domain:name> elements in the EPP <check> command of EPP domain name mapping [RFC5731] define the domain names to check for matching trademarks. The <launch:check> element contains the following child elements:

<launch:phase> Contains the value of the active launch phase of the server. The server SHOULD validate the value against the active server launch phase.

Example Claims Check command using the <check> domain command and the <launch:check> extension with the "type" explicitly set to "claims", to determine if "domain1.example", "domain2.example", and "domain3.example" require claims notices during the "claims" launch phase:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:   <check>
C:    <domain:check
C:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:      <domain:name>domain1.example</domain:name>
C:      <domain:name>domain2.example</domain:name>
C:      <domain:name>domain3.example</domain:name>
C:    </domain:check>
C:   </check>
C:   <extension>
C:    <launch:check
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:     type="claims">
C:      <launch:phase>claims</launch:phase>
C:    </launch:check>
C:   </extension>
C:   <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

If the <check> command has been processed successfully, the EPP <response> MUST contain an <extension> <launch:chkData> element that identifies the launch namespace. The <launch:chkData> element contains the following child elements:

<launch:phase> The phase that mirrors the <launch:phase> element included in the <launch:check>.

<launch:cd> One or more <launch:cd> elements that contain the following child elements:

<launch:name> Contains the fully qualified name of the queried domain name. This element MUST contain an "exists" attribute whose value indicates if a matching trademark exists for the domain name that requires the use of the "Claims Create Form" on a Domain Create Command. A value of "1" (or "true") means

that a matching trademark does exist and that the "Claims Create Form" is required on a Domain Create Command. A value of "0" (or "false") means that a matching trademark does not exist or that the "Claims Create Form" is NOT required on a Domain Create Command.

<launch:claimKey> Zero or more OPTIONAL claim keys that MAY be passed to a third-party Trademark Validator such as the ICANN Trademark Clearinghouse (TMCH) for querying the information needed to generate a Trademark Claims Notice. The <launch:claimKey> is used as the key for the query in place of the domain name to securely query the service without using a well-known value like a domain name. The OPTIONAL "validatorID" attribute is the Validator Identifier (Section 2.2) whose value indicates which Trademark Validator to query for the Claims Notice information, with the default being the ICANN TMCH. The "validatorID" attribute MAY reference a non-trademark claims clearinghouse identifer to support other forms of claims notices.

Example Claims Check response when a claims notice is not required for the domain name domain1.example, a claims notice is required for the domain name domain2.example in the "tmch", and a claims notice is required for the domain name domain3.example in the "tmch" and "custom-tmch", for the "claims" launch phase:

```
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:    <extension>
S:     <launch:chkData
S:      xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:      <launch:phase>claims</launch:phase>
S:      <launch:cd>
S:        <launch:name exists="0">domain1.example</launch:name>
S:      </launch:cd>
S:      <launch:cd>
S:        <launch:name exists="1">domain2.example</launch:name>
S:        <launch:claimKey validatorID="tmch">
S:        2013041500/2/6/9/rJ1NrDO92vDsAzf7EQzgjX4R0000000001
S:        </launch:claimKey>
S:      </launch:cd>
S:      <launch:cd>
S:        <launch:name exists="1">domain3.example</launch:name>
S:        <launch:claimKey validatorID="tmch">
S:        2013041500/2/6/9/rJ1NrDO92vDsAzf7EQzgjX4R0000000001
S:        </launch:claimKey>
S:        <launch:claimKey validatorID="custom-tmch">
S:        20140423200/1/2/3/rJ1Nr2vDsAzasdff7EasdfgjX4R000000002
S:        </launch:claimKey>
S:      </launch:cd>
S:     </launch:chkData>
S:    </extension>
S:    <trID>
S:     <clTRID>ABC-12345</clTRID>
S:     <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

3.1.2. Availability Check Form

The Availability Check Form defines additional elements to extend the EPP <check> command described in the EPP domain name mapping [RFC5731]. No additional elements are defined for the EPP <check>

response. This form MUST be identified by setting the <launch:check> "type" attribute to "avail".

The EPP <check> command is used to determine if an object can be provisioned within a repository. Domain names may be made available only in unique launch phases, whilst remaining unavailable for concurrent launch phases. In addition to the elements expressed in the <domain:check>, the command is extended with the <launch:check> element that contains the following child elements:

<launch:phase> The launch phase to which domain name availability should be determined.

Example Availability Check Form command using the <check> domain command and the <launch:check> extension with the "type" set to "avail", to determine the availability of two domain names in the "idn-release" custom launch phase:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:   <check>
C:    <domain:check
C:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:      <domain:name>domain1.example</domain:name>
C:      <domain:name>domain2.example</domain:name>
C:    </domain:check>
C:   </check>
C:   <extension>
C:    <launch:check
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:     type="avail">
C:      <launch:phase name="idn-release">custom</launch:phase>
C:    </launch:check>
C:   </extension>
C:   <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

The Availability Check Form does not define any extension to the response of an <check> domain command. After processing the command, the server replies with a standard EPP response as defined in the EPP domain name mapping [RFC5731].

3.1.3. Trademark Check Form

The Trademark Check Form defines a new command called the Trademark Check Command that is used to determine whether or not there are any matching trademarks for each domain name passed in the command, independent of the active launch phase of the server and whether the "Claims Create Form" is required on a Domain Create Command. The availability check information defined in the EPP domain name mapping [RFC5731] MUST NOT be returned for the Trademark Check Command. This form MUST be identified by setting the <launch:check> "type" attribute to "trademark".

Instead of returning whether the domain name is available, the Trademark Check Command will return whether or not at least one matching trademark exists for the domain name. If there is at least one matching trademark that exists for the domain name, a <launch:claimKey> element is returned. The client MAY then use the value of the <launch:claimKey> element to obtain Trademark Claims Notice information from Trademark Validator based on the Validator Identifier (Section 2.2).

The <domain:name> elements in the EPP <check> command of EPP domain name mapping [RFC5731] define the domain names to check for matching trademarks. The <launch:check> element does not contain any child elements with the "Trademark Check Form":

Example Trademark Check command using the <check> domain command and the <launch:check> extension with the "type" set to "trademark", to determine if "domain1.example", "domain2.example", and "domain3.example" have any matching trademarks:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:   <check>
C:    <domain:check
C:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:      <domain:name>domain1.example</domain:name>
C:      <domain:name>domain2.example</domain:name>
C:      <domain:name>domain3.example</domain:name>
C:    </domain:check>
C:   </check>
C:   <extension>
C:    <launch:check
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:     type="trademark"/>
C:   </extension>
C:   <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

If the <check> command has been processed successfully, the EPP <response> MUST contain an <extension> <launch:chkData> element that identifies the launch namespace. The <launch:chkData> element contains the following child elements:

<launch:cd> One or more <launch:cd> elements that contain the following child elements:

<launch:name> Contains the fully qualified name of the queried domain name. This element MUST contain an "exists" attribute whose value indicates if a matching trademark exists for the domain name. A value of "1" (or "true") means that a matching trademark does exist. A value of "0" (or "false") means that a matching trademark does not exist.

<launch:claimKey> Zero or more OPTIONAL claim keys that MAY be passed to a third-party Trademark Validator such as the ICANN Trademark Clearinghouse (TMCH) for querying the information needed to generate a Trademark Claims Notice. The <launch:claimKey> is used as the key for the query in place of the domain name to securely query the service without using a well-known value like a domain name. The OPTIONAL "validatorID" attribute is the Validator Identifier

(Section 2.2) whose value indicates which Trademark Validator to query for the Claims Notice information, with the default being the ICANN TMCH. The "validatorID" attribute MAY reference a non-trademark claims clearinghouse identifer to support other forms of claims notices.

Example Trademark Check response when no matching trademarks are found for the domain name domain1.example, matching trademarks are found for the domain name domain2.example in the "tmch", matching trademarks are found for domain name domain3.example in the "tmch" and "custom-tmch", for the "claims" launch phase:

```
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:    <extension>
S:     <launch:chkData
S:      xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:      <launch:cd>
S:        <launch:name exists="0">domain1.example</launch:name>
S:      </launch:cd>
S:      <launch:cd>
S:        <launch:name exists="1">domain2.example</launch:name>
S:        <launch:claimKey validatorID="tmch">
S:        2013041500/2/6/9/rJ1NrDO92vDsAzf7EQzgjX4R0000000001
S:        </launch:claimKey>
S:      </launch:cd>
S:      <launch:cd>
S:        <launch:name exists="1">domain3.example</launch:name>
S:        <launch:claimKey validatorID="tmch">
S:        2013041500/2/6/9/rJ1NrDO92vDsAzf7EQzgjX4R0000000001
S:        </launch:claimKey>
S:        <launch:claimKey validatorID="custom-tmch">
S:        20140423200/1/2/3/rJ1Nr2vDsAzasdff7EasdfgjX4R000000002
S:        </launch:claimKey>
S:      </launch:cd>
S:     </launch:chkData>
S:    </extension>
S:    <trID>
S:     <clTRID>ABC-12345</clTRID>
S:     <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

3.2. EPP <info> Command

This extension defines additional elements to extend the EPP <info> command and response to be used in conjunction with the EPP domain name mapping [RFC5731].

The EPP <info> command is used to retrieve information for a launch phase registration or application. The Application Identifier (Section 2.1) returned in the <launch:creData> element of the create response (Section 3.3) is used for retrieving information for a Launch Application. A <launch:info> element is sent along with the regular <info> domain command. The <launch:info> element includes an OPTIONAL "includeMark" boolean attribute, with a default value of "false", to indicate whether or not to include the mark in the response. The <launch:info> element contains the following child elements:

<launch:phase> The phase during which the application or registration was submitted or is associated with. Server policy defines the phases that are supported.
<launch:applicationID> OPTIONAL application identifier of the Launch Application.

Example <info> domain command with the <launch:info> extension to retrieve information for the sunrise application for domain.example and application identifier "abc123":

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:   <info>
C:    <domain:info
C:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:      <domain:name>domain.example</domain:name>
C:    </domain:info>
C:   </info>
C:   <extension>
C:    <launch:info
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:       includeMark="true">
C:      <launch:phase>sunrise</launch:phase>
C:      <launch:applicationID>abc123</launch:applicationID>
C:    </launch:info>
C:   </extension>
C:   <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example <info> domain command with the <launch:info> extension to retrieve information for the sunrise registration for domain.example:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:   <info>
C:    <domain:info
C:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:      <domain:name>domain.example</domain:name>
C:    </domain:info>
C:   </info>
C:   <extension>
C:    <launch:info
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:      <launch:phase>sunrise</launch:phase>
C:    </launch:info>
C:   </extension>
C:   <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

If the query was successful, the server replies with a <launch:infData> element along with the regular EPP <resData>. The <launch:infData> contains the following child elements:

<launch:phase> The phase during which the application was submitted, or is associated with, that matches the associated <info> command <launch:phase>.
<launch:applicationID> OPTIONAL Application Identifier of the Launch Application.
<launch:status> OPTIONAL status of the Launch Application using one of the supported status values (Section 2.4).
<mark:mark> Zero or more <mark:mark> (Section 2.6.2) elements.

Example <info> domain response using the <launch:infData> extension with the mark information:

```
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
S:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        <domain:name>domain.example</domain:name>
S:        <domain:roid>EXAMPLE1-REP</domain:roid>
S:        <domain:status s="pendingCreate"/>
S:        <domain:registrant>jd1234</domain:registrant>
S:        <domain:contact type="admin">sh8013</domain:contact>
S:        <domain:contact type="tech">sh8013</domain:contact>
S:        <domain:clID>ClientX</domain:clID>
S:        <domain:crID>ClientY</domain:crID>
S:        <domain:crDate>2012-04-03T22:00:00.0Z</domain:crDate>
S:        <domain:authInfo>
S:          <domain:pw>2fooBAR</domain:pw>
S:        </domain:authInfo>
S:      </domain:infData>
S:    </resData>
S:    <extension>
S:      <launch:infData
S:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:        <launch:phase>sunrise</launch:phase>
S:          <launch:applicationID>abc123</launch:applicationID>
S:          <launch:status s="pendingValidation"/>
S:          <mark:mark
S:            xmlns:mark="urn:ietf:params:xml:ns:mark-1.0">
S:             ...
S:         </mark:mark>
S:      </launch:infData>
S:    </extension>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

3.3. EPP <create> Command

There are four forms of the extension to the EPP <create> command that include the Sunrise Create Form (Section 3.3.1), the Claims Create Form (Section 3.3.2), the General Create Form (Section 3.3.3), and the Mixed Create Form (Section 3.3.4). The form is dependent on the supported launch phases (Section 2.3) as defined below.

sunrise The EPP <create> command with the "sunrise" launch phase is used to submit a registration with trademark information that can be verified by the server with the <domain:name> value. The Sunrise Create Form (Section 3.3.1) is used for the "sunrise" launch phase.

landrush The EPP <create> command with the "landrush" launch phase MAY use the General Create Form (Section 3.3.3) to explicitly specify the phase and optionally define the expected type of object to create.

claims The EPP <create> command with the "claims" launch phase is used to pass the information associated with the presentation and acceptance of the Claims Notice. The Claims Create Form (Section 3.3.2) is used and the General Create Form (Section 3.3.3) MAY be used for the "claims" launch phase.

open The EPP <create> command with the "open" launch phase is undefined but the form supported is up to server policy. Use of the Claims Create Form (Section 3.3.2) MAY be used to pass the information associated with the presentation and acceptance of the Claims Notice if required for the domain name.

custom The EPP <create> command with the "custom" launch phase is undefined but the form supported is up to server policy.

3.3.1. Sunrise Create Form

The Sunrise Create Form of the extension to the EPP domain name mapping [RFC5731] includes the verifiable trademark information that the server uses to match against the domain name to authorize the domain create. A server MUST support one of four models in Claim Validation Models (Section 2.6) to verify the trademark information passed by the client.

A <launch:create> element is sent along with the regular <create> domain command. The <launch:create> element has an OPTIONAL "type" attribute that defines the expected type of object ("application" or "registration") to create. The server SHOULD validate the "type" attribute, when passed, against the type of object that will be created. The <launch:create> element contains the following child elements:

<launch:phase> The identifier for the launch phase.

<launch:codeMark> or <smd:signedMark> or <smd:encodedSignedMark>

<launch:codeMark> Zero or more <launch:codeMark> elements. The <launch:codeMark> child elements are defined in the <launch:codeMark> element (Section 2.6.1) section.
<smd:signedMark> Zero or more <smd:signedMark> elements. The <smd:signedMark> child elements are defined in the <smd:signedMark> element (Section 2.6.3.1) section.
<smd:encodedSignedMark> Zero or more <smd:encodedSignedMark> elements. The <smd:encodedSignedMark> child elements are defined in the <smd:encodedSignedMark> element (Section 2.6.3.2) section.

The following is an example <create> domain command using the <launch:create> extension, following the "code" validation model, with multiple sunrise codes:

```
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>domain.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:        <launch:phase>sunrise</launch:phase>
C:        <launch:codeMark>
C:          <launch:code validatorID="sample1">
C:            49FD46E6C4B45C55D4AC</launch:code>
C:        </launch:codeMark>
C:        <launch:codeMark>
C:          <launch:code>49FD46E6C4B45C55D4AD</launch:code>
C:        </launch:codeMark>
C:        <launch:codeMark>
C:          <launch:code validatorID="sample2">
C:            49FD46E6C4B45C55D4AE</launch:code>
C:        </launch:codeMark>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

The following is an example <create> domain command using the <launch:create> extension, following the "mark" validation model, with the mark information:

```
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>domainone.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:        <launch:phase>sunrise</launch:phase>
C:        <launch:codeMark>
C:          <mark:mark
C:            xmlns:mark="urn:ietf:params:xml:ns:mark-1.0">
C:            ...
C:          </mark:mark>
C:        </launch:codeMark>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

The following is an example <create> domain command using the <launch:create> extension, following the "code with mark" validation model, with a code and mark information:

```
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>domain.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:        <launch:phase>sunrise</launch:phase>
C:        <launch:codeMark>
C:          <launch:code validatorID="sample">
C:            49FD46E6C4B45C55D4AC</launch:code>
C:          <mark:mark
C:           xmlns:mark="urn:ietf:params:xml:ns:mark-1.0">
C:           ...
C:          </mark:mark>
C:        </launch:codeMark>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

The following is an example <create> domain command using the <launch:create> extension, following the "signed mark" validation model, with the signed mark information for a sunrise application:

```
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>domainone.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:       type="application">
C:        <launch:phase>sunrise</launch:phase>
C:        <smd:signedMark id="signedMark"
C:         xmlns:smd="urn:ietf:params:xml:ns:signedMark-1.0">
C:         ...
C:        </smd:signedMark>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

The following is an example <create> domain command using the <launch:create> extension, following the "signed mark" validation model, with the base64 encoded signed mark information:

```
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>domainone.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:        <launch:phase>sunrise</launch:phase>
C:        <smd:encodedSignedMark
C:         xmlns:smd="urn:ietf:params:xml:ns:signedMark-1.0">
C:         ...
C:        </smd:encodedSignedMark>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

3.3.2. Claims Create Form

The Claims Create Form of the extension to the EPP domain name mapping [RFC5731] includes the information related to the registrant’s acceptance of the Claims Notice.

A <launch:create> element is sent along with the regular <create> domain command. The <launch:create> element has an OPTIONAL "type" attribute that defines the expected type of object ("application" or "registration") to create. The server SHOULD validate the "type" attribute, when passed, against the type of object that will be created. The <launch:create> element contains the following child elements:

<launch:phase> Contains the value of the active launch phase of the server. The server SHOULD validate the value against the active server launch phase.
<launch:notice> One or more <launch:notice> elements that contain the following child elements:

<launch:noticeID> Unique notice identifier for the Claims Notice. The <launch:noticeID> element has an OPTIONAL "validatorID" attribute is the Validator Identifier (Section 2.2) whose value indicates which Trademark Validator is the source of the claims notice, with the default being the ICANN TMCH.
<launch:notAfter> Expiry of the claims notice.
<launch:acceptedDate> Contains the date and time that the claims notice was accepted.

The following is an example <create> domain command using the <launch:create> extension with the <launch:notice> information for the "tmch" and the "custom-tmch" validators, for the "claims" launch phase:

```
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>domain.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:        <launch:phase>claims</launch:phase>
C:        <launch:notice>
C:          <launch:noticeID validatorID="tmch">
C:          370d0b7c9223372036854775807</launch:noticeID>
C:          <launch:notAfter>2014-06-19T10:00:00.0Z
C:          </launch:notAfter>
C:          <launch:acceptedDate>2014-06-19T09:00:00.0Z
C:          </launch:acceptedDate>
C:        </launch:notice>
C:        <launch:notice>
C:          <launch:noticeID validatorID="custom-tmch">
C:          470d0b7c9223654313275808</launch:noticeID>
C:          <launch:notAfter>2014-06-19T10:00:00.0Z
C:          </launch:notAfter>
C:          <launch:acceptedDate>2014-06-19T09:00:30.0Z
C:          </launch:acceptedDate>
C:        </launch:notice>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

3.3.3. General Create Form

The General Create Form of the extension to the EPP domain name mapping [RFC5731] includes the launch phase and optionally the object type to create. The OPTIONAL "type" attribute defines the expected type of object ("application" or "registration") to create. The server SHOULD validate the "type" attribute, when passed, against the type of object that will be created.

A <launch:create> element is sent along with the regular <create> domain command. The <launch:create> element contains the following child elements:

<launch:phase> Contains the value of the active launch phase of the server. The server SHOULD validate the value against the active server launch phase.

The following is an example <create> domain command using the <launch:create> extension for a "landrush" launch phase application:

```
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>domain.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:       type="application">
C:        <launch:phase>landrush</launch:phase>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

3.3.4. Mixed Create Form

The Mixed Create Form supports a mix of the create forms, where for example the Sunrise Create Form (Section 3.3.1) and the Claims Create Form (Section 3.3.2) MAY be supported in a single command by including both the verified trademark information and the information related to the registrant’s acceptance of the Claims Notice. The server MAY support the Mixed Create Form. The "custom" launch phase SHOULD be used when using the Mixed Create Form.

The following is an example <create> domain command using the <launch:create> extension, with using a mix of the Sunrise Create Form (Section 3.3.1) and the Claims Create Form (Section 3.3.2) by including both a mark and a notice:

```
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>domainone.example</domain:name>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <launch:create
C:       xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
C:       type="application">
C:        <launch:phase name="non-tmch-sunrise">custom</launch:phase>
C:        <launch:codeMark>
C:          <mark:mark
C:            xmlns:mark="urn:ietf:params:xml:ns:mark-1.0">
C:            ...
C:          </mark:mark>
C:        </launch:codeMark>
C:        <launch:notice>
C:          <launch:noticeID validatorID="tmch">
C:            49FD46E6C4B45C55D4AC
C:          </launch:noticeID>
C:          <launch:notAfter>2012-06-19T10:00:10.0Z
C:          </launch:notAfter>
C:          <launch:acceptedDate>2012-06-19T09:01:30.0Z
C:          </launch:acceptedDate>
C:        </launch:notice>
C:      </launch:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

3.3.5. Create Response

If the create was successful, the server MAY reply with the <launch:creData> element along with the regular EPP <resData> to indicate the server generated Application Identifier (Section 2.1), when multiple applications of a given domain name are supported; otherwise no extension is included with the regular EPP <resData>. The <launch:creData> element contains the following child elements:

<launch:phase> The phase of the application that mirrors the <launch:phase> element included in the <launch:create>.
<launch:applicationID> The application identifier of the application.

An example response when multiple overlapping applications are supported by the server:

```
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:      <domain:creData
S:         xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:       <domain:name>domain.example</domain:name>
S:       <domain:crDate>2010-08-10T15:38:26.623854Z</domain:crDate>
S:      </domain:creData>
S:    </resData>
S:    <extension>
S:      <launch:creData
S:        xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
S:        <launch:phase>sunrise</launch:phase>
S:        <launch:applicationID>2393-9323-E08C-03B1
S:        </launch:applicationID>
S:      </launch:creData>
S:    </extension>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

3.4. EPP <update> Command

This extension defines additional elements to extend the EPP <update> command to be used in conjunction with the domain name mapping.

A client MUST NOT pass the extension on an EPP <update> command to a server that does not support launch applications. A server that does not support launch applications during its launch phase MUST return an EPP error result code of 2102 when receiving an EPP <update> command with the extension.

Registry policies permitting, clients may update an application object by submitting an EPP <update> command along with a <launch:update> element to indicate the application object to be updated. The <launch:update> element contains the following child elements:

<launch:phase> The phase during which the application was submitted or is associated with.
<launch:applicationID> The application identifier for which the client wishes to update.

The following is an example <update> domain command with the <launch:update> extension to add and remove a name server of a sunrise application with the application identifier "abc123":

```
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>domain.example</domain:name>
C:        <domain:add>
C:            <domain:ns>
C:              <domain:hostObj>ns2.domain.example</domain:hostObj>
C:            </domain:ns>
C:          </domain:add>
C:          <domain:rem>
C:            <domain:ns>
C:              <domain:hostObj>ns1.domain.example</domain:hostObj>
C:            </domain:ns>
C:          </domain:rem>
C:      </domain:update>
C:    </update>
C:    <extension>
C:    <launch:update
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:      <launch:phase>sunrise</launch:phase>
C:      <launch:applicationID>abc123</launch:applicationID>
C:    </launch:update>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

This extension does not define any extension to the response of an <update> domain command. After processing the command, the server replies with a standard EPP response as defined in the EPP domain name mapping [RFC5731].

3.5. EPP <delete> Command

This extension defines additional elements to extend the EPP <delete> command to be used in conjunction with the domain name mapping.

A client MUST NOT pass the extension on an EPP <delete> command to a server that does not support launch applications. A server that does not support launch applications during its launch phase MUST return

an EPP error result code of 2102 when receiving an EPP <delete> command with the extension.

Registry policies permitting, clients MAY withdraw an application by submitting an EPP <delete> command along with a <launch:delete> element to indicate the application object to be deleted. The <launch:delete> element contains the following child elements:

<launch:phase> The phase during which the application was submitted or is associated with.
<launch:applicationID> The application identifier for which the client wishes to delete.

The following is an example <delete> domain command with the <launch:delete> extension:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:   <delete>
C:    <domain:delete
C:     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:      <domain:name>domain.example</domain:name>
C:    </domain:delete>
C:   </delete>
C:   <extension>
C:    <launch:delete
C:     xmlns:launch="urn:ietf:params:xml:ns:launch-1.0">
C:      <launch:phase>sunrise</launch:phase>
C:      <launch:applicationID>abc123</launch:applicationID>
C:    </launch:delete>
C:   </extension>
C:   <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

This extension does not define any extension to the response of a <delete> domain command. After processing the command, the server replies with a standard EPP response as defined in the EPP domain name mapping [RFC5731].

3.6. EPP <renew> Command

This extension does not define any extension to the EPP <renew> command or response described in the EPP domain name mapping [RFC5731].

3.7. EPP <transfer> Command

This extension does not define any extension to the EPP <transfer> command or response described in the EPP domain name mapping [RFC5731].

4. Formal Syntax

One schema is presented here that is the EPP Launch Phase Mapping schema.

The formal syntax presented here is a complete schema representation of the object mapping suitable for automated validation of EPP XML instances. The BEGIN and END tags are not part of the schema; they are used to note the beginning and ending of the schema for URI registration purposes.

4.1. Launch Schema

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace="urn:ietf:params:xml:ns:launch-1.0"
  xmlns:launch="urn:ietf:params:xml:ns:launch-1.0"
  xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
  xmlns:mark="urn:ietf:params:xml:ns:mark-1.0"
  xmlns:smd="urn:ietf:params:xml:ns:signedMark-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

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

  <annotation>
    <documentation>
      Extensible Provisioning Protocol v1.0
      domain name extension schema
      for the launch phase processing.
    </documentation>
  </annotation>

  <!--
  Child elements found in EPP commands.
  -->
  <element name="check" type="launch:checkType"/>
  <element name="info" type="launch:infoType"/>
  <element name="create" type="launch:createType"/>
  <element name="update" type="launch:idContainerType"/>
  <element name="delete" type="launch:idContainerType"/>

  <!--
  Common container of id (identifier) element
  -->
  <complexType name="idContainerType">
    <sequence>
      <element name="phase"
      type="launch:phaseType"/>
      <element name="applicationID"
      type="launch:applicationIDType"/>
    </sequence>
  </complexType>

  <!--
```

```
     Definition for application identifier
     -->
     <simpleType name="applicationIDType">
       <restriction base="token"/>
     </simpleType>

     <!--
     Definition for launch phase.  Name is an optional attribute
     used to extend the phase type.  For example, when
     using the phase type value of &qt;custom&gt;, the name
     can be used to specify the custom phase.
     -->
     <complexType name="phaseType">
       <simpleContent>
         <extension base="launch:phaseTypeValue">
           <attribute name="name" type="token"/>
         </extension>
       </simpleContent>
     </complexType>

     <!--
     Enumeration of for launch phase values.
     -->
     <simpleType name="phaseTypeValue">
         <restriction base="token">
           <enumeration value="sunrise"/>
           <enumeration value="landrush"/>
           <enumeration value="claims"/>
           <enumeration value="open"/>
           <enumeration value="custom"/>
       </restriction>
     </simpleType>


     <!--
     Definition for the sunrise code
     -->
     <simpleType name="codeValue">
       <restriction base="token">
         <minLength value="1"/>
       </restriction>
     </simpleType>

     <complexType name="codeType">
       <simpleContent>
         <extension base="launch:codeValue">
           <attribute name="validatorID"
           type="launch:validatorIDType" use="optional"/>
```

```
        </extension>
      </simpleContent>
    </complexType>

    <!--
    Definition for the notice identifier
    -->
    <simpleType name="noticeIDValue">
      <restriction base="token">
        <minLength value="1"/>
      </restriction>
    </simpleType>

    <complexType name="noticeIDType">
      <simpleContent>
        <extension base="launch:noticeIDValue">
          <attribute name="validatorID"
          type="launch:validatorIDType" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>

    <!--
    Definition for the validator identifier
    -->
    <simpleType name="validatorIDType">
      <restriction base="token">
        <minLength value="1"/>
      </restriction>
    </simpleType>

    <!--
    Possible status values for sunrise application
    -->
      <simpleType name="statusValueType">
      <restriction base="token">
          <enumeration value="pendingValidation"/>
          <enumeration value="validated"/>
          <enumeration value="invalid"/>
          <enumeration value="pendingAllocation"/>
          <enumeration value="allocated"/>
          <enumeration value="rejected"/>
          <enumeration value="custom"/>
        </restriction>
      </simpleType>

    <!--
    Status type definition
```

```
     -->
     <complexType name="statusType">
       <simpleContent>
         <extension base="normalizedString">
           <attribute name="s" type="launch:statusValueType"
             use="required"/>
           <attribute name="lang" type="language"
             default="en"/>
           <attribute name="name" type="token"/>
         </extension>
       </simpleContent>
     </complexType>

     <!--
     codeMark Type that contains an optional code
     with mark information.
     -->
     <complexType name="codeMarkType">
       <sequence>
       <element name="code" type="launch:codeType"
         minOccurs="0"/>
       <element ref="mark:abstractMark"
         minOccurs="0"/>
     </sequence>
     </complexType>

     <!--
     Child elements for the create command
     -->
     <complexType name="createType">
       <sequence>
         <element name="phase" type="launch:phaseType"/>
         <choice minOccurs="0">
           <element name="codeMark" type="launch:codeMarkType"
             maxOccurs="unbounded"/>
           <element ref="smd:abstractSignedMark"
            maxOccurs="unbounded"/>
           <element ref="smd:encodedSignedMark"
            maxOccurs="unbounded"/>
         </choice>
         <element name="notice"
          type="launch:createNoticeType"
          minOccurs="0" maxOccurs="unbounded"/>
       </sequence>
       <attribute name="type" type="launch:objectType"/>
     </complexType>

     <!--
```

```
     Type of launch object
     -->
     <simpleType name="objectType">
         <restriction base="token">
           <enumeration value="application"/>
           <enumeration value="registration"/>
       </restriction>
     </simpleType>


     <!--
     Child elements of the create notice element.
     -->
     <complexType name="createNoticeType">
       <sequence>
         <element name="noticeID" type="launch:noticeIDType"/>
         <element name="notAfter" type="dateTime"/>
         <element name="acceptedDate" type="dateTime"/>
       </sequence>
     </complexType>


     <!--
     Child elements of check (Claims Check Command).
     -->
     <complexType name="checkType">
       <sequence>
         <element name="phase" type="launch:phaseType"
           minOccurs="0"/>
       </sequence>
       <attribute name="type" type="launch:checkFormType"
       default="claims"/>
     </complexType>


     <!--
     Type of check form
     (claims check or availability check)
     -->
     <simpleType name="checkFormType">
         <restriction base="token">
           <enumeration value="claims"/>
           <enumeration value="avail"/>
           <enumeration value="trademark"/>
       </restriction>
     </simpleType>
```

```
<!--
Child elements of info command.
-->
<complexType name="infoType">
  <sequence>
    <element name="phase" type="launch:phaseType"/>
    <element name="applicationID"
      type="launch:applicationIDType"
      minOccurs="0"/>
  </sequence>
  <attribute name="includeMark" type="boolean"
    default="false"/>
</complexType>

<!--
Child response elements.
-->
<element name="chkData" type="launch:chkDataType"/>
<element name="creData" type="launch:idContainerType"/>
<element name="infData" type="launch:infDataType"/>

<!--
 <check> response elements.
 -->
<complexType name="chkDataType">
  <sequence>
    <element name="phase" type="launch:phaseType"
     minOccurs="0"/>
    <element name="cd" type="launch:cdType"
     maxOccurs="unbounded"/>
  </sequence>
</complexType>

<complexType name="cdType">
  <sequence>
    <element name="name" type="launch:cdNameType"/>
    <element name="claimKey" type="launch:claimKeyType"
     minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<complexType name="cdNameType">
  <simpleContent>
    <extension base="eppcom:labelType">
      <attribute name="exists" type="boolean"
       use="required"/>
    </extension>
  </simpleContent>
```

```
    </complexType>

    <complexType name="claimKeyType">
      <simpleContent>
        <extension base="token">
          <attribute name="validatorID"
          type="launch:validatorIDType" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>

    <!--
    <info> response elements
    -->
    <complexType name="infDataType">
      <sequence>
        <element name="phase" type="launch:phaseType"/>
       <element name="applicationID"
        type="launch:applicationIDType"
        minOccurs="0"/>
       <element name="status" type="launch:statusType"
        minOccurs="0"/>
        <element ref="mark:abstractMark"
        minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>

  </schema>
  END
```

5. IANA Considerations

5.1. XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688].

Registration request for the launch namespace:

URI: urn:ietf:params:xml:ns:launch-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the launch XML schema:

URI: urn:ietf:params:xml:schema:launch-1.0

Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

5.2. EPP Extension Registry

The EPP extension described in this document should be registered by the IANA in the EPP Extension Registry described in [RFC7451]. The details of the registration are as follows:

Name of Extension: "Launch Phase Mapping for the Extensible Provisioning Protocol (EPP)"

Document status: Standards Track

Reference: (insert reference to RFC version of this document)

Registrant Name and Email Address: IESG, <iesg@ietf.org>

TLDs: Any

IPR Disclosure: None

Status: Active

Notes: None

6. Implementation Status

Note to RFC Editor: Please remove this section and the reference to RFC 7942 [RFC7942] before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942 [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942 [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable

experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

6.1. Verisign EPP SDK

Organization: Verisign Inc.

Name: Verisign EPP SDK

Description: The Verisign EPP SDK includes both a full client implementation and a full server stub implementation of draft-ietf-regext-launchphase.

Level of maturity: Production

Coverage: All aspects of the protocol are implemented.

Licensing: GNU Lesser General Public License

Contact: jgould@verisign.com

URL: http://www.verisigninc.com/en_US/channel-resources/domain-registry-products/epp-sdks

6.2. Verisign Consolidated Top Level Domain (CTLD) SRS

Organization: Verisign Inc.

Name: Verisign Consolidated Top Level Domain (CTLD) Shared Registry System (SRS)

Description: The Verisign Consolidated Top Level Domain (CTLD) Shared Registry System (SRS) implements the server-side of draft-ietf-regext-launchphase for a variety of Top Level Domains (TLD’s).

Level of maturity: Production

Coverage: The "signed mark" Mark Validation Model, the Claims Check Form for the EPP <check> Command, the Sunrise and Claims Forms for the EPP <create> Command of Launch Registrations and Launch Applications. For Launch Applications the Poll Messaging, the EPP <info> Command, the EPP <update> Command, and the EPP <delete> Command is covered.

Licensing: Proprietary

Contact: jgould@verisign.com

6.3. Verisign .COM / .NET SRS

Organization: Verisign Inc.

Name: Verisign .COM / .NET Shared Registry System (SRS)

Description: The Verisign Shared Registry System (SRS) for .COM, .NET and other IDN TLD's implements the server-side of draft-ietf-regext-launchphase.

Level of maturity: Operational Test Environment (OTE)

Coverage: The "signed mark" Mark Validation Model, the Claims Check Form for the EPP <check> Command, the Sunrise and Claims Forms for the EPP <create> Command of Launch Registrations.

Licensing: Proprietary

Contact: jgould@verisign.com

6.4. REngin v3.7

Organization: Domain Name Services (Pty) Ltd

Name: REngin v3.7

Description: Server side implementation only

Level of maturity: Production

Coverage: All features from version 12 have been implemented

Licensing: Proprietary Licensing with Maintenance Contracts

Contact: info@dnservices.co.za

URL: https://www.registry.net.za and soon http://dnservices.co.za

6.5. RegistryEngine EPP Service

Organization: CentralNic

Name: RegistryEngine EPP Service

Description: Generic high-volume EPP service for gTLDs, ccTLDs and SLDs

Level of maturity: Deployed in CentralNic's production environment as well as two other gTLD registry systems, and two ccTLD registry systems.

Coverage: Majority of elements including TMCH sunrise, landrush and TM claims as well as sunrise applications validated using codes.

Licensing: Proprietary In-House software

Contact: epp@centralnic.com

URL: https://www.centralnic.com

6.6. Neustar EPP SDK

Organization: Neustar

Name: Neustar EPP SDK

Description: The Neustar EPP SDK includes client implementation of draft-ietf-regext-launchphase in both Java and C++.

Level of maturity: Production

Coverage: All aspects of the protocol are implemented.

Licensing: GNU Lesser General Public License

Contact: trung.tran@neustar.biz

6.7. gTLD Shared Registry System

Organization: Stichting Internet Domeinnaamregistratie Nederland (SIDN)

Name: gTLD Shared Registry System

Description: The gTLD SRS implements the server side of the draft-ietf-regext-launchphase.

Level of maturity: (soon) Production

Coverage: The following parts of the draft are supported:

Signed mark validation model using Digital Signature (Section 2.6.3)
Claims Check Form (Section 3.1.1)
Sunrise Create Form (Section 3.3.1)

Claims Create Form (Section 3.3.2)

The parts of the document not described here are not implemented.

Licensing: Proprietary

Contact: rik.ribbers@sidn.nl

7. Security Considerations

The mapping extensions described in this document do not provide any security services beyond those described by EPP [RFC5730], the EPP domain name mapping [RFC5731], and protocol layers used by EPP. The security considerations described in these other specifications apply to this specification as well.

Updates to, and deletion of an application object must be restricted to clients authorized to perform the said operation on the object.

As information contained within an application, or even the mere fact that an application exists may be confidential. Any attempt to operate on an application object by an unauthorized client MUST be rejected with an EPP 2201 (authorization error) return code. Server policy may allow <info> operation with filtered output by clients other than the sponsoring client, in which case the <domain:infData> and <launch:infData> response SHOULD be filtered to include only fields that are publicly accessible.

8. Acknowledgements

The authors wish to acknowledge the efforts of the leading participants of the Community TMCH Model that led to many of the changes to this document, which include Chris Wright, Jeff Neuman, Jeff Eckhaus, and Will Shorter.

Special suggestions that have been incorporated into this document were provided by Jothan Frakes, Keith Gaughan, Seth Goldman, Scott Hollenbeck, Michael Holloway, Jan Jansen, Rubens Kuhl, Ben Levac, Gustavo Lozano, Klaus Malorny, Alexander Mayrhofer, Patrick Mevzek, James Mitchell, Francisco Obispo, Mike O'Connell, Bernhard Reutner-Fischer, Trung Tran, Ulrich Wisser and Sharon Wodjenski.

Some of the description of the Trademark Claims Phase was based on the work done by Gustavo Lozano in the ICANN TMCH functional specifications.

9. References

9.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <http://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, <http://www.rfc-editor.org/info/rfc5731>.

[RFC7848] Lozano, G., "Mark and Signed Mark Objects Mapping", RFC 7848, DOI 10.17487/RFC7848, June 2016, <http://www.rfc-editor.org/info/rfc7848>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <http://www.rfc-editor.org/info/rfc7942>.

9.2. Informative References

[RFC7451] Hollenbeck, S., "Extension Registry for the Extensible Provisioning Protocol", RFC 7451, DOI 10.17487/RFC7451, February 2015, <http://www.rfc-editor.org/info/rfc7451>.

Appendix A. Change History

A.1. Change from 00 to 01

1. Changed to use camel case for the XML elements.
2. Replaced "cancelled" status to "rejected" status.
3. Added the child elements of the <claim> element.
4. Removed the XML schema and replaced with "[TBD]".

A.2. Change from 01 to 02

1. Added support for both the ICANN and ARI/Neustar TMCH models.
2. Changed the namespace URI and prefix to use "launch" instead of "launchphase".
3. Added definition of multiple claim validation models.
4. Added the <launch:signedClaim> and <launch:signedNotice> elements.
5. Added support for Claims Info Command

A.3. Change from 02 to 03

1. Removed XSI namespace per Keith Gaughan’s suggestion on the provreg list.
2. Added extensibility to the launch:status element and added the pendingAuction status per Trung Tran’s feedback on the provreg list.
3. Added support for the Claims Check Command, updated the location and contents of the signedNotice, and replaced most references of Claim to Mark based on the work being done on the ARI/Neustar launch model.

A.4. Change from 03 to 04

1. Removed references to the ICANN model.
2. Removed support for the Claims Info Command.
3. Removed use of the signedClaim.
4. Revised the method for referring to the signedClaim from the XML Signature using the IDREF URI.
5. Split the launch-1.0.xsd into three XML schemas including launch-1.0.xsd, signeMark-1.0.xsd, and mark-1.0.xsd.
6. Split the "claims" launch phase to the "claims1" and "claims2" launch phases.
7. Added support for the encodedSignedMark with base64 encoded signedMark.
8. Changed the elements in the createNoticeType to include the noticeID, timestamp, and the source elements.
9. Added the class and effectiveDate elements to mark.

A.5. Change from 04 to 05

1. Removed reference to <smd:zone> in the <smd:signedMark> example.
2. Incorporated feedback from Bernhard Reutner-Fischer on the provreg mail list.
3. Added missing launch XML prefix to applicationIDType reference in the idContainerType of the Launch Schema.
4. Added missing description of the <mark:pc> element in the <mark:addr> element.

5. Updated note on replication of the EPP contact mapping elements in the Mark Contact section.

A.6. Change from 05 to 06

1. Removed the definition of the mark-1.0 and signedMark-1.0 and replaced with reference to draft-lozano-smd, that contains the definition for the mark, signed marked, and encoded signed mark.
2. Split the <launch:timestamp> into <launch:generatedDate> and <launch:acceptedDate> based on feedback from Trung Tran.
3. Added the "includeMark" optional attribute to the <launch:info> element to enable the client to request whether or not to include the mark in the info response.
4. Fixed state diagram to remove redundant transition from "invalid" to "rejected"; thanks Klaus Malorny.

A.7. Change from 06 to 07

1. Proof-read grammar and spelling.
2. Changed "pendingAuction" status to "pendingAllocation", changed "pending" to "pendingValidation" status, per proposal from Trung Tran and seconded by Rubens Kuhl.
3. Added text related to the use of RFC 5731 pendingCreate to the Application Identifier section.
4. Added the Poll Messaging section to define the use of poll messaging for intermediate state transitions and pending action poll messaging for final state transitions.

A.8. Change from 07 to 08

1. Added support for use of the launch statuses and poll messaging for Launch Registrations based on feedback from Sharon Wodjenski and Trung Tran.
2. Incorporated changes based on updates or clarifications in draft-lozano-tmch-func-spec-01, which include:

    1. Removed the unused <launch:generatedDate> element.
    2. Removed the <launch:source> element.
    3. Added the <launch:notAfter> element based on the required <tmNotice:notAfter> element.

A.9. Change from 08 to 09

1. Made <choice> element optional in <launch:create> to allow passing just the <launch:phase> in <launch:create> per request from Ben Levac.
2. Added optional "type" attribute in <launch:create> to enable the client to explicitly define the desired type of object

(application or registration) to create to all forms of the create extension.
3. Added text that the server SHOULD validate the <launch:phase> element in the Launch Phases section.
4. Add the "General Create Form" to the create command extension to support the request from Ben Levac.
5. Updated the text for the Poll Messaging section based on feedback from Klaus Malorny.
6. Replaced the "claims1" and "claims2" phases with the "claims" phase based on discussion on the provreg list.
7. Added support for a mixed create model (Sunrise Create Model and Claims Create Model), where a trademark (encoded signed mark, etc.) and notice can be passed, based on a request from James Mitchell.
8. Added text for the handling of the overlapping "claims" and "landrush" launch phases.
9. Added support for two check forms (claims check form and availability check form) based on a request from James Mitchell. The availability check form was based on the text in draft-rbp-application-epp-mapping.

A.10. Change from 09 to 10

1. Changed noticeIDType from base64Binary to token to be compatible with draft-lozano-tmch-func-spec-05.
2. Changed codeType from base64Binary to token to be more generic.
3. Updated based on feedback from Alexander Mayrhofer, which include:

    1. Changed "extension to the domain name extension" to "extension to the domain name mapping".
    2. Changed use of 2004 return code to 2306 return code when phase passed mismatches active phase and sub-phase.
    3. Changed description of "allocated" and "rejected" statuses.
    4. Moved sentence on a synchronous <domain:create> command without the use of an intermediate application, then an Application Identifier MAY not be needed to the Application Identifier section.
    5. Restructured the Mark Validation Models section to include the "<launch:codeMark> element" sub-section, the "<mark:mark> element" sub-section, and the Digital Signature sub-section.
    6. Changed "Registries may" to "Registries MAY".
    7. Changed "extensed" to "extended" in "Availability Check Form" section.
    8. Broke the mix of create forms in the "EPP <create> Command" section to a fourth "Mixed Create Form" with its own sub-section.

9. Removed "displayed or" from "displayed or accepted" in the <launch:acceptedDate> description.
10. Replaced "given domain name is supported" with "given domain name are supported" in the "Create Response" section.
11. Changed the reference of 2303 (object does not exist) in the "Security Considerations" section to 2201 (authorization error).
12. Added arrow from "invalid" status to "pendingValidation" status and "pendingAllocation" status to "rejected" status in the State Transition Diagram.

4. Added the "C:" and "S:" example prefixes and related text in the "Conventions Used in This Document" section.

A.11. Change from 10 to 11

1. Moved the claims check response <launch:chkData> element under the <extension> element instead of the <resData> element based on the request from Francisco Obispo.

A.12. Change from 11 to 12

1. Added support for multiple validator identifiers for claims notices and marks based on a request and text provided by Mike O'Connell.
2. Removed domain:exDate element from example in section 3.3.5 based on a request from Seth Goldman on the provreg list.
3. Added clarifying text for clients not passing the launch extension on update and delete commands to servers that do not support launch applications based on a request from Sharon Wodjenski on the provreg list.

A.13. Change from 12 to EPPEXT 00

1. Changed to eppext working group draft by changing draft-tan-epp-launchphase to draft-ietf-eppext-launchphase and by changing references of draft-lozano-tmch-smd to draft-ietf-eppext-tmch-smd.

A.14. Change EPPEXT 00 to EPPEXT 01

1. Removed text associated with support for the combining of status values based on feedback from Patrick Mevzek on the provreg mailing list, discussion on the eppext mailing list, and discussion at the eppext IETF meeting on March 6, 2014.

A.15. Change EPPEXT 01 to EPPEXT 02

1. Changed the <launch:claim> element to be zero or more elements and the <launch:notice> element to be one or more elements in the Claims Create Form. These changes were needed to be able to support more than one concurrent claims services.

A.16. Change EPPEXT 02 to EPPEXT 03

1. Added the "Implementation Status" section based on an action item from the eppext IETF-91 meeting.
2. Moved Section 7 "IANA Considerations" and Section 9 "Security Considerations" before Section 5 "Acknowledgements". Moved "Change Log" Section to end.
3. Updated the text for the Claims Check Form and the Claims Create Form to support checking for the need of the claims notice and passing the claims notice outside of the "claims" phase.
4. Added the new Trademark Check Form to support determining whether or not a trademark exists that matches the domain name independent of whether a claims notice is required on create. This was based on a request from Trung Tran and a discussion on the eppext mailing list.

A.17. Change EPPEXT 03 to EPPEXT 04

1. Amended XML Namespace section of IANA Considerations, added EPP Extension Registry section.

A.18. Change EPPEXT 04 to EPPEXT 05

1. Added a missing comma to the descripton of the <launch:phase> element, based on feedback from Keith Gaughan on the eppext mailing list.
2. Added the SIDN implementation status information.
3. Fixed a few indentation issues in the samples.

A.19. Change EPPEXT 05 to EPPEXT 06

1. Removed duplicate "TMCH Functional Specification" URIs based on feedback from Scott Hollenbeck on the eppext mailing list.
2. Changed references of example?.tld to domain?.example to be consistent with RFC 6761 based on feedback from Scott Hollenbeck on the eppext mailing list.
3. A template was added to section 5 to register the XML schema in addition to the namespace based on feedback from Scott Hollenbeck on the eppext mailing list.

A.20. Change EPPEXT 06 to EPPEXT 07

1. Changed reference of lozano-tmch-func-spec to ietf-eppext-tmch-func-spec.

A.21. Change from EPPEXT 07 to REGEXT 00

1. Changed to regext working group draft by changing draft-ietf-eppext-launchphase to draft-ietf-regext-launchphase and by changing references of draft-ietf-eppext-tmch-func-spec to draft-ietf-regext-tmch-func-spec.

A.22. Change from REGEXT 00 to REGEXT 01

1. Fixed reference of Claims Check Command to Trademark Check Command in the Trademark Check Form section.
2. Replaced reference of draft-ietf-eppext-tmch-smd to RFC 7848.

A.23. Change from REGEXT 01 to REGEXT 02

1. Removed the reference to ietf-regext-tmch-func-spec and briefly described the trademark claims phase that is relavent to draft-ietf-regext-launchphase.

A.24. Change from REGEXT 02 to REGEXT 03

1. Ping update.

A.25. Change from REGEXT 03 to REGEXT 04

1. Updates based on feedback from Scott Hollenbeck that include:

    1. Nit on reference to RFC 7848 in section 1.
    2. Added reference to <domain:create> for the request to create a Launch Application in section 2.1.
    3. Removed the second paragraph of section 2.1 describing the option of creating an application identifier for a Launch Registration.
    4. Provided clarification in section 2.2 on the responsibility of the server to ensure that the supported validator identifiers are unique.
    5. Updated the text in section 2.5 referencing the domain name object in RFC 5731.
    6. Updated the copyright to 2017 in section 4.1.

A.26. Change from REGEXT 04 to REGEXT 05

1. Updates based on feedback from Ulrich Wisser that include:

    1. Updated reference to obsoleted RFC 6982 with RFC 7942.
    2. Moved RFC 7451 reference from normative to informative.

Authors' Addresses

James Gould
VeriSign, Inc.
12061 Bluemont Way
Reston, VA 20190
US

Email: jgould@verisign.com
URI: http://www.verisigninc.com

Wil Tan
Cloud Registry
Suite 32 Seabridge House
377 Kent St
Sydney, NSW 2000
AU

Phone: +61 414 710899
Email: wil@cloudregistry.net
URI: http://www.cloudregistry.net

Gavin Brown
CentralNic Ltd
35-39 Mooregate
London, England EC2R 6AR
GB

Phone: +44 20 33 88 0600
Email: gavin.brown@centralnic.com
URI: https://www.centralnic.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: June 10, 2017

L. Zhou
N. Kong
G. Zhou
X. Lee
CNNIC
J. Gould
VeriSign, Inc.

December 7, 2016

Extensible Provisioning Protocol (EPP) Reseller Mapping
draft-ietf-regext-reseller-01

Abstract

This document describes an Extensible Provisioning Protocol (EPP) mapping for provisioning and management of reseller object stored in a shared central repository. Specified in Extensible Markup Language (XML), this extended mapping is applied to provide additional features required for the provisioning of resellers.

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 June 10, 2017.

Copyright Notice

Table of Contents

1. Introduction

Domain resellers are the individuals or companies that act as agents for domain name registrars. A domain name registrar is a direct customer of the domain name registry, is represented as the sponsoring client to the server in [RFC5730], and may have several resellers to help them sell domain names to end users.

This document describes an extension mapping for version 1.0 of the Extensible Provisioning Protocol (EPP) [RFC5730]. This EPP mapping specifies the reseller object mapping.

This document is specified using the XML 1.0 as described in [W3C.REC-xml-20040204] and XML Schema notation as described in [W3C.REC-xmlschema-1-20041028] and [W3C.REC-xmlschema-2-20041028].

2. Conventions Used in This Document

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

In examples, "C:" represents lines sent by a protocol client and "S:" represents lines returned by a protocol server. Indentation and white space in examples are provided only to illustrate element relationships and are not a REQUIRED feature of this specification.

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.

"reseller-1.0" in is used as an abbreviation for "urn:ietf:params:xml:ns:reseller-1.0". The XML namespace prefix "reseller" is used, but implementations MUST NOT depend on it and instead employ a proper namespace-aware XML parser and serializer to interpret and output the XML documents.

3. Object Attributes

An EPP reseller object has attributes and associated values that can be viewed and modified by the sponsoring client or the server. This section describes each attribute type in detail. The formal syntax for the attribute values described here can be found in the "Formal Syntax" section of this document and in the appropriate normative references.

3.1. Reseller Identifier

Reseller identifier provides the ID of the reseller of a sponsoring registrar. Its corresponding element is <reseller:id> defined in this document. All reseller objects are identified by a server-unique identifier.

3.2. Contact and Client Identifiers

All EPP contacts are identified by a server-unique identifier. Contact identifiers are character strings with a specific minimum length, a specified maximum length, and a specified format. Contact identifiers use the "clIDType" client identifier syntax described in [RFC5730].

3.3. Reseller State

A reseller object MUST always have at least one associated state value. Valid values include "ok", "readonly" and "terminated".

State Value Descriptions:

- ok: the normal status value for the reseller object.
- readonly: transform commands submitted with the reseller identifier in the reseller extension would not be allowed.
- terminated: query and transform commands submitted with the reseller identifier in the reseller extension would not be allowed.

3.4. Parent Identifier

There can be more than one layer of resellers. The parent identifier, as defined with the <reseller:parentId> element, represents the parent reseller identifier in a child reseller. The parent identifier is not defined for the top level reseller, namely the registrar of the registry. An N-tier reseller has a parent reseller and at least one child reseller. A reseller customer has a parent reseller and no child resellers.

Loops SHOULD be prohibited. If reseller A has B as parent identifier, reseller B must not have reseller A as parent identifier.

3.5. URL

The URL represents the reseller web home page, as defined with the <reseller:url> element.

3.6. Disclosure of Data Elements and Attributes

This document supports the same disclosure features described in Section 2.9 of with the use of the <reseller:disclose> element. [RFC5733].

The <reseller:disclose> element MUST contain at least one of the following child elements:

<reseller:name type="int"/>

<reseller:name type="loc"/>

<reseller:addr type="int"/>

<reseller:addr type="loc"/>

<reseller:voice/>

<reseller:fax/>

<reseller:email/>

<reseller:url/>

<reseller:contact/>

4. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in the EPP core protocol specification [RFC5730]. The command mappings described here are specifically for use in provisioning and managing reseller information via EPP.

4.1. EPP Query Commands

EPP provides two commands to retrieve domain information: <check> to determine if a reseller object can be provisioned within a repository, and <info> to retrieve detailed information associated with a reseller object. This document does not define a mapping for the EPP <transfer> command.

4.1.1. EPP <check> Command

The EPP <check> command is used to determine if an object can be provisioned within a repository. It provides a hint that allows a client to anticipate the success or failure of provisioning an object using the <create> command, as object-provisioning requirements are ultimately a matter of server policy.

In addition to the standard EPP command elements, the <check> command MUST contain a <reseller:check> element that identifies the reseller namespace. The <reseller:check> element contains the following child elements:

- One or more <reseller:id> elements that contain the server-unique identifier of the reseller objects to be queried.

Example <check> command:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <check>
C:      <reseller:check
C:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
C:        <reseller:id>res1523</reseller:id>
C:        <reseller:id>re1523</reseller:id>
C:        <reseller:id>1523res</reseller:id>
C:      </reseller:check>
C:    </check>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <check> command has been processed successfully, the EPP <resData> element MUST contain a child <reseller:chkData> element that identifies the reseller namespace. The <reseller:chkData> element contains one or more <reseller:cd> elements that contain the following child elements:

- A <reseller:id> element that identifies the queried object. This element MUST contain an "avail" attribute whose value indicates object availability (can it be provisioned or not) at the moment the <check> command was completed. A value of "1" or "true" means that the object can be provisioned. A value of "0" or "false" means that the object cannot be provisioned.

o An OPTIONAL <reseller:reason> element that MAY be provided when an object cannot be provisioned. If present, this element contains server-specific text to help explain why the object cannot be provisioned. This text MUST be represented in the response language previously negotiated with the client; an OPTIONAL "lang" attribute MAY be present to identify the language if the negotiated value is something other than the default value of "en"(English).

Example <check> response:

```
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:      <reseller:chkData
S:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
S:        <reseller:cd>
S:          <reseller:id avail="1">res1523</reseller:id>
S:        </reseller:cd>
S:        <reseller:cd>
S:          <reseller:id avail="0">re1523</reseller:id>
S:          <reseller:reason>In use</reseller:reason>
S:        </reseller:cd>
S:        <reseller:cd>
S:          <reseller:id avail="1">1523res</reseller:id>
S:        </reseller:cd>
S:      </reseller:chkData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

An EPP error response MUST be returned if a <check> command cannot be processed for any reason.

4.1.2. EPP <info> Command

The EPP <info> command is used to retrieve information associated with a reseller object. In addition to the standard EPP command elements, the <info> command MUST contain a <reseller:info> element

that identifies the reseller namespace. The <reseller:info> element contains the following child elements:

- A <reseller:id> element that contains the server-unique identifier of the reseller object to be queried.

Example <info> command:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <info>
C:      <reseller:info
C:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
C:        <reseller:id>res1523</reseller:id>
C:      </reseller:info>
C:    </info>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When an <info> command has been processed successfully, the EPP <resData> element MUST contain a child <reseller:infData> element that identifies the reseller namespace. The <reseller:infData> element contains the following child elements:

- A <reseller:id> element that contains the server-unique identifier of the reseller object, as defined in Section 3.1.

- A <reseller:roid> element that contains the Repository Object IDentifier assigned to the reseller object when the object was created.

- A <reseller:state> element that contains the operational state of the reseller, as defined in Section 3.3.

- An OPTIONAL <reseller:parentId> element that contains the identifier of the parent object, as defined in Section 3.4.

- One or two <reseller:postalInfo> elements that contain postal-address information. Two elements are provided so that address information can be provided in both internationalized and localized forms; a "type" attribute is used to identify the two forms. If an internationalized form (type="int") is provided, element content MUST be represented in a subset of UTF-8 that can be represented in the 7-bit US-ASCII character set. If a localized form (type="loc") is provided, element content MAY be

represented in unrestricted UTF-8. The <reseller:postalInfo> element contains the following child elements:

* A <reseller:name> element that contains the name of the reseller, which SHOULD be the name of the organization.

* A <reseller:addr> element that contains address information associated with the reseller. A <reseller:addr> element contains the following child elements:

  + One, two, or three OPTIONAL <reseller:street> elements that contain the reseller's street address.

  + A <reseller:city> element that contains the reseller's city.

  + An OPTIONAL <reseller:sp> element that contains the reseller's state or province.

  + An OPTIONAL <reseller:pc> element that contains the reseller's postal code.

  + A <reseller:cc> element that contains the reseller's country code.

o An OPTIONAL <reseller:voice> element that contains the reseller's voice telephone number.

o An OPTIONAL <reseller:fax> element that contains the reseller's facsimile telephone number.

o A <reseller:email> element that contains the reseller's email address.

o A <reseller:url> element that contains the URL to the website of the reseller.

o Zero or more OPTIONAL <reseller:contact> elements that contain identifiers for the contact objects to be associated with the reseller object. Contact object identifiers MUST be known to the server before the contact object can be associated with the reseller object. An attribute "type" associated with <reseller:contact> is used to represent contact types. The type values include admin, tech and billing.

o A <reseller:clID> element that contains the identifier of the sponsoring client, who is the domain name registrar.

- A <reseller:crID> element that contains the identifier of the client that created the reseller object.
- A <reseller:crDate> element that contains the date and time of reseller-object creation.
- A <reseller:upID> element that contains the identifier of the client that last updated the reseller object. This element MUST NOT be present if the reseller has never been modified.
- A <reseller:upDate> element that contains the date and time of the most recent reseller-object modification. This element MUST NOT be present if the reseller object has never been modified.
- An OPTIONAL <reseller:disclose> element that identifies elements that require exceptional server-operator handling to allow or restrict disclosure to third parties. See Section 3.6 for a description of the child elements contained within the <reseller:disclose> element.

Example <info> response for the sponsoring client:

```
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:      <reseller:infData
S:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
S:        <reseller:id>res1523</reseller:id>
S:        <reseller:roid>res1523-REP</reseller:roid>
S:        <reseller:state>ok</reseller:state>
S:        <reseller:parentId>1523res</reseller:parentId>
S:        <reseller:postalInfo type="int">
S:          <reseller:name>Example Reseller Inc.</reseller:name>
S:          <reseller:addr>
S:            <reseller:street>123 Example Dr.</reseller:street>
S:            <reseller:street>Suite 100</reseller:street>
S:            <reseller:city>Dulles</reseller:city>
S:            <reseller:sp>VA</reseller:sp>
S:            <reseller:pc>20166-6503</reseller:pc>
S:            <reseller:cc>US</reseller:cc>
S:          </reseller:addr>
S:        </reseller:postalInfo>
S:        <reseller:voice x="1234">+1.7035555555</reseller:voice>
S:        <reseller:fax>+1.7035555556</reseller:fax>
S:        <reseller:email>contact@reseller.example</reseller:email>
S:        <reseller:url>http://reseller.example</reseller:url>
S:        <reseller:contact type="admin">sh8013</reseller:contact>
S:        <reseller:contact type="billing">sh8013</reseller:contact>
S:        <reseller:clID>ClientY</reseller:clID>
S:        <reseller:crID>ClientX</reseller:crID>
S:        <reseller:crDate>1999-04-03T22:00:00.0Z</reseller:crDate>
S:        <reseller:upID>ClientX</reseller:upID>
S:        <reseller:upDate>1999-12-03T09:00:00.0Z</reseller:upDate>
S:        <reseller:disclose flag="0">
S:          <reseller:voice/>
S:          <reseller:email/>
S:        </reseller:disclose>
S:      </reseller:infData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

Example <info> for the non-sponsoring client, according to the disclosure policy:

```
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:      <reseller:infData
S:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
S:        <reseller:id>res1523</reseller:id>
S:        <reseller:roid>res1523-REP</reseller:roid>
S:        <reseller:state>ok</reseller:state>
S:        <reseller:parentId>1523res</reseller:parentId>
S:        <reseller:postalInfo type="int">
S:          <reseller:name>Example Reseller Inc.</reseller:name>
S:          <reseller:addr>
S:            <reseller:street>123 Example Dr.</reseller:street>
S:            <reseller:street>Suite 100</reseller:street>
S:            <reseller:city>Dulles</reseller:city>
S:            <reseller:sp>VA</reseller:sp>
S:            <reseller:pc>20166-6503</reseller:pc>
S:            <reseller:cc>US</reseller:cc>
S:          </reseller:addr>
S:        </reseller:postalInfo>
S:        <reseller:fax>+1.7035555556</reseller:fax>
S:        <reseller:url>http://reseller.example</reseller:url>
S:        <reseller:clID>ClientY</reseller:clID>
S:        <reseller:crID>ClientX</reseller:crID>
S:        <reseller:crDate>1999-04-03T22:00:00.0Z</reseller:crDate>
S:        <reseller:upID>ClientX</reseller:upID>
S:        <reseller:upDate>1999-12-03T09:00:00.0Z</reseller:upDate>
S:      </reseller:infData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

An EPP error response MUST be returned if an <info> command cannot be processed for any reason.

4.1.3. EPP <transfer> Command

The transfer semantics does not apply to reseller object. No EPP <transfer> command is defined in this document.

4.2. EPP Transform Commands

EPP provides four commands to transform reseller-object information: <create> to create an instance of a reseller object, <delete> to delete an instance of a reseller object, <transfer> to manage reseller-object sponsorship changes, and <update> to change information associated with a reseller object. This document does not define a mapping for the EPP <transfer> and <renew> command.

Transform commands are typically processed and completed in real time. Server operators MAY receive and process transform commands but defer completing the requested action if human or third-party review is required before the requested action can be completed. In such situations, the server MUST return a 1001 response code to the client to note that the command has been received and processed but that the requested action is pending. The server MUST also manage the status of the object that is the subject of the command to reflect the initiation and completion of the requested action. Once the action has been completed, all clients involved in the transaction MUST be notified using a service message that the action has been completed and that the status of the object has changed. Other notification methods MAY be used in addition to the required service message.

4.2.1. EPP <create> Command

The EPP <create> command provides a transform operation that allows a client to create a reseller object. In addition to the standard EPP command elements, the <create> command MUST contain a <reseller:create> element that identifies the reseller namespace. The <reseller:create> element contains the following child elements:

- A <reseller:id> element that contains the desired server-unique identifier for the reseller to be created, as defined in Section 3.1.
- A <reseller:state> element that contains the operational status of the reseller, as defined in Section 3.3.
- An OPTIONAL <reseller:parentId> element that contains the identifier of the parent object, as defined in Section 3.4.

- One or two <reseller:postalInfo> elements that contain postal-address information. Two elements are provided so that address information can be provided in both internationalized and localized forms; a "type" attribute is used to identify the two forms. If an internationalized form (type="int") is provided, element content MUST be represented in a subset of UTF-8 that can be represented in the 7-bit US-ASCII character set. If a localized form (type="loc") is provided, element content MAY be represented in unrestricted UTF-8. The <reseller:postalInfo> element contains the following child elements:
  - A <reseller:name> element that contains the name of the reseller, which SHOULD be the name of the organization.
  - A <reseller:addr> element that contains address information associated with the reseller. A <reseller:addr> element contains the following child elements:
    - One, two, or three OPTIONAL <reseller:street> elements that contain the reseller’s street address.
    - A <reseller:city> element that contains the reseller’s city.
    - An OPTIONAL <reseller:sp> element that contains the reseller’s state or province.
    - An OPTIONAL <reseller:pc> element that contains the reseller’s postal code.
    - A <reseller:cc> element that contains the reseller’s country code.
- An OPTIONAL <reseller:voice> element that contains the reseller’s voice telephone number.
- An OPTIONAL <reseller:fax> element that contains the reseller’s facsimile telephone number.
- A <reseller:email> element that contains the reseller’s email address.
- A <reseller:url> element that contains the URL to the website of the reseller.
- Zero or more OPTIONAL <reseller:contact> elements that contain identifiers for the human or organizational social information objects associated with the reseller object.

o An OPTIONAL <reseller:disclose> element that identifies elements that require exceptional server-operator handling to allow or restrict disclosure to third parties. See Section 3.6 for a description of the child elements contained within the <reseller:disclose> element.

Example <create> command:

```
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:      <reseller:create
C:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
C:        <reseller:id>res1523</reseller:id>
C:        <reseller:state>ok</reseller:state>
C:        <reseller:parentId>1523res</reseller:parentId>
C:        <reseller:postalInfo type="int">
C:          <reseller:name>Example Reseller Inc.</reseller:name>
C:          <reseller:addr>
C:            <reseller:street>123 Example Dr.</reseller:street>
C:            <reseller:street>Suite 100</reseller:street>
C:            <reseller:city>Dulles</reseller:city>
C:            <reseller:sp>VA</reseller:sp>
C:            <reseller:pc>20166-6503</reseller:pc>
C:            <reseller:cc>US</reseller:cc>
C:          </reseller:addr>
C:        </reseller:postalInfo>
C:        <reseller:voice x="1234">+1.7035555555</reseller:voice>
C:        <reseller:fax>+1.7035555556</reseller:fax>
C:        <reseller:email>contact@reseller.example</reseller:email>
C:        <reseller:url>http://reseller.example</reseller:url>
C:        <reseller:contact type="admin">sh8013</reseller:contact>
C:        <reseller:contact type="billing">sh8013</reseller:contact>
C:        <reseller:disclose flag="0">
C:          <reseller:voice/>
C:          <reseller:email/>
C:        </reseller:disclose>
C:      </reseller:create>
C:    </create>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <create> command has been processed successfully, the EPP <resData> element MUST contain a child <reseller:creData> element

that identifies the reseller namespace. The <reseller:creData> element contains the following child elements:

- A <reseller:id> element that contains the server-unique identifier for the created reseller, as defined in Section 3.1.
- A <reseller:crDate> element that contains the date and time of reseller-object creation.

Example <create> response:

```
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:      <reseller:creData
S:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
S:        <reseller:id>res1523</reseller:id>
S:        <reseller:crDate>1999-04-03T22:00:00.0Z</reseller:crDate>
S:      </reseller:creData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

An EPP error response MUST be returned if a <create> command cannot be processed for any reason.

4.2.2. EPP <delete> Command

The EPP <delete> command provides a transform operation that allows a client to delete a reseller object. In addition to the standard EPP command elements, the <delete> command MUST contain a <reseller:delete> element that identifies the reseller namespace. The <reseller:delete> element MUST contain the following child element:

- A <reseller:id> element that contains the server-unique identifier of the reseller object, as defined in Section 3.1, to be deleted.

A reseller object SHOULD NOT be deleted if it is associated with other known objects. An associated reseller SHOULD NOT be deleted until associations with other known objects have been broken. A server SHOULD notify clients that object relationships exist by sending a 2305 error response code when a <delete> command is attempted and fails due to existing object relationships.

Example <delete> command:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <delete>
C:      <reseller:delete
C:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
C:        <reseller:id>res1523</reseller:id>
C:      </reseller:delete>
C:    </delete>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <delete> command has been processed successfully, a server MUST respond with an EPP response with no <resData> element.

Example <delete> response:

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

An EPP error response MUST be returned if a <delete> command cannot be processed for any reason.

4.2.3. EPP <renew> Command

Renewal semantics do not apply to reseller objects, so there is no mapping defined for the EPP <renew> command.

4.2.4. EPP <transfer> Command

Transfer semantics do not apply to reseller objects, so there is no mapping defined for the EPP <transfer> command.

4.2.5. EPP <update> Command

The EPP <update> command provides a transform operation that allows a client to modify the attributes of a reseller object. In addition to the standard EPP command elements, the <update> command MUST contain a <reseller:update> element that identifies the reseller namespace. The <reseller:update> element contains the following child elements:

- A <reseller:id> element that contains the server-unique identifier of the reseller object to be updated, as defined in Section 3.1.
- An OPTIONAL <reseller:add> element that contains attribute values to be added to the object.
- An OPTIONAL <reseller:rem> element that contains attribute values to be removed from the object.
- An OPTIONAL <reseller:chg> element that contains attribute values to be changed.

At least one <reseller:add>, <reseller:rem> or <reseller:rem> element MUST be provided if the command is not being extended. All of these elements MAY be omitted if an <update> extension is present. The <reseller:add> and <reseller:rem> elements contain the following child element:

- Zero or more <reseller:contact> elements that contain the identifiers for contact objects to be associated with or removed from the reseller object. Contact object identifiers MUST be known to the server before the contact object can be associated with the reseller object.

A <reseller:chg> element contains the following OPTIONAL child elements. At least one child element MUST be present:

- A <reseller:state> element that contains the operational status of the reseller.

- A <reseller:parentId> element that contains the identifier of the parent object.

- One or two <reseller:postalInfo> elements that contain postal-address information. Two elements are provided so that address information can be provided in both internationalized and localized forms; a "type" attribute is used to identify the two forms. If an internationalized form (type="int") is provided, element content MUST be represented in a subset of UTF-8 that can be represented in the 7-bit US-ASCII character set. If a localized form (type="loc") is provided, element content MAY be represented in unrestricted UTF-8. The <reseller:postalInfo> element contains the following child elements:

  * A <reseller:name> element that contains the name of the reseller, which SHOULD be the name of the organization.

  * A <reseller:addr> element that contains address information associated with the reseller. A <reseller:addr> element contains the following child elements:

    + One, two, or three OPTIONAL <reseller:street> elements that contain the reseller’s street address.

    + A <reseller:city> element that contains the reseller’s city.

    + An OPTIONAL <reseller:sp> element that contains the reseller’s state or province.

    + An OPTIONAL <reseller:pc> element that contains the reseller’s postal code.

    + A <reseller:cc> element that contains the reseller’s country code.

- An <reseller:voice> element that contains the reseller’s voice telephone number.

- An <reseller:fax> element that contains the reseller’s facsimile telephone number.

- A <reseller:email> element that contains the reseller’s email address.

- A <reseller:url> element that contains the URL to the website of the reseller.

o An <reseller:disclose> element that identifies elements that require exceptional server-operator handling to allow or restrict disclosure to third parties. See Section 2.9 in [RFC5733] for a description of the child elements contained within the <reseller:disclose> element.

Example <update> command:

```
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:      <reseller:update
C:       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0">
C:        <reseller:id>res1523</reseller:id>
C:        <reseller:add>
C:          <reseller:contact type="tech">sh8013</reseller:contact>
C:        </reseller:add>
C:        <reseller:chg>
C:          <reseller:state>readonly</reseller:state>
C:          <reseller:postalInfo type="int">
C:            <reseller:addr>
C:              <reseller:street>124 Example Dr.</reseller:street>
C:              <reseller:street>Suite 200</reseller:street>
C:              <reseller:city>Dulles</reseller:city>
C:              <reseller:sp>VA</reseller:sp>
C:              <reseller:pc>20166-6503</reseller:pc>
C:              <reseller:cc>US</reseller:cc>
C:            </reseller:addr>
C:          </reseller:postalInfo>
C:          <reseller:voice>+1.7034444444</reseller:voice>
C:          <reseller:fax/>
C:          <reseller:disclose flag="1">
C:            <reseller:voice/>
C:            <reseller:email/>
C:          </reseller:disclose>
C:        </reseller:chg>
C:      </reseller:update>
C:    </update>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When an <update> command has been processed successfully, a server MUST respond with an EPP response with no <resData> element.

Example <update> response:

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

An EPP error response MUST be returned if an <update> command cannot be processed for any reason.

5. Formal Syntax

An EPP object mapping is specified in XML Schema notation. The formal syntax presented here is a complete schema representation of the object mapping suitable for automated validation of EPP XML instances. The BEGIN and END tags are not part of the schema; they are used to note the beginning and ending of the schema for URI registration purposes.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="urn:ietf:params:xml:ns:reseller-1.0"
       xmlns:reseller="urn:ietf:params:xml:ns:reseller-1.0"
       xmlns:epp="urn:ietf:params:xml:ns:epp-1.0"
       xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0"
       xmlns="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified">

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

 <annotation>
   <documentation>
     Extensible Provisioning Protocol v1.0
```

```
      reseller provisioning schema.
    </documentation>
  </annotation>

<!--
Child elements found in EPP commands.
-->
 <element name="create" type="reseller:createType"/>
 <element name="delete" type="reseller:sIDType"/>
 <element name="update" type="reseller:updateType"/>
 <element name="check" type="reseller:mIDType"/>
 <element name="info" type="reseller:infoType"/>

<!--
Utility types.
-->
<simpleType name="stateType">
  <restriction base="token">
    <enumeration value="ok"/>
    <enumeration value="readonly"/>
    <enumeration value="terminated"/>
  </restriction>
</simpleType>

<complexType name="postalInfoType">
  <sequence>
    <element name="name" type="contact:postalLineType"/>
    <element name="addr" type="contact:addrType"/>
  </sequence>
  <attribute name="type" type="contact:postalInfoEnumType"
   use="required"/>
</complexType>

<complexType name="discloseType">
  <sequence>
    <element name="name" type="contact:intLocType"
     minOccurs="0" maxOccurs="2"/>
    <element name="addr" type="contact:intLocType"
     minOccurs="0" maxOccurs="2"/>
    <element name="voice" minOccurs="0"/>
    <element name="fax" minOccurs="0"/>
    <element name="email" minOccurs="0"/>
    <element name="url" minOccurs="0"/>
    <element name="contact" minOccurs="0"/>
  </sequence>
  <attribute name="flag" type="boolean" use="required"/>
</complexType>
```

```
<!--
Child elements of the <create> command.
-->
 <complexType name="createType">
   <sequence>
     <element name="id" type="eppcom:clIDType"/>
     <element name="state" type="reseller:stateType"
      minOccurs="0"/>
     <element name="parentId" type="eppcom:clIDType"
      minOccurs="0"/>
     <element name="postalInfo" type="reseller:postalInfoType"
      maxOccurs="2"/>
     <element name="voice" type="contact:e164Type"
      minOccurs="0"/>
     <element name="fax" type="contact:e164Type"
      minOccurs="0"/>
     <element name="email" type="eppcom:minTokenType"/>
     <element name="url" type="anyURI"
      minOccurs="0"/>
     <element name="contact" type="domain:contactType"
      minOccurs="0" maxOccurs="3"/>
     <element name="disclose" type="reseller:discloseType"
      minOccurs="0"/>
   </sequence>
 </complexType>

<!--
Child element of commands that require only an identifier.
-->
 <complexType name="sIDType">
   <sequence>
     <element name="id" type="eppcom:clIDType"/>
   </sequence>
 </complexType>

<!--
Child element of commands that accept multiple identifiers.
-->
 <complexType name="mIDType">
   <sequence>
     <element name="id" type="eppcom:clIDType"
      maxOccurs="unbounded"/>
   </sequence>
 </complexType>

<!--
Child elements of the <info> commands.
-->
```

```
       <complexType name="infoType">
         <sequence>
           <element name="id" type="eppcom:clIDType"/>
         </sequence>
       </complexType>

      <!--
      Child elements of the <update> command.
      -->
       <complexType name="updateType">
         <sequence>
           <element name="id" type="eppcom:clIDType"/>
           <element name="add" type="reseller:addRemType"
            minOccurs="0"/>
           <element name="rem" type="reseller:addRemType"
            minOccurs="0"/>
           <element name="chg" type="reseller:chgType"
            minOccurs="0"/>
         </sequence>
       </complexType>

      <!--
      Data elements that can be added or removed.
      -->
       <complexType name="addRemType">
         <sequence>
           <element name="contact" type="domain:contactType"
            minOccurs="0"/>
         </sequence>
       </complexType>

      <!--
      Data elements that can be changed.
      -->
       <complexType name="chgType">
         <sequence>
           <element name="state" type="reseller:stateType"
            minOccurs="0"/>
           <element name="parentId" type="eppcom:clIDType"
            minOccurs="0"/>
           <element name="postalInfo" type="reseller:chgPostalInfoType"
            minOccurs="0" maxOccurs="2"/>
           <element name="voice" type="contact:e164Type"
            minOccurs="0"/>
           <element name="fax" type="contact:e164Type"
            minOccurs="0"/>
           <element name="email" type="eppcom:minTokenType"
            minOccurs="0"/>
```

```
          <element name="url" type="anyURI"
           minOccurs="0"/>
          <element name="disclose" type="reseller:discloseType"
           minOccurs="0"/>
        </sequence>
       </complexType>

       <complexType name="chgPostalInfoType">
        <sequence>
          <element name="name" type="contact:postalLineType"
           minOccurs="0"/>
          <element name="addr" type="contact:addrType"
           minOccurs="0"/>
        </sequence>
        <attribute name="type" type="contact:postalInfoEnumType"
         use="required"/>
       </complexType>

      <!--
      Child response elements.
      -->
       <element name="chkData" type="contact:chkDataType"/>
       <element name="creData" type="contact:creDataType"/>
       <element name="infData" type="reseller:infDataType"/>

      <!--
      <info> response elements.
      -->
       <complexType name="infDataType">
        <sequence>
          <element name="id" type="eppcom:clIDType"/>
          <element name="roid" type="eppcom:roidType"/>
          <element name="state" type="reseller:stateType"/>
          <element name="parentId" type="eppcom:clIDType"
           minOccurs="0"/>
          <element name="postalInfo" type="reseller:postalInfoType"
           maxOccurs="2"/>
          <element name="voice" type="contact:e164Type"
           minOccurs="0"/>
          <element name="fax" type="contact:e164Type"
           minOccurs="0"/>
          <element name="email" type="eppcom:minTokenType"/>
          <element name="url" type="anyURI"
           minOccurs="0"/>
          <element name="contact" type="domain:contactType"
           minOccurs="0" maxOccurs="3"/>
          <element name="clID" type="eppcom:clIDType"/>
          <element name="crID" type="eppcom:clIDType"/>
```

```
          <element name="crDate" type="dateTime"/>
          <element name="upID" type="eppcom:clIDType"
           minOccurs="0"/>
          <element name="upDate" type="dateTime"
           minOccurs="0"/>
          <element name="disclose" type="reseller:discloseType"
           minOccurs="0"/>
        </sequence>
       </complexType>

      <!--
      End of schema.
      -->
      </schema>
      END
```

6. Internationalization Considerations

EPP is represented in XML, which provides native support for encoding information using the Unicode character set and its more compact representations including UTF-8. Conformant XML processors recognize both UTF-8 and UTF-16. Though XML includes provisions to identify and use other character encodings through use of an "encoding" attribute in an <?xml?> declaration, use of UTF-8 is RECOMMENDED.

As an extension of the EPP reseller object mapping, the elements and element content described in this document MUST inherit the internationalization conventions used to represent higher-layer domain and core protocol structures present in an XML instance that includes this extension.

7. IANA Considerations

7.1. XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. IANA is requested to assignment the following URI.

Registration request for the reseller namespace:

- URI: urn:ietf:params:xml:ns:reseller-1.0
- Registrant Contact: See the "Author’s Address" section of this document.
- XML: See the "Formal Syntax" section of this document.

7.2. EPP Extension Registry

The EPP extension described in this document should be registered by the IANA in the EPP Extension Registry described in [RFC7451]. The details of the registration are as follows:

Name of Extension: Domain Reseller Object Extension

Document status: Standards Track

Reference: (insert reference to RFC version of this document)

Registrant Name and Email Address: See the "Author's Address" section of this document.

TLDs: any

IPR Disclosure: none

Status: active

Notes: none

8. Security Considerations

Authorization information described in [RFC5733] is not supported in this document. If the querying client is not the sponsoring registrar of the reseller, not all the object information is accessible. The disclose element defined in [RFC5733] is used to allow or restrict disclosure of object elements to third parties. Other mechanism, such as defining a registry customized authorization information list according to their local policies and regulations, is also possible.

9. Acknowledgement

The authors would like to thank Rik Ribbers, Marc Groeneweg and Patrick Mevzek for their careful review and valuable comments.

10. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

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

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

[RFC7451] Hollenbeck, S., "Extension Registry for the Extensible Provisioning Protocol", RFC 7451, DOI 10.17487/RFC7451, February 2015, <http://www.rfc-editor.org/info/rfc7451>.

[W3C.REC-xml-20040204]
Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, ""Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium FirstEdition REC-xml-20040204", February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204>.

[W3C.REC-xmlschema-1-20041028]
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, ""XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028", October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[W3C.REC-xmlschema-2-20041028]
Biron, P. and A. Malhotra, ""XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028", October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

Appendix A. Change Log

Initial -00: Individual document submitted.

-01:

* Updated abstract text.

* Added sentences to avoid loop of parent identifiers in section 3.4.

* Revised typos in section 3.6.

* Added explanation of contact type attribute in section 4.1.2.

* Updated <info> responses.

* Deleted description of <transfer> command in section 4.1 and 4.2.

* Deleted whoisInfo disclose type in XML schema.

* Deleted maxOccurs of addRemType.

* Deleted extra "OPTIONAL" in section 4.2.5.

* Updated typos in <update> response.

-02:

* Changed author information.

* Updated url definition.

* Updated XML schema.

-03:

* Changed author information.

* Updated section 3.1.

* Refactored the XSD file. Added <chgPostalInfoType> element.

* Added acknowledgement.

WG document-00: WG document submitted

WG document-01: Keep document alive for further discussion. Reseller object or entity object with multiple roles?

Authors' Addresses

Linlin Zhou
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 2677
Email: zhoulinlin@cnnic.cn

Ning Kong
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3147
Email: nkong@cnnic.cn

Guiqing Zhou
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 2692
Email: zhouguiqing@cnnic.cn

Xiaodong Lee
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3020
Email: xl@cnnic.cn

James Gould
VeriSign, Inc.
12061 Bluemont Way
Reston, VA 20190
US

Email: jgould@verisign.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: June 10, 2017

L. Zhou
N. Kong
J. Wei
X. Lee
CNNIC
J. Gould
VeriSign, Inc.

December 7, 2016

Reseller Extension for the Extensible Provisioning Protocol (EPP)
draft-ietf-regext-reseller-ext-01

Abstract

This mapping, an extension to EPP object mappings like the EPP domain name mapping [RFC5731], to support assigning a reseller to any existing object (domain, host, contact) as well as any future objects. Specified in Extensible Markup Language (XML), this extended mapping is applied to provide additional features required for the provisioning of resellers.

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 June 10, 2017.

Copyright Notice

Table of Contents

1. Introduction

Domain resellers are the individuals or companies act as agents for ICANN accredited registrars. A domain name registrar may have several resellers to help them sell domain names to end users.

Generally speaking, resellers provide domain registration information via registrar's EPP client without reseller information. On one hand, registrars are concerned about how to identify resellers. On the other hand, end users would also be confused by the WHOIS service without corresponding reseller information. This requirement imposes a challenge for the domain registries since there is no definition of resellers in the existing EPP domain name mapping. Out of band method could solve this problem but may increase extra cost.

In order to facilitate provisioning and management of reseller information in a shared central repository, this document proposes a reseller extension of [RFC5731], [RFC5732] and [RFC5733]. The examples provided in this document are used for the domain object for illustration purposes. The host and contact object could be extended in the same way with the domain object.

A reseller mapping object defined in [ID.draft-ietf-regext-reseller] SHOULD be created first. The reseller information specified in this document SHOULD reference the existing reseller identifier and reseller name.

This document is specified using the XML 1.0 as described in [W3C.REC-xml-20040204] and XML Schema notation as described in [W3C.REC-xmlschema-1-20041028] and [W3C.REC-xmlschema-2-20041028].

2. Conventions Used in This Document

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

In examples, "C:" represents lines sent by a protocol client and "S:" represents lines returned by a protocol server. Indentation and white space in examples are provided only to illustrate element relationships and are not a REQUIRED feature of this specification.

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.

resellerext-1.0 in this document is used as an abbreviation for urn:ietf:params:xml:ns:resellerext-1.0.

3. Object Attributes

This extension adds additional elements to the EPP domain name mapping [RFC5731]. Only the new elements are described here.

3.1. Reseller Identifier

Reseller identifier provides the ID of the reseller of a sponsoring registrar. Its corresponding element is <resellerext:id> which refers to the <reseller:id> element defined in [ID.draft-ietf-regext-reseller]. All reseller objects are identified by a server-unique identifier

3.2. Reseller Name

Reseller name provides the name of the reseller of a sponsoring registrar. Its corresponding element is <resellerext:name> which refers to the <reseller:name> element defined in [ID.draft-ietf-regext-reseller].

4. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in the EPP core protocol specification [RFC5730]. The command mappings described here are specifically for use in provisioning and managing reseller information via EPP.

4.1. EPP Query Commands

EPP provides three commands to retrieve domain information: <check> to determine if a domain object can be provisioned within a repository, <info> to retrieve detailed information associated with a domain object, and <transfer> to retrieve domain-object transfer status information.

4.1.1. EPP <check> Command

This extension does not add any elements to the EPP <check> command or <check> response described in the EPP domain name mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733].

4.1.2. EPP <info> Command

This extension does not add any element to the EPP <info> command described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733]. However, additional elements are defined for the <info> response.

Example <info> command:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <command>
C:    <info>
C:      <domain:info xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:        <domain:authInfo>
C:          <domain:pw>fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:info>
C:    </info>
C:    <clTRID>ngcl-mIFICBNP</clTRID>
C:  </command>
C:</epp>
```

When an <info> command has been processed successfully, the EPP <resData> element MUST contain child elements as described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733]. In addition, the EPP <extension> element SHOULD contain a child <resellerext:infData> element that identifies the extension namespace if the domain object has data associated with this extension and based on its service policy. The <resellerext:infData> element contains the following child elements:

- A <resellerext:id> element that contains the identifier of the reseller of a sponsoring registrar.

Example <info> response for an authorized client:

```
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 lang="en-US">Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <domain:infData xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        <domain:name>example.com</domain:name>
S:        <domain:roid>EXAMPLE1-REP</domain:roid>
S:        <domain:status s="ok"/>
S:        <domain:registrant>jd1234</domain:registrant>
S:        <domain:contact type="admin">sh8013</domain:contact>
S:        <domain:contact type="billing">sh8013</domain:contact>
S:        <domain:contact type="tech">sh8013</domain:contact>
S:        <domain:ns>
S:          <domain:hostObj>ns1.example.com</domain:hostObj>
S:        </domain:ns>
S:        <domain:clID>ClientX</domain:clID>
S:        <domain:crID>ClientY</domain:crID>
S:        <domain:crDate>2015-02-06T04:01:21.0Z</domain:crDate>
S:        <domain:exDate>2018-02-06T04:01:21.0Z</domain:exDate>
S:        <domain:authInfo>
S:          <domain:pw>2fooBAR</domain:pw>
S:        </domain:authInfo>
S:      </domain:infData>
S:    </resData>
S:    <extension>
S:      <rgp:infData xmlns:rgp="urn:ietf:params:xml:ns:rgp-1.0">
S:        <rgp:rgpStatus s="addPeriod"/>
S:      </rgp:infData>
S:      <resellerext:infData xmlns:resellerext="urn:ietf:params:xml:ns:resellere
xt-1.0">
S:        <resellerext:id>myreseller</resellerext:id>
S:      </resellerext:infData>
S:    </extension>
S:    <trID>
S:      <clTRID>ngcl-IvJjzMZc</clTRID>
S:      <svTRID>test142AWQONJZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

<info> response for the unauthorized client has not been changed, see [RFC5731], [RFC5732] and [RFC5733]for detail.

An EPP error response MUST be returned if an <info> command cannot be processed for any reason.

4.1.3. EPP <transfer> Command

This extension does not add any elements to the EPP <transfer> command or <transfer> response described in the EPP domain name mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733].

4.2. EPP Transform Commands

EPP provides five commands to transform domain objects: <create> to create an instance of a domain object, <delete> to delete an instance of a domain object, <renew> to extend the validity period of a domain object, <transfer> to manage domain object sponsorship changes, and <update> to change information associated with a domain object.

4.2.1. EPP <create> Command

This extension defines additional elements for the EPP <create> command described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733]. No additional elements are defined for the EPP <create> response.

The EPP <create> command provides a transform operation that allows a client to create a domain object. In addition to the EPP command elements described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733], the command MUST contain an <extension> element, and the <extension> element MUST contain a child <resellerext:create> element that identifies the extension namespace if the client wants to associate data defined in this extension to the domain object. The <resellerext:create> element contains the following child elements:

- A <resellerext:id> element that contains the identifier of the reseller of a sponsoring registrar.

Example <create> Command:

```
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 xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:        <domain:period unit="y">3</domain:period>
C:        <domain:ns>
C:          <domain:hostObj>ns1.example.com</domain:hostObj>
C:        </domain:ns>
C:        <domain:registrant>jd1234</domain:registrant>
C:        <domain:contact type="tech">sh8013</domain:contact>
C:        <domain:contact type="billing">sh8013</domain:contact>
C:        <domain:contact type="admin">sh8013</domain:contact>
C:        <domain:authInfo>
C:          <domain:pw roid="ddddd-dddd">fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:create>
C:    </create>
C:    <extension>
C:      <resellerext:create xmlns:resellerext="urn:ietf:params:xml:ns:resellerex
t-1.0">
C:        <resellerext:id>myreseller</resellerext:id>
C:      </resellerext:create>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <create> command has been processed successfully, the EPP response is as described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733].

An EPP error response MUST be returned if a <create> command cannot be processed for any reason.

4.2.2. EPP <delete> Command

This extension does not add any elements to the EPP <delete> command or <delete> response described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733].

4.2.3. EPP <renew> Command

This extension does not add any elements to the EPP <renew> command or <renew> response described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733].

4.2.4. EPP <transfer> Command

This extension does not add any elements to the EPP <transfer> command or <transfer> response described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733], but after a successful transfer of an object with an assigned reseller, the server SHOULD clear the assigned reseller value.

4.2.5. EPP <update> Command

This extension defines additional elements for the EPP <update> command described in the EPP domain mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733]. No additional elements are defined for the EPP <update> response.

The EPP <update> command provides a transform operation that allows a client to modify the attributes of a domain object. In addition to the EPP command elements described in the EPP domain mapping, the command MUST contain an <extension> element, and the <extension> element MUST contain a child <resellerext:update> element that identifies the extension namespace if the client wants to update the domain object with data defined in this extension. The <resellerext:update> element contains the following child elements:

- An OPTIONAL <resellerext:add> element that contains attribute values to be added to the object.
- An OPTIONAL <resellerext:rem> element that contains attribute values to be removed from the object.
- An OPTIONAL <resellerext:chg> element that contains attribute values to be changed.

At least one and only one <resellerext:add>, <resellerext:rem> or <resellerext:rem> element MUST be provided. The <resellerext:add>, <resellerext:rem> and <resellerext:rem> elements contain the following child element:

- A <resellerext:id> element that contains the identifier of the reseller of a sponsoring registrar.

Example <update> command, adding a reseller:

```
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 xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:      </domain:update>
C:    </update>
C:    <extension>
C:      <resellerext:update xmlns:resellerext="urn:ietf:params:xml:ns:resellerex
t-1.0">
C:        <resellerext:add>
C:          <resellerext:id>myreseller</resellerext:id>
C:        </resellerext:add>
C:      </resellerext:update>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example <update> command, removing a reseller:

```
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 xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:      </domain:update>
C:    </update>
C:    <extension>
C:      <resellerext:update xmlns:resellerext="urn:ietf:params:xml:ns:resellerex
t-1.0">
C:        <resellerext:rem>
C:          <resellerext:id>myreseller</resellerext:id>
C:        </resellerext:rem>
C:      </resellerext:update>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example <update> command, updating reseller identifier:

```
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 xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:      </domain:update>
C:    </update>
C:    <extension>
C:      <resellerext:update xmlns:resellerext="urn:ietf:params:xml:ns:resellerex
t-1.0">
C:        <resellerext:chg>
C:          <resellerext:id>myreseller</resellerext:id>
C:        </resellerext:chg>
C:      </resellerext:update>
C:    </extension>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When an extended <update> command has been processed successfully, the EPP response is as described in the EPP domain name mapping [RFC5731], host mapping [RFC5732] and contact mapping [RFC5733].

5. Formal Syntax

An EPP object mapping is specified in XML Schema notation. The formal syntax presented here is a complete schema representation of the object mapping suitable for automated validation of EPP XML instances. The BEGIN and END tags are not part of the schema; they are used to note the beginning and ending of the schema for URI registration purposes.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>

<schema targetNamespace="urn:ietf:params:xml:ns:resellerext-1.0"
        xmlns:resellerext="urn:ietf:params:xml:ns:resellerext-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"
        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>
      Extensible Provisioning Protocol v1.0
      Domain Reseller Extension Schema v1.0
    </documentation>
  </annotation>

<!-- Child elements found in EPP commands. -->
  <element name="create" type="resellerext:createType"/>
  <element name="update" type="resellerext:updateType"/>

<!-- Child elements of the <resellerext:create> command
All elements must be present at time of creation
-->
  <complexType name="createType">
    <sequence>
      <!-- agent identifier that sells the domain, e.g. registrar, reseller -->
      <element name="id" type="eppcom:clIDType"/>
    </sequence>
  </complexType>

<!--
Child elements of <resellerext:update> command
-->

  <complexType name="updateType">
    <sequence>
      <element name="add" type="resellerext:addRemChgType" minOccurs="0"/>
      <element name="rem" type="resellerext:addRemChgType" minOccurs="0"/>
      <element name="chg" type="resellerext:addRemChgType" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="addRemChgType">
    <sequence>
      <!-- agent identifier that sells the domain, e.g. registrar, reseller -->
      <element name="id" type="eppcom:clIDType" minOccurs="0"/>
    </sequence>
  </complexType>

<!-- Child response element -->

  <element name="infData" type="resellerext:infDataType"/>
```

```
<!-- <resellerext:infData> response elements -->

  <complexType name="infDataType">
    <sequence>
      <!-- agent identifier that sells the domain, e.g. registrar, reseller -->
      <element name="id" type="eppcom:clIDType" minOccurs="0"/>
    </sequence>
  </complexType>

<!-- End of schema. -->
</schema>
END
```

6. Internationalization Considerations

EPP is represented in XML, which provides native support for encoding information using the Unicode character set and its more compact representations including UTF-8. Conformant XML processors recognize both UTF-8 and UTF-16. Though XML includes provisions to identify and use other character encodings through use of an "encoding" attribute in an <?xml?> declaration, use of UTF-8 is RECOMMENDED.

As an extension of the EPP domain name mapping, the elements, element content described in this document MUST inherit the internationalization conventions used to represent higher-layer domain and core protocol structures present in an XML instance that includes this extension.

7. IANA Considerations

7.1. XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. IANA is requested to assignment the following URI.

Registration request for the reseller namespace:

- URI: urn:ietf:params:xml:ns:reseller-1.0
- Registrant Contact: See the "Author’s Address" section of this document.
- XML: See the "Formal Syntax" section of this document.

7.2. EPP Extension Registry

The EPP extension described in this document should be registered by the IANA in the EPP Extension Registry described in [RFC7451]. The details of the registration are as follows:

Name of Extension: Domain Reseller Extension

Document status: Standards Track

Reference: (insert reference to RFC version of this document)

Registrant Name and Email Address: See the "Author’s Address" section of this document.

TLDs: any

IPR Disclosure: none

Status: active

Notes: none

8. Security Considerations

The object mapping extension described in this document does not provide any other security services or introduce any additional considerations beyond those described by [RFC5730], [RFC5731], [RFC5732] and [RFC5733] or those caused by the protocol layers used by EPP.

9. Acknowledgement

The authors would like to thank Rik Ribbers, Marc Groeneweg and Patrick Mevzek for their careful review and valuable comments.

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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5733>.

[RFC7451] Hollenbeck, S., "Extension Registry for the Extensible Provisioning Protocol", RFC 7451, DOI 10.17487/RFC7451, February 2015, <http://www.rfc-editor.org/info/rfc7451>.

[W3C.REC-xml-20040204] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, ""Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium FirstEdition REC-xml-20040204", February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204>.

[W3C.REC-xmlschema-1-20041028] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, ""XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028", October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[W3C.REC-xmlschema-2-20041028] Biron, P. and A. Malhotra, ""XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028", October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

10.2. Informative References

[ID.draft-ietf-regext-reseller] Zhou, L., Kong, N., Zhou, G., Lee, X., and J. Gould, "Extensible Provisioning Protocol (EPP) Reseller Mapping", Jun 2016, <http://tools.ietf.org/html/draft-ietf-regext-reseller>.

Appendix A. Change Log

Initial -00: Individual document submitted.

-01:

* Updated abstract and introduction.
* Revised typos in info response.
* Added explanations on how to process reseller extension after successful transfer operation.
* Modified <update> explanation.
* Deleted reseller name element in <create> and <update> commands.
* Removed some inaccurate comments from xml schema.
* Modified the element name of reseller id and reseller name.

-02:

* Changed author information.
* Updated xml typos <reseller:infData> to <resellerext:infData> in <info> response.

-03:

* Changed author information.
* Updated section 3.1.
* Removed reseller name element in <info> response.
* Added acknowledgement.
* Revised the typo "resellerr" to "resellerext".

WG document-00: WG document submitted

WG document-01: Keep document alive for further discussion. The requirement of reseller information is clear for both registrar and registry. What we should reach a consensus is whether the extension should support only a name or ID and name.

Authors' Addresses

Linlin Zhou
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 2677
Email: zhoulinlin@cnnic.cn

Ning Kong
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3147
Email: nkong@cnnic.cn

Junkai Wei
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3494
Email: weijunkai@cnnic.cn

Xiaodong Lee
CNNIC
4 South 4th Street, Zhongguancun, Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3020
Email: xl@cnnic.cn

James Gould
VeriSign, Inc.
12061 Bluemont Way
Reston, VA 20190
US

Email: jgould@verisign.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 19, 2017

A. Newton
ARIN
M. Sanz
DENIC eG

December 16, 2016

Using RDAP as a Domain Availability Service
draft-newton-regext-rdap-domain-availability-00

Abstract

This document describes a minimal profile of RDAP which can be used to check the availability of domain names available for registration.

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 June 19, 2017.

Copyright Notice

1. Introduction

While RDAP [RFC7480] has all the necessary properties to serve the information necessary to determine if a domain name is available for registration, without proper signalling an RDAP server is likely to return more information than is required to fulfill this requirement, thus wasting bandwidth and server resources, and may return negative answers when positive answers are more appropriate.

This document defines HTTP query parameters to be used by clients to signal the intent of the query is for an availability check. Servers may then respond with a much smaller and less costly payload than they would have otherwise. The contents of the payload would be at the discretion of the policies of the server operator within the guidelines of RDAP responses [RFC7483].

2. Query Parameter

To signal that the client is only interested in domain availability checking, the client MUST use the query parameter 'availabilityCheck' in the RDAP query for a domain with value '1'. This query parameter has no meaning for other RDAP queries, and therefore should be ignored in those cases.

The following is an example RDAP query for a domain availability check.

```
https://example.com/rdap/domain/example.com?availabilityCheck=1
```

Figure 1: Example RDAP Query

The following is an example response.

```
{
  "rdapConformance" : [ "rdap_level_0", "domain_check_0" ],
  "objectClassName" : "domain",
  "ldhName" : "example.com",
  "status" : [ "inactive", "redemption period" ]
}
```

This response, not including the HTTP headers, is 174 octets long.

Figure 2: Example RDAP Response

When this query parameter is given, servers MUST reply with HTTP return codes as specified in Section 3.

3. Positive and Negative Answers

RDAP [RFC7480] provides guidance on using HTTP return codes more contextual for the lookup of domain registration information than domain availability information. A query for domain registration information should result in a negative answer (e.g. 404 NOT FOUND) if the name is not registered (or not in active use or in another state where it may be considered to have a unregistration according to policy). However, a query for domain availability information upon the same domain name should yield a positive answer if the domain is available for registration within the bounds of the registry policy. In other words, the return codes may vary given the appropriate context.

In the context of domain availability, RDAP servers should return positive answers (in accordance with server policies for rate limiting and access control) if the domain registry for which the RDAP server is responding allows registration of the queried domain even if the domain is currently registered at the time of the query. For example, an RDAP server answering for foo.example would return a positive answer (i.e. 200 OK) for fiz.foo.example if the domain status was active or inactive, but it would return a negative answer for buzz.bar.example because it is not the registry for bar.example.

The example in Figure 2 shows a positive answer for a domain that is inactive (i.e. not registered). This example shows a positive answer for a domain that is active, including variants and the expiration date.

```
{
  "rdapConformance" : [ "rdap_level_0", "domain_check_0" ],
  "objectClassName" : "domain",
  "handle" : "XXXX",
  "ldhName" : "xn--fo-5ja.example",
  "unicodeName" : "f&#8730;>=o.example",
  "variants" :
  [
    {
      "relation" : [ "registered", "conjoined" ],
      "variantNames" :
      [
        {
          "ldhName" : "xn--fo-cka.example",
          "unicodeName" : "f&#8730;[micro]o.example"
        },
        {
          "ldhName" : "xn--fo-fka.example",
          "unicodeName" : "f&#8730;&#8706;o.example"
        }
      ]
    },
    {
      "relation" : [ "unregistered", "registration restricted" ],
      "variantNames" :
      [
        {
          "ldhName": "xn--fo-8ja.example",
          "unicodeName" : "f&#8730;JPYo.example"
        }
      ]
    }
  ],
  "status" : [ "active", "transfer prohibited" ],
  "events" :
  [
    {
      "eventAction" : "expiration",
      "eventDate" : "2019-12-31T23:59:59Z"
    }
  ]
}
```

This response, not including the HTTP headers, is 908 octets long in UTF-8 encoding.

Figure 3: Example Positive Answer

4. Availability Information Query

All of the examples given in this document have been reduced from examples in [RFC7483], demonstrating that domain availability information is a subset of domain registration information in the RDAP data model. Some clients may desire both domain registration information and domain availability information. To signal that the client is interested in both contexts, the client MUST use the query parameter 'availabilityInformation' in the RDAP query for a domain with value '1'. This query parameter has no meaning for other RDAP queries, and therefore should be ignored in those cases.

Servers MUST respond with positive answers as specified in Section 3, but SHOULD NOT redact other registration information (i.e. entities, name servers, etc...) in accordance with their access policies.

5. RDAP Conformance

Servers MUST include "domain_check_0" in the rdapConformance array of their response when modifying their behavior according to this specification. That is, this value should only be placed in the array when responding to a query with one of the query parameters in Section 2 or Section 4.

6. Normative References

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", RFC 7480, DOI 10.17487/RFC7480, March 2015, <http://www.rfc-editor.org/info/rfc7480>.

[RFC7483] Newton, A. and S. Hollenbeck, "JSON Responses for the Registration Data Access Protocol (RDAP)", RFC 7483, DOI 10.17487/RFC7483, March 2015, <http://www.rfc-editor.org/info/rfc7483>.

Authors' Addresses

Andrew Lee Newton
American Registry for Internet Numbers
PO Box 232290
Centreville, VA 20120
US

Email: andy@arin.net
URI: http://www.arin.net

Marcos Sanz
DENIC eG
Kaiserstrasse 75 - 77
Frankfurt am Main 60329
Germany

Email: sanz@denic.de
URI: https://www.denic.de