[Note that this file is a concatenation of more than one RFC.]

Network Working Group
Request for Comments: 5730
STD: 69
Obsoletes: 4930
Category: Standards Track

S. Hollenbeck
VeriSign, Inc.
August 2009

Extensible Provisioning Protocol (EPP)

Abstract

This document describes an application-layer client-server protocol for the provisioning and management of objects stored in a shared central repository. Specified in XML, the protocol defines generic object management operations and an extensible framework that maps protocol operations to objects. This document includes a protocol specification, an object mapping template, and an XML media type registration. This document obsoletes RFC 4930.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes specifications for the Extensible Provisioning Protocol (EPP) version 1.0, an XML text protocol that permits multiple service providers to perform object-provisioning operations using a shared central object repository. EPP is specified using the Extensible Markup Language (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]. EPP meets and exceeds the requirements for a generic registry registrar protocol as described in [RFC3375]. This document obsoletes RFC 4930 [RFC4930].

EPP content is identified by MIME media type application/epp+xml. Registration information for this media type is included in an appendix to this document.

EPP is intended for use in diverse operating environments where transport and security requirements vary greatly. It is unlikely that a single transport or security specification will meet the needs of all anticipated operators, so EPP was designed for use in a layered protocol environment. Bindings to specific transport and security protocols are outside the scope of this specification.

The original motivation for this protocol was to provide a standard Internet domain name registration protocol for use between domain name registrars and domain name registries. This protocol provides a means of interaction between a registrar’s applications and registry applications. It is expected that this protocol will have additional uses beyond domain name registration.

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.

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 [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 REQUIRED features of this protocol. A protocol client that is authorized to manage an existing object is described as a "sponsoring" client throughout this document.

2. Protocol Description

EPP is a stateful XML protocol that can be layered over multiple transport protocols. Protected using lower-layer security protocols, clients exchange identification, authentication, and option information, and then engage in a series of client-initiated command-response exchanges. All EPP commands are atomic (there is no partial success or partial failure) and designed so that they can be made idempotent (executing a command more than once has the same net effect on system state as successfully executing the command once).

EPP provides four basic service elements: service discovery, commands, responses, and an extension framework that supports definition of managed objects and the relationship of protocol requests and responses to those objects.

An EPP server MUST respond to client-initiated communication (which can be either a lower-layer connection request or an EPP service discovery message) by returning a greeting to a client. A server MUST promptly respond to each EPP command with a coordinated response that describes the results of processing the command. The following server state machine diagram illustrates the message exchange process in detail:

```
              |
              V
      +-----------------+                  +-----------------+
      |   Waiting for   |     Connected    |     Prepare     |
      |      Client     |----------------->|     Greeting    |
      +-----------------+    or <hello>    +-----------------+
         ^                                           |
         | Close Connection                     Send |
         |     or Idle                      Greeting |
      +-----------------+                            V
      |       End       |     Timeout      +-----------------+
      |     Session     |<-----------------|   Waiting for   |
      +-----------------+                  |      Client     |
         ^    ^    ^        Send +-------->|  Authentication |
         |    |    |    Response |         +-----------------+
         |    |    |     +--------------+            |
         |    |    |     | Prepare Fail |            | <login>
         |    |    +-----|   Response   |            | Received
         |    |    Send  +--------------+            V
         |    |    2501          ^         +-----------------+
         |    |   Response       |         |   Processing    |
         |    |                  +---------|     <login>     |
         |    |                  Auth Fail +-----------------+
         |    |       Timeout                         |
         |    +-------------------------------+       | Auth OK
         |                                    |       V
         |   +-----------------+  <hello>  +-----------------+
         |   |     Prepare     |<----------|   Waiting for   |
         |   |     Greeting    |---------->|   Command or    |
         |   +-----------------+   Send    |     <hello>     |
         | Send x5xx             Greeting  +-----------------+
         | Response  +-----------------+  Send    ^  |
         +-----------|     Prepare     | Response |  | Command
                     |     Response    |----------+  | Received
                     +-----------------+             V
                                ^          +-----------------+
                        Command |          |   Processing    |
                      Processed +----------|     Command     |
                                           +-----------------+
```

Figure 1: EPP Server State Machine

EPP commands fall into three categories: session management commands, query commands, and object transform commands. Session management commands are used to establish and end persistent sessions with an EPP server. Query commands are used to perform read-only object information retrieval operations. Transform commands are used to perform read-write object management operations.

Commands are processed by a server in the order they are received from a client. Though an immediate response confirming receipt and processing of the command is produced by the server, the protocol includes features that allow for offline review of transform commands before the requested action is actually completed. In such situations, the response from the server MUST clearly note that the command has been received and processed but that the requested action is pending. The state of the corresponding object MUST clearly reflect processing of the pending action. The server MUST also notify the client when offline processing of the action has been completed. Object mappings SHOULD describe standard formats for notices that describe completion of offline processing.

EPP uses XML namespaces to provide an extensible object management framework and to identify schemas required for XML instance parsing and validation. These namespaces and schema definitions are used to identify both the base protocol schema and the schemas for managed objects. The XML namespace prefixes used in examples (such as the string "foo" in "xmlns:foo") are solely for illustrative purposes. A conforming implementation MUST NOT require the use of these or any other specific namespace prefixes.

All XML instances SHOULD begin with an <?xml?> declaration to identify the version of XML that is being used, optionally identify use of the character encoding used, and optionally provide a hint to an XML parser that an external schema file is needed to validate the XML instance. Conformant XML parsers recognize both UTF-8 (defined in RFC 3629 [RFC3629]) and UTF-16 (defined in RFC 2781 [RFC2781]); per RFC 2277 [RFC2277], UTF-8 is the RECOMMENDED character encoding for use with EPP.

Character encodings other than UTF-8 and UTF-16 are allowed by XML. UTF-8 is the default encoding assumed by XML in the absence of an "encoding" attribute or a byte order mark (BOM); thus, the "encoding" attribute in the XML declaration is OPTIONAL if UTF-8 encoding is used. EPP clients and servers MUST accept a UTF-8 BOM if present, though emitting a UTF-8 BOM is NOT RECOMMENDED.

Example XML declarations:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<?xml version="1.0" standalone="no"?>

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

<?xml version="1.0"?>
```

2.1. Transport Mapping Considerations

As described previously, EPP can be layered over multiple transport protocols. There are, however, a common set of considerations that MUST be addressed by any transport mapping defined for EPP. These include:

- The transport mapping MUST preserve command order.

- The transport mapping MUST address the relationship between sessions and the client-server connection concept.

- The transport mapping MUST preserve the stateful nature of the protocol.

- The transport mapping MUST frame data units.

- The transport mapping MUST be onto a transport, such as TCP [RFC0793] or Stream Control Transmission Protocol (SCTP) [RFC4960], that provides congestion avoidance that follows RFC 2914 [RFC2914]; or, if it maps onto a protocol such as SMTP [RFC5321] or Blocks Extensible Exchange Protocol (BEEP) [RFC3080], then the performance issues need to take into account issues of overload, server availability, and so forth.

- The transport mapping MUST ensure reliability.

- The transport mapping MUST explicitly allow or prohibit pipelining.

Pipelining, also known as command streaming, is when a client sends multiple commands to a server without waiting for each corresponding response. After sending the commands, the client waits for the responses to arrive in the order corresponding to the completed commands. Performance gains can sometimes be realized with pipelining, especially with high-latency transports, but there are additional considerations associated with defining a transport mapping that supports pipelining:

- Commands MUST be processed independent of each other.

- Depending on the transport, pipelining MAY be possible in the form of sending a complete session in a well-defined "batch".

- The transport mapping MUST describe how an error in processing a command affects continued operation of the session.

A transport mapping MUST explain how all of these requirements are met, given the transport protocol being used to exchange data.

2.2. Protocol Identification

All EPP XML instances MUST begin with an <epp> element. This element identifies the start of an EPP protocol element and the namespace used within the protocol. The <epp> start element and the associated </epp> ending element MUST be applied to all structures sent by both clients and servers.

Example "start" and "end" EPP elements:

```
<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
</epp>
```

2.3. Hello Format

EPP MAY be carried over both connection-oriented and connection-less transport protocols. An EPP client MAY request a <greeting> from an EPP server at any time between a successful <login> command and a <logout> command by sending a <hello> to a server. Use of this element is essential in a connection-less environment where a server cannot return a <greeting> in response to a client-initiated connection. An EPP <hello> MUST be an empty element with no child elements.

Example <hello>:

```
C:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
C:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
C:  <hello/>
C:</epp>
```

2.4. Greeting Format

An EPP server responds to a successful connection and <hello> element by returning a <greeting> element to the client. An EPP greeting contains the following elements:

- An <svID> element that contains the name of the server.

- An <svDate> element that contains the server's current date and time in Universal Coordinated Time (UTC).

- An <svcMenu> element that identifies the services supported by the server, including:

- One or more <version> elements that identify the protocol versions supported by the server.

- One or more <lang> elements that contain the identifiers of the text response languages known by the server. Language identifiers MUST be structured as documented in [RFC4646].

- One or more <objURI> elements that contain namespace URIs representing the objects that the server is capable of managing. A server MAY limit object management privileges on a per-client basis.

- An OPTIONAL <svcExtension> element that contains one or more <extURI> elements that contain namespace URIs representing object extensions supported by the server.

- A <dcp> (data collection policy) element that contains child elements used to describe the server's privacy policy for data collection and management. Policy implications usually extend beyond the client-server relationship. Both clients and servers can have relationships with other entities that need to know the server operator's data collection policy to make informed provisioning decisions. Policy information MUST be disclosed to provisioning entities, though the method of disclosing policy data outside of direct protocol interaction is beyond the scope of this specification. Child elements include the following:

  - An <access> element that describes the access provided by the server to the client on behalf of the originating data source. The <access> element MUST contain one of the following child elements:

    - <all/>: Access is given to all identified data.

    - <none/>: No access is provided to identified data.

    - <null/>: Data is not persistent, so no access is possible.

    - <personal/>: Access is given to identified data relating to individuals and organizational entities.

    - <personalAndOther/>: Access is given to identified data relating to individuals, organizational entities, and other data of a non-personal nature.

  + <other/>: Access is given to other identified data of a non-personal nature.

* One or more <statement> elements that describe data collection purposes, data recipients, and data retention. Each <statement> element MUST contain a <purpose> element, a <recipient> element, and a <retention> element. The <purpose> element MUST contain one or more of the following child elements that describe the purposes for which data is collected:

  + <admin/>: Administrative purposes. Information can be used for administrative and technical support of the provisioning system.

  + <contact/>: Contact for marketing purposes. Information can be used to contact individuals, through a communications channel other than the protocol, for the promotion of a product or service.

  + <prov/>: Object-provisioning purposes. Information can be used to identify objects and inter-object relationships.

  + <other/>: Other purposes. Information may be used in other ways not captured by the above definitions.

* The <recipient> element MUST contain one or more of the following child elements that describes the recipients of collected data:

  + <other/>: Other entities following unknown practices.

  + <ours>: Server operator and/or entities acting as agents or entities for whom the server operator is acting as an agent. An agent in this instance is defined as a third party that processes data only on behalf of the service provider for the completion of the stated purposes. The <ours> element contains an OPTIONAL <recDesc> element that can be used to describe the recipient.

  + <public/>: Public forums.

  + <same/>: Other entities following server practices.

  + <unrelated/>: Unrelated third parties.

* The <retention> element MUST contain one of the following child elements that describes data retention practices:

  + <business/>: Data persists per business practices.

  + <indefinite/>: Data persists indefinitely.

  + <legal/>: Data persists per legal requirements.

  + <none/>: Data is not persistent and is not retained for more than a brief period of time necessary to make use of it during the course of a single online interaction.

  + <stated/>: Data persists to meet the stated purpose.

* An OPTIONAL <expiry> element that describes the lifetime of the policy. The <expiry> element MUST contain one of the following child elements:

  + <absolute/>: The policy is valid from the current date and time until it expires on the specified date and time.

  + <relative/>: The policy is valid from the current date and time until the end of the specified duration.

Data collection policy elements are based on work described in the World Wide Web Consortium’s Platform for Privacy Preferences [W3C.REC-P3P-20020416] specification.

Example greeting:

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

```
   S:      <access><all/></access>
   S:      <statement>
   S:        <purpose><admin/><prov/></purpose>
   S:        <recipient><ours/><public/></recipient>
   S:        <retention><stated/></retention>
   S:      </statement>
   S:    </dcp>
   S:  </greeting>
   S:</epp>
```

2.5. Command Format

An EPP client interacts with an EPP server by sending a command to the server and receiving a response from the server. In addition to the standard EPP elements, an EPP command contains the following elements:

- A command element whose tag corresponds to one of the valid EPP commands described in this document. The command element MAY contain either protocol-specified or object-specified child elements.

- An OPTIONAL <extension> element that MAY be used for server-defined command extensions.

- An OPTIONAL <clTRID> (client transaction identifier) element that MAY be used to uniquely identify the command to the client. Clients are responsible for maintaining their own transaction identifier space to ensure uniqueness.

Example command with object-specified child elements:

```
   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:      <obj:info xmlns:obj="urn:ietf:params:xml:ns:obj">
   C:        <obj:name>example</obj:name>
   C:      </obj:info>
   C:    </info>
   C:    <clTRID>ABC-12345</clTRID>
   C:  </command>
   C:</epp>
```

2.6. Response Format

An EPP server responds to a client command by returning a response to the client. EPP commands are atomic, so a command will either succeed completely or fail completely. Success and failure results MUST NOT be mixed. In addition to the standard EPP elements, an EPP response contains the following elements:

- One or more <result> elements that document the success or failure of command execution. If the command was processed successfully, only one <result> element MUST be returned. If the command was not processed successfully, multiple <result> elements MAY be returned to document failure conditions. Each <result> element contains the following attribute and child elements:

  - A "code" attribute whose value is a four-digit, decimal number that describes the success or failure of the command.

  - A <msg> element containing a human-readable description of the response code. The language of the response is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value MUST be "en" (English).

  - Zero or more OPTIONAL <value> elements that identify a client-provided element (including XML tag and value) or other information that caused a server error condition.

  - Zero or more OPTIONAL <extValue> elements that can be used to provide additional error diagnostic information, including:

    * A <value> element that identifies a client-provided element (including XML tag and value) that caused a server error condition.

    * A <reason> element containing a human-readable message that describes the reason for the error. The language of the response is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value MUST be "en" (English).

- An OPTIONAL <msgQ> element that describes messages queued for client retrieval. A <msgQ> element MUST NOT be present if there are no messages queued for client retrieval. A <msgQ> element MAY be present in responses to EPP commands other than the <poll> command if messages are queued for retrieval. A <msgQ> element MUST be present in responses to the EPP <poll> command if messages are queued for retrieval. The <msgQ> element contains the following attributes:

   o A "count" attribute that describes the number of messages that exist in the queue.

   o An "id" attribute used to uniquely identify the message at the head of the queue.

   The <msgQ> element contains the following OPTIONAL child elements that MUST be returned in response to a <poll> request command and MUST NOT be returned in response to any other command, including a <poll> acknowledgement:

   o A <qDate> element that contains the date and time that the message was enqueued.

   o A <msg> element containing a human-readable message. The language of the response is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value MUST be "en" (English). This element MAY contain XML content for formatting purposes, but the XML content is not specified by the protocol and will thus not be processed for validity.

- An OPTIONAL <resData> (response data) element that contains child elements specific to the command and associated object.

- An OPTIONAL <extension> element that MAY be used for server-defined response extensions.

- A <trID> (transaction identifier) element containing the transaction identifier assigned by the server to the command for which the response is being returned. The transaction identifier is formed using the <clTRID> associated with the command if supplied by the client and a <svTRID> (server transaction identifier) that is assigned by and unique to the server.

Transaction identifiers provide command-response synchronization integrity. They SHOULD be logged, retained, and protected to ensure that both the client and the server have consistent temporal and state-management records.

Example response without <value> or <resData>:

```
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">Command completed successfully</msg>
S:    </result>
S:    <trID>
```

```
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

Example response with <resData>:

```
   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:      <obj:creData xmlns:obj="urn:ietf:params:xml:ns:obj">
   S:        <obj:name>example</obj:name>
   S:      </obj:creData>
   S:    </resData>
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

Example response with error value elements:

```
   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="2004">
   S:      <msg>Parameter value range error</msg>
   S:      <value xmlns:obj="urn:ietf:params:xml:ns:obj">
   S:        <obj:elem1>2525</obj:elem1>
   S:      </value>
   S:    </result>
   S:    <result code="2005">
   S:      <msg>Parameter value syntax error</msg>
   S:      <value xmlns:obj="urn:ietf:params:xml:ns:obj">
   S:        <obj:elem2>ex(ample</obj:elem2>
   S:      </value>
   S:      <extValue>
   S:        <value xmlns:obj="urn:ietf:params:xml:ns:obj">
   S:          <obj:elem3>abc.ex(ample</obj:elem3>
   S:        </value>
   S:        <reason>Invalid character found.</reason>
   S:      </extValue>
   S:    </result>
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

Example response with notice of waiting server messages:

```
   S:<?xml version="1.0" encoding="UTF-8" standalone="no"?>
   S:<epp xmlns="urn:ietf:params:xml:ns:epp-1.0">
   S:  <response>
   S:    <result code="1000">
   S:      <msg>Command completed successfully</msg>
   S:    </result>
   S:    <msgQ count="5" id="12345"/>
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

Command success or failure MUST NOT be assumed if no response is returned or if a returned response is malformed. Protocol idempotency ensures the safety of retrying a command in cases of response-delivery failure.

2.7. Protocol Extension Framework

EPP provides an extension framework that allows features to be added at the protocol, object, and command-response levels.

2.7.1. Protocol Extension

The EPP extension framework allows for definition of new protocol elements identified using XML namespace notation with a reference to an XML schema that defines the namespace. The <epp> element that identifies the beginning of a protocol instance includes multiple child element choices, one of which is an <extension> element whose children define the extension. For example, a protocol extension element would be described in generic terms as follows:

```
   C:<epp>
   C:  <extension>
   C:    <!-- One or more extension elements. -->
   C:    <ext:foo xmlns:ext="urn:ietf:params:xml:ns:ext">
   C:      <!-- One or more extension child elements. -->
   C:    </ext:foo>
   C:  </extension>
   C:</epp>
```

This document does not define mappings for specific extensions. Extension specifications MUST be described in separate documents that define the objects and operations subject to the extension.

2.7.2. Object Extension

EPP provides an extensible object management framework that defines the syntax and semantics of protocol operations applied to a managed object. This framework pushes the definition of each protocol operation into the context of a specific object, providing the ability to add mappings for new objects without having to modify the base protocol.

Protocol elements that contain data specific to objects are identified using XML namespace notation with a reference to an XML schema that defines the namespace. The schema for EPP supports use of dynamic object schemas on a per-command and per-response basis. For example, the start of an object-specific command element would be described in generic terms as follows:

```
   C:<EPPCommandName>
   C:  <object:command xmlns:object="urn:ietf:params:xml:ns:object">
   C:    <!-- One or more object-specific command elements. -->
   C:  </object:command>
   C:</EPPCommandName>
```

An object-specific response element would be described similarly:

```
   S:<resData>
   S:  <object:resData xmlns:object="urn:ietf:params:xml:ns:object">
   S:    <!-- One or more object-specific response elements. -->
   S:  </object:resData>
   S:</resData>
```

This document does not define mappings for specific objects. The mapping of EPP to an object MUST be described in separate documents that specifically address each command and response in the context of the object. A suggested object mapping outline is included as an appendix to this document.

2.7.3. Command-Response Extension

EPP provides a facility for protocol command and response extensions. Protocol commands and responses MAY be extended by an <extension> element that contains additional elements whose syntax and semantics are not explicitly defined by EPP or an EPP object mapping. This element is OPTIONAL. Extensions are typically defined by agreement between client and server and MAY be used to extend EPP for unique operational needs. A server-extended command element would be described in generic terms as follows:

```
C:<command>
C:  <!-- EPPCommandName can be "create", "update", etc. -->
C:  <EPPCommandName>
C:    <object:command xmlns:object="urn:ietf:params:xml:ns:object">
C:      <!-- One or more object-specific command elements. -->
C:    </object:command>
C:  </EPPCommandName>
C:  <extension>
C:    <!-- One or more server-defined elements. -->
C:  </extension>
C:</command>
```

A server-extended response element would be described similarly:

```
S:<response>
S:  <result code="1000">
S:    <msg lang="en">Command completed successfully</msg>
S:  </result>
S:  <extension>
S:    <!-- One or more server-defined elements. -->
S:  </extension>
S:  <trID>
S:    <clTRID>ABC-12345</clTRID>
S:    <svTRID>54321-XYZ</svTRID>
S:  </trID>
S:</response>
```

This document does not define any specific server extensions. The mapping of server extensions to EPP MUST be described in separate documents that specifically address extended commands and responses in the server’s operational context.

2.8. Object Identification

Some objects, such as name servers and contacts, can have utility in multiple repositories. However, maintaining disjoint copies of object information in multiple repositories can lead to inconsistencies that have adverse consequences for the Internet. For example, changing the name of a name server in one repository but not in a second repository that refers to the server for domain name delegation can produce unexpected DNS query results.

Globally unique identifiers can help facilitate object-information sharing between repositories. A globally unique identifier MUST be assigned to every object when the object is created; the identifier MUST be returned to the client as part of any request to retrieve the detailed attributes of an object. Specific identifier values are a matter of repository policy, but they SHOULD be constructed according to the following algorithm:

a. Divide the provisioning repository world into a number of object repository classes.

b. Each repository within a class is assigned an identifier that is maintained by IANA.

c. Each repository is responsible for assigning a unique local identifier for each object within the repository.

d. The globally unique identifier is a concatenation of the local identifier, followed by a hyphen ("-", ASCII value 0x002D), followed by the repository identifier.

2.9. Protocol Commands

EPP provides commands to manage sessions, retrieve object information, and perform transformation operations on objects. All EPP commands are atomic and designed so that they can be made idempotent, either succeeding completely or failing completely and producing predictable results in case of repeated executions. This section describes each EPP command, including examples with representative server responses.

2.9.1. Session Management Commands

EPP provides two commands for session management: <login> to establish a session with a server and <logout> to end a session with a server. The <login> command establishes an ongoing server session that preserves client identity and authorization information during the duration of the session.

2.9.1.1. EPP <login> Command

The EPP <login> command is used to establish a session with an EPP server in response to a greeting issued by the server. A <login> command MUST be sent to a server before any other EPP command to establish an ongoing session. A server operator MAY limit the number of failed login attempts N, 1 <= N <= infinity, after which a login failure results in the connection to the server (if a connection exists) being closed.

A client identifier and initial password MUST be created on the server before a client can successfully complete a <login> command. The client identifier and initial password MUST be delivered to the client using an out-of-band method that protects the identifier and password from inadvertent disclosure.

In addition to the standard EPP command elements, the <login> command contains the following child elements:

- A <clID> element that contains the client identifier assigned to the client by the server.

- A <pw> element that contains the client’s plain text password. The value of this element is case sensitive.

- An OPTIONAL <newPW> element that contains a new plain text password to be assigned to the client for use with subsequent <login> commands. The value of this element is case sensitive.

- An <options> element that contains the following child elements:

  - A <version> element that contains the protocol version to be used for the command or ongoing server session.

  - A <lang> element that contains the text response language to be used for the command or ongoing server session commands.

  The values of the <version> and <lang> elements MUST exactly match one of the values presented in the EPP greeting.

- A <svcs> element that contains one or more <objURI> elements that contain namespace URIs representing the objects to be managed during the session. The <svcs> element MAY contain an OPTIONAL <svcExtension> element that contains one or more <extURI> elements that identify object extensions to be used during the session.

The PLAIN Simple Authentication and Security Layer (SASL) mechanism presented in [RFC4616] describes a format for providing a user identifier, an authorization identifier, and a password as part of a single plain-text string. The EPP authentication mechanism is similar, though EPP does not require a session-level authorization identifier and the user identifier and password are separated into distinct XML elements. Additional identification and authorization schemes MUST be provided at other protocol layers to provide more robust security services.

Example <login> 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:    <login>
C:      <clID>ClientX</clID>
C:      <pw>foo-BAR2</pw>
C:      <newPW>bar-FOO2</newPW>
C:      <options>
C:        <version>1.0</version>
C:        <lang>en</lang>
C:      </options>
C:      <svcs>
C:        <objURI>urn:ietf:params:xml:ns:obj1</objURI>
C:        <objURI>urn:ietf:params:xml:ns:obj2</objURI>
C:        <objURI>urn:ietf:params:xml:ns:obj3</objURI>
C:        <svcExtension>
C:          <extURI>http://custom/obj1ext-1.0</extURI>
C:        </svcExtension>
C:      </svcs>
C:    </login>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <login> command has been processed successfully, a server MUST respond with an EPP response with no <resData> element. If successful, the server will respond by creating and maintaining a new session that SHOULD be terminated by a future <logout> command.

Example <login> 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>
```

The EPP <login> command is used to establish a session with an EPP server. A <login> command MUST be rejected if received within the bounds of an existing session. This command MUST be available to all clients.

2.9.1.2. EPP <logout> Command

The EPP <logout> command is used to end a session with an EPP server. The <logout> command MUST be represented as an empty element with no child elements.

A server MAY end a session due to client inactivity or excessive client-session longevity. The parameters for determining excessive client inactivity or session longevity are a matter of server policy and are not specified by this protocol.

Transport mappings MUST explicitly describe any connection-oriented processing that takes place after processing a <logout> command and ending a session.

Example <logout> 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:    <logout/>
   C:    <clTRID>ABC-12345</clTRID>
   C:  </command>
   C:</epp>
```

When a <logout> command has been processed successfully, a server MUST respond with an EPP response with no <resData> element. If successful, the server MUST also end the current session.

Example <logout> 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="1500">
   S:      <msg>Command completed successfully; ending session</msg>
   S:    </result>
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

The EPP <logout> command is used to end a session with an EPP server. A <logout> command MUST be rejected if the command has not been preceded by a successful <login> command. This command MUST be available to all clients.

2.9.2. Query Commands

2.9.2.1. 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.

The elements needed to identify an object are object-specific, so the child elements of the <check> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <check> command contains the following child elements:

- An object-specific <obj:check> element that identifies the objects to be queried. Multiple objects of the same type MAY be queried within a single <check> command.

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:      <obj:check xmlns:obj="urn:ietf:params:xml:ns:obj">
   C:        <obj:name>example1</obj:name>
   C:        <obj:name>example2</obj:name>
   C:        <obj:name>example3</obj:name>
   C:      </obj:check>
   C:    </check>
   C:    <clTRID>ABC-12346</clTRID>
   C:  </command>
   C:</epp>
```

When a <check> command has been processed successfully, a server MUST respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific, though the EPP <resData> element MUST contain a child <obj:chkData> element that contains one or more <obj:cd> (check data) elements. Each <obj:cd> element contains the following child elements:

- An object-specific 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.

- An OPTIONAL <obj: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:      <obj:chkData xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <obj:cd>
S:          <obj:name avail="1">example1</obj:name>
S:        </obj:cd>
S:        <obj:cd>
S:          <obj:name avail="0">example2</obj:name>
S:          <obj:reason>In use</obj:reason>
S:        </obj:cd>
S:        <obj:cd>
S:          <obj:name avail="1">example3</obj:name>
S:        </obj:cd>
S:      </obj:chkData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12346</clTRID>
```

```
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <check> command is used to determine if an object can be provisioned within a repository. This action MUST be open to all authorized clients.

2.9.2.2. EPP <info> Command

The EPP <info> command is used to retrieve information associated with an existing object. The elements needed to identify an object and the type of information associated with an object are both object-specific, so the child elements of the <info> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <info> command contains the following child elements:

- An object-specific <obj:info> element that identifies the 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:      <obj:info xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:info>
C:    </info>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

When an <info> command has been processed successfully, a server MUST respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace and the Repository Object IDentifier (ROID) that was assigned to the object when the object was created. Other child elements of the <resData> element are object-specific.

Example <info> 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:      <obj:infData xmlns:obj="urn:ietf:params:xml:ns:obj">
   S:        <obj:roid>EXAMPLE1-REP</obj:roid>
   S:        <!-- Object-specific elements. -->
   S:      </obj:infData>
   S:    </resData>
   S:    <trID>
   S:      <clTRID>ABC-12346</clTRID>
   S:      <svTRID>54322-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

The EPP <info> command is used to retrieve information associated with an existing object. This action SHOULD be limited to authorized clients; restricting this action to the sponsoring client is RECOMMENDED.

2.9.2.3. EPP <poll> Command

The EPP <poll> command is used to discover and retrieve service messages queued by a server for individual clients. If the message queue is not empty, a successful response to a <poll> command MUST return the first message from the message queue. Each response returned from the server includes a server-unique message identifier that MUST be provided to acknowledge receipt of the message, and a counter that indicates the number of messages in the queue. After a message has been received by the client, the client MUST respond to the message with an explicit acknowledgement to confirm that the message has been received. A server MUST dequeue the message and decrement the queue counter after receiving acknowledgement from the client, making the next message in the queue (if any) available for retrieval.

Servers can occasionally perform actions on objects that are not in direct response to a client request, or an action taken by one client can indirectly involve a second client. Examples of such actions include deletion upon expiration, automatic renewal upon expiration, and transfer coordination; other types of service information MAY be defined as a matter of server policy. Service messages SHOULD be created for passive clients affected by an action on an object. Service messages MAY also be created for active clients that request an action on an object, though such messages MUST NOT replace the normal protocol response to the request. For example, <transfer> actions SHOULD be reported to the client that has the authority to approve or reject a transfer request. Other methods of server-client action notification, such as offline reporting, are also possible and are beyond the scope of this specification.

Message queues can consume server resources if clients do not retrieve and acknowledge messages on a regular basis. Servers MAY implement other mechanisms to dequeue and deliver messages if queue maintenance needs exceed server resource consumption limits. Server operators SHOULD consider time-sensitivity and resource management factors when selecting a delivery method for service information because some message types can be reasonably delivered using non-protocol methods that require fewer server resources.

Some of the information returned in response to a <poll> command can be object-specific, so some child elements of the <poll> response MAY be specified using the EPP extension framework. The <poll> command MUST be represented as an empty element with no child elements. An "op" attribute with value "req" is REQUIRED to retrieve the first message from the server message queue. An "op" attribute (with value "ack") and a "msgID" attribute (whose value corresponds to the value of the "id" attribute copied from the <msg> element in the message being acknowledged) are REQUIRED to acknowledge receipt of a message.

Example <poll> 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:    <poll op="req"/>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

The returned result code notes that a message has been dequeued and returned in response to a <poll> command.

Example <poll> response with object-specific 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="1301">
S:      <msg>Command completed successfully; ack to dequeue</msg>
S:    </result>
S:    <msgQ count="5" id="12345">
S:      <qDate>2000-06-08T22:00:00.0Z</qDate>
S:      <msg>Transfer requested.</msg>
S:    </msgQ>
```

```
   S:    <resData>
   S:      <obj:trnData
   S:       xmlns:obj="urn:ietf:params:xml:ns:obj-1.0">
   S:        <obj:name>example.com</obj:name>
   S:        <obj:trStatus>pending</obj:trStatus>
   S:        <obj:reID>ClientX</obj:reID>
   S:        <obj:reDate>2000-06-08T22:00:00.0Z</obj:reDate>
   S:        <obj:acID>ClientY</obj:acID>
   S:        <obj:acDate>2000-06-13T22:00:00.0Z</obj:acDate>
   S:        <obj:exDate>2002-09-08T22:00:00.0Z</obj:exDate>
   S:      </obj:trnData>
   S:    </resData>
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

A client MUST acknowledge each response to dequeue the message and make subsequent messages available for retrieval.

Example <poll> acknowledgement 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:    <poll op="ack" msgID="12345"/>
   C:    <clTRID>ABC-12346</clTRID>
   C:  </command>
   C:</epp>
```

A <poll> acknowledgement response notes the ID of the message that has been acknowledged and the number of messages remaining in the queue.

Example <poll> acknowledgement 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:    <msgQ count="4" id="12345"/>
   S:    <trID>
   S:      <clTRID>ABC-12346</clTRID>
   S:      <svTRID>54322-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

Service messages can also be returned without object information.

Example <poll> response with mixed message content and without object-specific 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="1301">
   S:      <msg>Command completed successfully; ack to dequeue</msg>
   S:    </result>
   S:    <msgQ count="4" id="12346">
   S:      <qDate>2000-06-08T22:10:00.0Z</qDate>
   S:      <msg lang="en">Credit balance low.
   S:        <limit>100</limit><bal>5</bal>
   S:      </msg>
   S:    </msgQ>
   S:    <trID>
   S:      <clTRID>ABC-12346</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

The returned result code and message is used to note an empty server message queue.

Example <poll> response to note an empty message queue:

```
      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="1300">
      S:      <msg>Command completed successfully; no messages</msg>
      S:    </result>
      S:    <trID>
      S:      <clTRID>ABC-12346</clTRID>
      S:      <svTRID>54321-XYZ</svTRID>
      S:    </trID>
      S:  </response>
      S:</epp>
```

The EPP <poll> command is used to discover and retrieve client service messages from a server. This action SHOULD be limited to authorized clients; queuing service messages and limiting queue access on a per-client basis is RECOMMENDED.

2.9.2.4. EPP <transfer> Query Command

The EPP <transfer> command provides a query operation that allows a client to determine real-time status of pending and completed transfer requests. The elements needed to identify an object that is the subject of a transfer request are object-specific, so the child elements of the <transfer> query command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <transfer> command contains an "op" attribute with value "query" and the following child elements:

- An object-specific <obj:transfer> element that identifies the object whose transfer status is requested.

Transfer status is typically considered sensitive information by the clients involved in the operation. Object mappings MUST provide features to restrict transfer queries to authorized clients, such as by requiring authorization information as part of the request.

Example <transfer> query 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="query">
C:      <obj:transfer xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:transfer>
C:    </transfer>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

When a <transfer> query command has been processed successfully, a server MUST respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific, but they MUST include elements that identify the object, the status of the transfer, the identifier of the client that requested the transfer, the date and time that the request was made, the identifier of the client that is authorized to act on the request, the date and time by which an action is expected, and an OPTIONAL date and time noting changes in the object’s validity period (if applicable) that occur as a result of 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="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <obj:trnData xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <obj:name>example</obj:name>
S:        <obj:trStatus>pending</obj:trStatus>
S:        <obj:reID>ClientX</obj:reID>
S:        <obj:reDate>2000-06-08T22:00:00.0Z</obj:reDate>
S:        <obj:acID>ClientY</obj:acID>
S:        <obj:acDate>2000-06-13T22:00:00.0Z</obj:acDate>
S:        <obj:exDate>2002-09-08T22:00:00.0Z</obj:exDate>
S:      </obj:trnData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12346</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <transfer> command provides a query operation that allows a client to determine real-time status of pending and completed transfer requests. This action SHOULD be limited to authorized clients; restricting queries to the requesting and responding clients is RECOMMENDED. Object transfer MAY be unavailable or limited by object-specific policies.

2.9.3. Object Transform Commands

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

2.9.3.1. EPP <create> Command

The EPP <create> command is used to create an instance of an object. An object can be created for an indefinite period of time, or an object can be created for a specific validity period. The EPP mapping for an object MUST describe the status of an object with respect to time in order to include expected client and server behavior if a validity period is used.

The elements needed to identify an object and associated attributes are object-specific, so the child elements of the <create> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <create> command contains the following child elements:

- An object-specific <obj:create> element that identifies the object to be created and the elements that are required to create the object.

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:      <obj:create xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:create>
C:    </create>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <create> command has been processed successfully, a server MAY respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific.

Example <create> response with <resData>:

```
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:      <obj:creData xmlns:obj="urn:ietf:params:xml:ns:obj">
```

```
S:        <!-- Object-specific elements. -->
S:      </obj:creData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54321-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <create> command is used to create an instance of an object. This action SHOULD be limited to authorized clients and MAY be restricted on a per-client basis.

2.9.3.2. EPP <delete> Command

The EPP <delete> command is used to remove an instance of an existing object. The elements needed to identify an object are object-specific, so the child elements of the <delete> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <delete> command contains the following child elements:

- An object-specific <obj:delete> element that identifies the object to be deleted.

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:      <obj:delete xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:delete>
C:    </delete>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

When a <delete> command has been processed successfully, a server MAY respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific.

Example <delete> response without <resData>:

```
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-12346</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <delete> command is used to remove an instance of an existing object. This action SHOULD be limited to authorized clients; restricting this action to the sponsoring client is RECOMMENDED.

2.9.3.3. EPP <renew> Command

The EPP <renew> command is used to extend the validity period of an existing object. The elements needed to identify and extend the validity period of an object are object-specific, so the child elements of the <renew> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <renew> command contains the following child elements:

- An object-specific <obj:renew> element that identifies the object to be renewed and the elements that are required to extend the validity period of the object.

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:      <obj:renew xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:renew>
C:    </renew>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

When a <renew> command has been processed successfully, a server MAY respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific.

Example <renew> response with <resData>:

```
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:      <obj:renData xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <!-- Object-specific elements. -->
S:      </obj:renData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12346</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <renew> command is used to extend the validity period of an existing object. This action SHOULD be limited to authorized clients; restricting this action to the sponsoring client is RECOMMENDED. Object renewal MAY be unavailable or limited by object-specific policies.

2.9.3.4. EPP <transfer> Command

The EPP <transfer> command is used to manage changes in client sponsorship of an existing object. Clients can initiate a transfer request, cancel a transfer request, approve a transfer request, and reject a transfer request using the "op" command attribute.

A client who wishes to assume sponsorship of a known object from another client uses the <transfer> command with the value of the "op" attribute set to "request". Once a transfer has been requested, the same client can cancel the request using a <transfer> command with the value of the "op" attribute set to "cancel". A request to cancel the transfer MUST be sent to the server before the current sponsoring client either approves or rejects the transfer request and before the server automatically processes the request due to responding client inactivity.

Once a transfer request has been received by the server, the server MUST notify the current sponsoring client of the requested transfer either by queuing a service message for retrieval via the <poll> command or by using an out-of-band mechanism to inform the client of the request. The current status of a pending <transfer> command for any object can be found using the <transfer> query command. Transfer service messages MUST include the object-specific elements specified for <transfer> command responses.

The current sponsoring client MAY explicitly approve or reject the transfer request. The client can approve the request using a <transfer> command with the value of the "op" attribute set to "approve". The client can reject the request using a <transfer> command with the value of the "op" attribute set to "reject".

A server MAY automatically approve or reject all transfer requests that are not explicitly approved or rejected by the current sponsoring client within a fixed amount of time. The amount of time to wait for explicit action and the default server behavior are local matters not specified by EPP, but they SHOULD be documented in a server-specific profile document that describes default server behavior for client information.

Objects eligible for transfer MUST have associated authorization information that MUST be provided to complete a <transfer> command. The type of authorization information required is object-specific; passwords or more complex mechanisms based on public key cryptography are typical.

The elements needed to identify and complete the transfer of an object are object-specific, so the child elements of the <transfer> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <transfer> command contains the following child elements:

- An object-specific <obj:transfer> element that identifies the object to be transferred and the elements that are required to process the transfer command.

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:      <obj:transfer xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:transfer>
```

```
C:    </transfer>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

When a <transfer> command has been processed successfully, a server MUST respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific, but they MUST include elements that identify the object, the status of the transfer, the identifier of the client that requested the transfer, the date and time that the request was made, the identifier of the client that is authorized to act on the request, the date and time by which an action is expected, and an OPTIONAL date and time noting changes in the object’s validity period (if applicable) that occur as a result of the transfer.

Example <transfer> response with <resData>:

```
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:      <obj:trnData xmlns:obj="urn:ietf:params:xml:ns:obj">
S:        <obj:name>example</obj:name>
S:        <obj:trStatus>pending</obj:trStatus>
S:        <obj:reID>ClientX</obj:reID>
S:        <obj:reDate>2000-06-08T22:00:00.0Z</obj:reDate>
S:        <obj:acID>ClientY</obj:acID>
S:        <obj:acDate>2000-06-13T22:00:00.0Z</obj:acDate>
S:        <obj:exDate>2002-09-08T22:00:00.0Z</obj:exDate>
S:      </obj:trnData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12346</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <transfer> command is used to manage changes in client sponsorship of an existing object. This action SHOULD be limited to authorized clients; restricting <transfer> requests to a client other than the current sponsoring client, <transfer> approval requests to the current sponsoring client, and <transfer> cancellation requests to the original requesting client is RECOMMENDED. Object transfer MAY be unavailable or limited by object-specific policies.

2.9.3.5. EPP <update> Command

The EPP <update> command is used to change information associated with an existing object. The elements needed to identify and modify an object are object-specific, so the child elements of the <update> command are specified using the EPP extension framework. In addition to the standard EPP command elements, the <update> command contains the following child elements:

- An object-specific <obj:update> element that identifies the object to be updated and the elements that are required to modify the object. Object-specific elements MUST identify values to be added, values to be removed, or values to be changed.

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:      <obj:update xmlns:obj="urn:ietf:params:xml:ns:obj">
C:        <!-- Object-specific elements. -->
C:      </obj:update>
C:    </update>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

When an <update> command has been processed successfully, a server MAY respond with an EPP <resData> element that MUST contain a child element that identifies the object namespace. The child elements of the <resData> element are object-specific.

Example <update> response without <resData>:

```
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-12346</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
```

```
S:    </trID>
S:  </response>
S:</epp>
```

The EPP <update> command is used to change information associated with an existing object. This action SHOULD be limited to authorized clients; restricting this action to the sponsoring client is RECOMMENDED.

3. Result Codes

EPP result codes are based on the theory of reply codes described in section 4.2.1 of [RFC5321]. EPP uses four decimal digits to describe the success or failure of each EPP command. Each of the digits of the reply have special significance.

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

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

1yzz	Positive completion reply. The command was accepted and processed by the system without error.
2yzz	Negative completion reply. The command was not accepted, and the requested action did not occur.

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

x0zz	Protocol Syntax
x1zz	Implementation-specific Rules
x2zz	Security
x3zz	Data Management
x4zz	Server System
x5zz	Connection Management

The third and fourth digits provide response detail within the categories defined by the first and second digits. The complete list of valid result codes is enumerated below and in the normative schema.

Every EPP response MUST include a result code and a human-readable description of the result code. The language used to represent the description MAY be identified using an instance of the "lang" attribute within the <msg> element. If not specified, the default language is English, identified as "en". A description of the structure of valid values for the "lang" attribute is described in [RFC4646].

Response text MAY be translated into other languages, though the translation MUST preserve the meaning of the code as described here. Response code values MUST NOT be changed when translating text.

Response text in the table below is enclosed in quotes to clearly mark the beginning and ending of each response string. Quotes MUST NOT be used to delimit these strings when returning response text via the protocol.

Successful command completion responses:

Code	Response text in English
1000	"Command completed successfully" This is the usual response code for a successfully completed command that is not addressed by any other 1xxx-series response code.
1001	"Command completed successfully; action pending" This response code MUST be returned when responding to a command that requires offline activity before the requested action can be completed. See Section 2 for a description of other processing requirements.
1300	"Command completed successfully; no messages" This response code MUST be returned when responding to a <poll> request command and the server message queue is empty.
1301	"Command completed successfully; ack to dequeue" This response code MUST be returned when responding to a <poll> request command and a message has been retrieved from the server message queue.

1500	"Command completed successfully; ending session" This response code MUST be returned when responding to a successful <logout> command.

Command error responses:

Code	Response text in English
2000	"Unknown command" This response code MUST be returned when a server receives a command element that is not defined by EPP.
2001	"Command syntax error" This response code MUST be returned when a server receives an improperly formed command element.
2002	"Command use error" This response code MUST be returned when a server receives a properly formed command element but the command cannot be executed due to a sequencing or context error. For example, a <logout> command cannot be executed without having first completed a <login> command.
2003	"Required parameter missing" This response code MUST be returned when a server receives a command for which a required parameter value has not been provided.
2004	"Parameter value range error" This response code MUST be returned when a server receives a command parameter whose value is outside the range of values specified by the protocol. The error value SHOULD be returned via a <value> element in the EPP response.
2005	"Parameter value syntax error" This response code MUST be returned when a server receives a command containing a parameter whose value is improperly formed. The error value SHOULD be returned via a <value> element in the EPP response.

2100 "Unimplemented protocol version"

This response code MUST be returned when a server receives a command element specifying a protocol version that is not implemented by the server.

2101 "Unimplemented command"

This response code MUST be returned when a server receives a valid EPP command element that is not implemented by the server. For example, a <transfer> command can be unimplemented for certain object types.

2102 "Unimplemented option"

This response code MUST be returned when a server receives a valid EPP command element that contains a protocol option that is not implemented by the server.

2103 "Unimplemented extension"

This response code MUST be returned when a server receives a valid EPP command element that contains a protocol command extension that is not implemented by the server.

2104 "Billing failure"

This response code MUST be returned when a server attempts to execute a billable operation and the command cannot be completed due to a client-billing failure.

2105 "Object is not eligible for renewal"

This response code MUST be returned when a client attempts to <renew> an object that is not eligible for renewal in accordance with server policy.

2106 "Object is not eligible for transfer"

This response code MUST be returned when a client attempts to <transfer> an object that is not eligible for transfer in accordance with server policy.

2200 "Authentication error"

This response code MUST be returned when a server notes an error when validating client credentials.

2201 "Authorization error"

This response code MUST be returned when a server notes a client-authorization error when executing a command. This error is used to note that a client lacks privileges to execute the requested command.

2202 "Invalid authorization information"

This response code MUST be returned when a server receives invalid command authorization information when attempting to confirm authorization to execute a command. This error is used to note that a client has the privileges required to execute the requested command, but the authorization information provided by the client does not match the authorization information archived by the server.

2300 "Object pending transfer"

This response code MUST be returned when a server receives a command to transfer of an object that is pending transfer due to an earlier transfer request.

2301 "Object not pending transfer"

This response code MUST be returned when a server receives a command to confirm, reject, or cancel the transfer of an object when no command has been made to transfer the object.

2302 "Object exists"

This response code MUST be returned when a server receives a command to create an object that already exists in the repository.

2303 "Object does not exist"

This response code MUST be returned when a server receives a command to query or transform an object that does not exist in the repository.

2304 "Object status prohibits operation"

This response code MUST be returned when a server receives a command to transform an object that cannot be completed due to server policy or business practices. For example, a server can disallow <transfer> commands under terms and conditions that are matters of local policy, or the server might have received a <delete> command for an object whose status prohibits deletion.

2305 "Object association prohibits operation"

This response code MUST be returned when a server receives a command to transform an object that cannot be completed due to dependencies on other objects that are associated with the target object. For example, a server can disallow <delete> commands while an object has active associations with other objects.

2306 "Parameter value policy error"

This response code MUST be returned when a server receives a command containing a parameter value that is syntactically valid but semantically invalid due to local policy. For example, the server can support a subset of a range of valid protocol parameter values. The error value SHOULD be returned via a <value> element in the EPP response.

2307 "Unimplemented object service"

This response code MUST be returned when a server receives a command to operate on an object service that is not supported by the server.

2308 "Data management policy violation"

This response code MUST be returned when a server receives a command whose execution results in a violation of server data management policies. For example, removing all attribute values or object associations from an object might be a violation of a server's data management policies.

2400 "Command failed"

This response code MUST be returned when a server is unable to execute a command due to an internal server error that is not related to the protocol. The failure can be transient. The server MUST keep any ongoing session active.

2500 "Command failed; server closing connection"

This response code MUST be returned when a server receives a command that cannot be completed due to an internal server error that is not related to the protocol. The failure is not transient and will cause other commands to fail as well. The server MUST end the active session and close the existing connection.

2501 "Authentication error; server closing connection"

This response code MUST be returned when a server notes an error when validating client credentials and a server-defined limit on the number of allowable failures has been exceeded. The server MUST close the existing connection.

2502 "Session limit exceeded; server closing connection"

This response code MUST be returned when a server receives a <login> command and the command cannot be completed because the client has exceeded a system-defined limit on the number of sessions that the client can establish. It might be possible to establish a session by ending existing unused sessions and closing inactive connections.

4. Formal Syntax

EPP is specified in XML Schema notation. The formal syntax presented here is a complete schema representation of EPP suitable for automated validation of EPP XML instances.

Two schemas are presented here. The first schema is the base EPP schema. The second schema defines elements and structures that can be used by both the base EPP schema and object mapping schema. 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. Base Schema

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

<schema targetNamespace="urn:ietf:params:xml:ns:epp-1.0"
        xmlns:epp="urn:ietf:params:xml:ns:epp-1.0"
        xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
        xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

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

  <annotation>
    <documentation>
      Extensible Provisioning Protocol v1.0 schema.
    </documentation>
  </annotation>

<!--
Every EPP XML instance must begin with this element.
-->
```

```
     <element name="epp" type="epp:eppType"/>

   <!--
   An EPP XML instance must contain a greeting, hello, command,
   response, or extension.
   -->
     <complexType name="eppType">
       <choice>
         <element name="greeting" type="epp:greetingType"/>
         <element name="hello"/>
         <element name="command" type="epp:commandType"/>
         <element name="response" type="epp:responseType"/>
         <element name="extension" type="epp:extAnyType"/>
       </choice>
     </complexType>

   <!--
   A greeting is sent by a server in response to a client connection
   or <hello>.
   -->
     <complexType name="greetingType">
       <sequence>
         <element name="svID" type="epp:sIDType"/>
         <element name="svDate" type="dateTime"/>
         <element name="svcMenu" type="epp:svcMenuType"/>
         <element name="dcp" type="epp:dcpType"/>
       </sequence>
     </complexType>

   <!--
   Server IDs are strings with minimum and maximum length restrictions.
   -->
     <simpleType name="sIDType">
       <restriction base="normalizedString">
         <minLength value="3"/>
         <maxLength value="64"/>
       </restriction>
     </simpleType>

   <!--
   A server greeting identifies available object services.
   -->
     <complexType name="svcMenuType">
       <sequence>
         <element name="version" type="epp:versionType"
          maxOccurs="unbounded"/>
         <element name="lang" type="language"
          maxOccurs="unbounded"/>
         <element name="objURI" type="anyURI"
          maxOccurs="unbounded"/>
         <element name="svcExtension" type="epp:extURIType"
          minOccurs="0"/>
       </sequence>
     </complexType>

   <!--
   Data Collection Policy types.
   -->
     <complexType name="dcpType">
       <sequence>
         <element name="access" type="epp:dcpAccessType"/>
         <element name="statement" type="epp:dcpStatementType"
          maxOccurs="unbounded"/>
         <element name="expiry" type="epp:dcpExpiryType"
          minOccurs="0"/>
       </sequence>
     </complexType>

     <complexType name="dcpAccessType">
       <choice>
         <element name="all"/>
         <element name="none"/>
         <element name="null"/>
         <element name="other"/>
         <element name="personal"/>
         <element name="personalAndOther"/>
       </choice>
     </complexType>

     <complexType name="dcpStatementType">
       <sequence>
         <element name="purpose" type="epp:dcpPurposeType"/>
         <element name="recipient" type="epp:dcpRecipientType"/>
         <element name="retention" type="epp:dcpRetentionType"/>
       </sequence>
     </complexType>

     <complexType name="dcpPurposeType">
       <sequence>
         <element name="admin"
          minOccurs="0"/>
         <element name="contact"
          minOccurs="0"/>
         <element name="other"
          minOccurs="0"/>
         <element name="prov"
```

```
          minOccurs="0"/>
       </sequence>
     </complexType>

     <complexType name="dcpRecipientType">
       <sequence>
         <element name="other"
          minOccurs="0"/>
         <element name="ours" type="epp:dcpOursType"
          minOccurs="0" maxOccurs="unbounded"/>
         <element name="public"
          minOccurs="0"/>
         <element name="same"
          minOccurs="0"/>
         <element name="unrelated"
          minOccurs="0"/>
       </sequence>
     </complexType>

     <complexType name="dcpOursType">
       <sequence>
         <element name="recDesc" type="epp:dcpRecDescType"
          minOccurs="0"/>
       </sequence>
     </complexType>

     <simpleType name="dcpRecDescType">
       <restriction base="token">
         <minLength value="1"/>
         <maxLength value="255"/>
       </restriction>
     </simpleType>

     <complexType name="dcpRetentionType">
       <choice>
         <element name="business"/>
         <element name="indefinite"/>
         <element name="legal"/>
         <element name="none"/>
         <element name="stated"/>
       </choice>
     </complexType>

     <complexType name="dcpExpiryType">
       <choice>
         <element name="absolute" type="dateTime"/>
         <element name="relative" type="duration"/>
       </choice>
     </complexType>

   <!--
   Extension framework types.
   -->
     <complexType name="extAnyType">
       <sequence>
         <any namespace="##other"
          maxOccurs="unbounded"/>
       </sequence>
     </complexType>

     <complexType name="extURIType">
       <sequence>
         <element name="extURI" type="anyURI"
          maxOccurs="unbounded"/>
       </sequence>
     </complexType>

   <!--
   An EPP version number is a dotted pair of decimal numbers.
   -->
     <simpleType name="versionType">
       <restriction base="token">
         <pattern value="[1-9]+\.[0-9]+"/>
         <enumeration value="1.0"/>
       </restriction>
     </simpleType>

   <!--
   Command types.
   -->
     <complexType name="commandType">
       <sequence>
         <choice>
           <element name="check" type="epp:readWriteType"/>
           <element name="create" type="epp:readWriteType"/>
           <element name="delete" type="epp:readWriteType"/>
           <element name="info" type="epp:readWriteType"/>
           <element name="login" type="epp:loginType"/>
           <element name="logout"/>
           <element name="poll" type="epp:pollType"/>
           <element name="renew" type="epp:readWriteType"/>
           <element name="transfer" type="epp:transferType"/>
           <element name="update" type="epp:readWriteType"/>
         </choice>
         <element name="extension" type="epp:extAnyType"
          minOccurs="0"/>
         <element name="clTRID" type="epp:trIDStringType"
          minOccurs="0"/>
       </sequence>
     </complexType>

   <!--
   The <login> command.
   -->
     <complexType name="loginType">
       <sequence>
         <element name="clID" type="eppcom:clIDType"/>
         <element name="pw" type="epp:pwType"/>
         <element name="newPW" type="epp:pwType"
          minOccurs="0"/>
         <element name="options" type="epp:credsOptionsType"/>
         <element name="svcs" type="epp:loginSvcType"/>
       </sequence>
     </complexType>

     <complexType name="credsOptionsType">
       <sequence>
         <element name="version" type="epp:versionType"/>
         <element name="lang" type="language"/>
       </sequence>
     </complexType>

     <simpleType name="pwType">
       <restriction base="token">
         <minLength value="6"/>
         <maxLength value="16"/>
       </restriction>
     </simpleType>

     <complexType name="loginSvcType">
       <sequence>
         <element name="objURI" type="anyURI"
          maxOccurs="unbounded"/>
         <element name="svcExtension" type="epp:extURIType"
          minOccurs="0"/>
       </sequence>
     </complexType>

   <!--
   The <poll> command.
   -->
     <complexType name="pollType">
       <attribute name="op" type="epp:pollOpType"
        use="required"/>
       <attribute name="msgID" type="token"/>
     </complexType>

     <simpleType name="pollOpType">
       <restriction base="token">
         <enumeration value="ack"/>
         <enumeration value="req"/>
       </restriction>
     </simpleType>

   <!--
   The <transfer> command.  This is object-specific, and uses attributes
   to identify the requested operation.
   -->
     <complexType name="transferType">
       <sequence>
         <any namespace="##other"/>
       </sequence>
       <attribute name="op" type="epp:transferOpType"
        use="required"/>
     </complexType>

     <simpleType name="transferOpType">
       <restriction base="token">
         <enumeration value="approve"/>
         <enumeration value="cancel"/>
         <enumeration value="query"/>
         <enumeration value="reject"/>
         <enumeration value="request"/>
       </restriction>
     </simpleType>

   <!--
   All other object-centric commands.  EPP doesn’t specify the syntax or
   semantics of object-centric command elements.  The elements MUST be
   described in detail in another schema specific to the object.
   -->
     <complexType name="readWriteType">
       <sequence>
         <any namespace="##other"/>
       </sequence>
     </complexType>

     <complexType name="trIDType">
       <sequence>
         <element name="clTRID" type="epp:trIDStringType"
          minOccurs="0"/>
         <element name="svTRID" type="epp:trIDStringType"/>
       </sequence>
     </complexType>

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

   <!--
   Response types.
   -->
     <complexType name="responseType">
       <sequence>
         <element name="result" type="epp:resultType"
          maxOccurs="unbounded"/>
         <element name="msgQ" type="epp:msgQType"
          minOccurs="0"/>

         <element name="resData" type="epp:extAnyType"
          minOccurs="0"/>
         <element name="extension" type="epp:extAnyType"
          minOccurs="0"/>
         <element name="trID" type="epp:trIDType"/>
       </sequence>
     </complexType>

     <complexType name="resultType">
       <sequence>
         <element name="msg" type="epp:msgType"/>
         <choice minOccurs="0" maxOccurs="unbounded">
           <element name="value" type="epp:errValueType"/>
           <element name="extValue" type="epp:extErrValueType"/>
         </choice>
       </sequence>
       <attribute name="code" type="epp:resultCodeType"
        use="required"/>
     </complexType>

     <complexType name="errValueType" mixed="true">
       <sequence>
         <any namespace="##any" processContents="skip"/>
       </sequence>
       <anyAttribute namespace="##any" processContents="skip"/>
     </complexType>
```

```
     <complexType name="extErrValueType">
       <sequence>
         <element name="value" type="epp:errValueType"/>
         <element name="reason" type="epp:msgType"/>
       </sequence>
     </complexType>

     <complexType name="msgQType">
       <sequence>
         <element name="qDate" type="dateTime"
          minOccurs="0"/>
         <element name="msg" type="epp:mixedMsgType"
          minOccurs="0"/>
       </sequence>
       <attribute name="count" type="unsignedLong"
        use="required"/>
       <attribute name="id" type="eppcom:minTokenType"
        use="required"/>
     </complexType>

     <complexType name="mixedMsgType" mixed="true">
       <sequence>
         <any processContents="skip"
          minOccurs="0" maxOccurs="unbounded"/>
       </sequence>
       <attribute name="lang" type="language"
        default="en"/>
     </complexType>

   <!--
   Human-readable text may be expressed in languages other than English.
   -->
     <complexType name="msgType">
       <simpleContent>
         <extension base="normalizedString">
           <attribute name="lang" type="language"
            default="en"/>
         </extension>
       </simpleContent>
     </complexType>

   <!--
   EPP result codes.
   -->
     <simpleType name="resultCodeType">
       <restriction base="unsignedShort">
         <enumeration value="1000"/>
         <enumeration value="1001"/>
         <enumeration value="1300"/>
         <enumeration value="1301"/>
         <enumeration value="1500"/>
         <enumeration value="2000"/>
         <enumeration value="2001"/>
         <enumeration value="2002"/>
         <enumeration value="2003"/>
         <enumeration value="2004"/>
         <enumeration value="2005"/>
         <enumeration value="2100"/>
         <enumeration value="2101"/>
         <enumeration value="2102"/>
         <enumeration value="2103"/>
         <enumeration value="2104"/>
         <enumeration value="2105"/>
         <enumeration value="2106"/>
         <enumeration value="2200"/>
         <enumeration value="2201"/>
         <enumeration value="2202"/>
         <enumeration value="2300"/>
         <enumeration value="2301"/>
         <enumeration value="2302"/>
         <enumeration value="2303"/>
         <enumeration value="2304"/>
         <enumeration value="2305"/>
         <enumeration value="2306"/>
         <enumeration value="2307"/>
         <enumeration value="2308"/>
         <enumeration value="2400"/>
         <enumeration value="2500"/>
         <enumeration value="2501"/>
         <enumeration value="2502"/>
       </restriction>
     </simpleType>

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

4.2. Shared Structure Schema

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

<schema targetNamespace="urn:ietf:params:xml:ns:eppcom-1.0"
        xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
        xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

  <annotation>
    <documentation>
      Extensible Provisioning Protocol v1.0
      shared structures schema.
    </documentation>
  </annotation>
```

```
   <!--
   Object authorization information types.
   -->
     <complexType name="pwAuthInfoType">
       <simpleContent>
         <extension base="normalizedString">
           <attribute name="roid" type="eppcom:roidType"/>
         </extension>
       </simpleContent>
     </complexType>

     <complexType name="extAuthInfoType">
       <sequence>
         <any namespace="##other"/>
       </sequence>
     </complexType>

   <!--
   <check> response types.
   -->
     <complexType name="reasonType">
       <simpleContent>
         <extension base="eppcom:reasonBaseType">
           <attribute name="lang" type="language"/>
         </extension>
       </simpleContent>
     </complexType>

     <simpleType name="reasonBaseType">
       <restriction base="token">
         <minLength value="1"/>
         <maxLength value="32"/>
       </restriction>
     </simpleType>

   <!--
   Abstract client and object identifier type.
   -->
     <simpleType name="clIDType">
       <restriction base="token">
         <minLength value="3"/>
         <maxLength value="16"/>
       </restriction>
     </simpleType>

   <!--
   DNS label type.
   -->
```

```
     <simpleType name="labelType">
       <restriction base="token">
         <minLength value="1"/>
         <maxLength value="255"/>
       </restriction>
     </simpleType>

   <!--
   Non-empty token type.
   -->
     <simpleType name="minTokenType">
       <restriction base="token">
         <minLength value="1"/>
       </restriction>
     </simpleType>

   <!--
   Repository Object IDentifier type.
   -->
     <simpleType name="roidType">
       <restriction base="token">
         <pattern value="(\w|_){1,80}-\w{1,8}"/>
       </restriction>
     </simpleType>

   <!--
   Transfer status identifiers.
   -->

     <simpleType name="trStatusType">
       <restriction base="token">
         <enumeration value="clientApproved"/>
         <enumeration value="clientCancelled"/>
         <enumeration value="clientRejected"/>
         <enumeration value="pending"/>
         <enumeration value="serverApproved"/>
         <enumeration value="serverCancelled"/>
       </restriction>
     </simpleType>

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

5. 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 in environments where parser-encoding-support incompatibility exists.

EPP includes a provision for returning a human-readable message with every result code. This document describes result codes in English, but the actual text returned with a result MAY be provided in a language negotiated when a session is established. Languages other than English MUST be noted through specification of a "lang" attribute for each message. Valid values for the "lang" attribute and "lang" negotiation elements are described in [RFC4646].

All date-time values presented via EPP MUST be expressed in Universal Coordinated Time using the Gregorian calendar. XML Schema allows use of time zone identifiers to indicate offsets from the zero meridian, but this option MUST NOT be used with EPP. The extended date-time form using upper case "T" and "Z" characters defined in [W3C.REC-xmlschema-2-20041028] MUST be used to represent date-time values, as XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

6. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. Four URI assignments have been registered by the IANA.

Registration request for the EPP namespace:

URI: urn:ietf:params:xml:ns:epp-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 EPP XML schema:

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

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

XML: See the "Base Schema" section of this document.

Registration request for the EPP shared structure namespace:

URI: urn:ietf:params:xml:ns:eppcom-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 EPP shared structure XML schema:

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

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

XML: See the "Shared Structure Schema" section of this document.

A MIME media type registration template is included in Appendix B.

7. Security Considerations

EPP provides only simple client-authentication services. A passive attack is sufficient to recover client identifiers and passwords, allowing trivial command forgery. Protection against most common attacks and more robust security services MUST be provided by other protocol layers. Specifically, EPP instances MUST be protected using a transport mechanism or application protocol that provides integrity, confidentiality, and mutual, strong client-server authentication.

EPP uses a variant of the PLAIN SASL mechanism described in [RFC4616] to provide a simple application-layer authentication service that augments or supplements authentication and identification services that might be available at other protocol layers. Where the PLAIN SASL mechanism specifies provision of an authorization identifier, authentication identifier, and password as a single string separated by ASCII NUL characters, EPP specifies use of a combined authorization and authentication identifier and a password provided as distinct XML elements.

Repeated password guessing attempts can be discouraged by limiting the number of <login> attempts that can be attempted on an open connection. A server MAY close an open connection if multiple <login> attempts are made with either an invalid client identifier, an invalid password, or both an invalid client identifier and an invalid password.

EPP uses authentication information associated with objects to confirm object-transfer authority. Authentication information exchanged between EPP clients and third-party entities MUST be exchanged using a facility that provides privacy and integrity services to protect against unintended disclosure and modification while in transit.

EPP instances SHOULD be protected using a transport mechanism or application protocol that provides anti-replay protection. EPP provides some protection against replay attacks through command idempotency and client-initiated transaction identification. Consecutive command replays will not change the state of an object in any way. There is, however, a chance of unintended or malicious consequence if a command is replayed after intervening commands have changed the object state and client identifiers are not used to detect replays. For example, a replayed <create> command that follows a <delete> command might succeed without additional facilities to prevent or detect the replay.

As described in Section 2, EPP includes features that allow for offline review of transform commands before the requested action is actually completed. The server is required to notify the client when offline processing of the action has been completed. Notifications can be sent using an out-of-band mechanism that is not protected by the mechanism used to provide EPP transport security. Notifications sent without EPP’s transport-security services should be protected using another mechanism that provides an appropriate level of protection for the notification.

8. Acknowledgements

RFC 3730 is a product of the PROVREG working group, which suggested improvements and provided many invaluable comments. The author wishes to acknowledge the efforts of WG chairs Edward Lewis and Jaap Akkerhuis for their process and editorial contributions. RFC 4930 and this document are individual submissions, based on the work done in RFC 3730.

Specific suggestions that have been incorporated into this document were provided by Chris Bason, Eric Brunner-Williams, Jordyn Buchanan, Roger Castillo Cortazar, Dave Crocker, Ayesha Damaraju, Sheer El-Showk, Patrik Faltstrom, James Gould, John Immordino, Dan Kohn, Hong Liu, Klaus Malorny, Dan Manley, Michael Mealling, Patrick Mevzek, Andrew Newton, Budi Rahardjo, Asbjorn Steira, Rick Wesson, and Jay Westerdal.

9. References

9.1. Normative References

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

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[RFC4646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 4646, September 2006.

[W3C.REC-xml-20040204] Sperberg-McQueen, C., Maler, E., Yergeau, F., Paoli, J., and T. Bray, "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] Maloney, M., Thompson, H., Mendelsohn, N., and D. Beech, "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] Malhotra, A. and P. Biron, "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>.

9.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

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

[RFC4616] Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, August 2006.

[RFC4930] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", RFC 4930, May 2007.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[W3C.REC-P3P-20020416]
Marchiori, M., "The Platform for Privacy Preferences 1.0 (P3P1.0) Specification", World Wide Web Consortium Recommendation REC-P3P-20020416, April 2002, <http://www.w3.org/TR/2002/REC-P3P-20020416>.

Appendix A. Object Mapping Template

This appendix describes a recommended outline for documenting the EPP mapping of an object. Documents that describe EPP object mappings SHOULD describe the mapping in a format similar to the one used here. Additional sections are required if the object mapping is written in Internet-Draft or RFC format.

1. Introduction

Provide an introduction that describes the object and gives an overview of the mapping to EPP.

2. Object Attributes

Describe the attributes associated with the object, including references to syntax specifications as appropriate. Examples of object attributes include a name or identifier and dates associated with modification events.

3. EPP Command Mapping

3.1. EPP Query Commands

3.1.1. EPP <check> Command

Describe the object-specific mappings required to implement the EPP <check> command. Include both sample commands and sample responses.

3.1.2. EPP <info> Command

Describe the object-specific mappings required to implement the EPP <info> command. Include both sample commands and sample responses.

3.1.3. EPP <poll> Command

Describe the object-specific mappings required to implement the EPP <poll> command. Include both sample commands and sample responses.

3.1.4. EPP <transfer> Command

Describe the object-specific mappings required to implement the EPP <transfer> query command. Include both sample commands and sample responses.

3.2. EPP Transform Commands

3.2.1. EPP <create> Command

Describe the object-specific mappings required to implement the EPP <create> command. Include both sample commands and sample responses. Describe the status of the object with respect to time, including expected client and server behavior if a validity period is used.

3.2.2. EPP <delete> Command

Describe the object-specific mappings required to implement the EPP <delete> command. Include both sample commands and sample responses.

3.2.3. EPP <renew> Command

Describe the object-specific mappings required to implement the EPP <renew> command. Include both sample commands and sample responses.

3.2.4. EPP <transfer> Command

Describe the object-specific mappings required to implement the EPP <transfer> command. Include both sample commands and sample responses.

3.2.4. EPP <update> Command

Describe the object-specific mappings required to implement the EPP <update> command. Include both sample commands and sample responses.

4. Formal Syntax

Provide the XML schema for the object mapping. An XML DTD MUST NOT be used, as DTDs do not provide sufficient support for XML namespaces and strong data typing.

Appendix B. Media Type Registration: application/epp+xml

MIME media type name: application

MIME subtype name: epp+xml

Required parameters: none

Optional parameters: Same as the charset parameter of application/xml as specified in [RFC3023].

Encoding considerations: Same as the encoding considerations of application/xml as specified in [RFC3023].

Security considerations: This type has all of the security considerations described in [RFC3023] plus the considerations specified in the Security Considerations section of this document.

Interoperability considerations: XML has proven to be interoperable across WWW Distributed Authoring and Versioning (WebDAV) clients and servers, and for import and export from multiple XML authoring tools. For maximum interoperability, validating processors are recommended. Although non-validating processors can be more efficient, they are not required to handle all features of XML. For further information, see Section 2.9, "Standalone Document Declaration", and Section 5, "Conformance", of [W3C.REC-xml-20040204].

Published specification: This document.

Applications that use this media type: EPP is device-, platform-, and vendor-neutral and is supported by multiple service providers.

Additional information: If used, magic numbers, fragment identifiers, base URIs, and use of the BOM should be as specified in [RFC3023].

Magic number(s): None.

File extension(s): .xml

Macintosh file type code(s): "TEXT"

Person & email address for further information: See the "Author's Address" section of this document.

Intended usage: COMMON

Author/Change controller: IETF

Appendix C. Changes from RFC 4930

1. Changed "This document obsoletes RFC 3730" to "This document obsoletes RFC 4930".

2. Replaced references to RFC 2595 with references to RFC 4616.

3. Replaced references to RFC 2821 with references to RFC 5321.

4. Replaced references to RFC 2960 with references to RFC 4960.

5. Replaced references to RFC 3066 with references to RFC 4646.

6. Replaced references to RFC 3730 with references to RFC 4930.

7. Added "A protocol client that is authorized to manage an existing object is described as a "sponsoring" client throughout this document" in Section 1.1.

8. Changed "This action MUST be open to all authorized clients" to "This command MUST be available to all clients" in the descriptions of the <login> and <logout> commands.

9. Changed "Specific result codes are listed in the table below" to "The complete list of valid result codes is enumerated below and in the normative schema" in Section 3.

10. Added new paragraph to Section 7 to give guidance on the need to protect offline transaction notices.

11. Added reference to Appendix B in the IANA Considerations section.

12. Added BSD license text to XML schema section.

Author's Address

Scott Hollenbeck
VeriSign, Inc.
21345 Ridgetop Circle
Dulles, VA 20166-6503
US

EMail: shollenbeck@verisign.com

==

Network Working Group   S. Hollenbeck
Request for Comments: 5731   VeriSign, Inc.
STD: 69   August 2009
Obsoletes: 4931
Category: Standards Track

Extensible Provisioning Protocol (EPP) Domain Name Mapping

Abstract

This document describes an Extensible Provisioning Protocol (EPP) mapping for the provisioning and management of Internet domain names stored in a shared central repository. Specified in XML, the mapping defines EPP command syntax and semantics as applied to domain names. This document obsoletes RFC 4931.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes an Internet domain name mapping for version 1.0 of the Extensible Provisioning Protocol (EPP). This mapping is specified using the Extensible Markup Language (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]. This document obsoletes RFC 4931 [RFC4931].

[RFC5730] provides a complete description of EPP command and response structures. A thorough understanding of the base protocol specification is necessary to understand the mapping described in this document.

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

1.1. Relationship of Domain Objects and Host Objects

The EPP mapping for host objects is described in [RFC5732]. This document assumes that domain name objects have a superordinate relationship to subordinate host name objects. For example, domain name "example.com" has a superordinate relationship to host name "ns1.example.com". EPP actions (such as object transfers) that do not preserve this relationship MUST be explicitly disallowed.

A host name object can be created in a repository for which no superordinate domain name object exists. For example, host name "ns1.example.com" can be created in the ".example" repository so that DNS domains in ".example" can be delegated to the host. Such hosts are described as "external" hosts in this specification since the name of the host does not belong to the namespace of the repository in which the host is being used for delegation purposes.

Whether a host is external or internal relates to the repository in which the host is being used for delegation purposes. Whether or not an internal host is subordinate relates to a domain within the repository. For example, host ns1.example1.com is a subordinate host of domain example1.com, but it is not a subordinate host of domain example2.com. ns1.example1.com can be used as a name server for example2.com. In this case, ns1.example1.com MUST be treated as an internal host, subject to the rules governing operations on subordinate hosts within the same repository.

Name server hosts for domain delegation can be specified either as references to existing host objects or as domain attributes that describe a host machine. A server operator MUST use one name server specification form consistently. A server operator that announces support for host objects in an EPP greeting MUST NOT allow domain attributes to describe a name server host machine. A server operator that does not announce support for host objects MUST allow domain attributes to describe a name server host machine. When domain attributes are used to describe a name server host machine, IP addresses SHOULD be required only as needed to generate DNS glue records.

Name servers are specified within a <domain:ns> element. This element MUST contain one or more <domain:hostObj> elements or one or more <domain:hostAttr> elements. A <domain:hostObj> element contains the fully qualified name of a known name server host object. A <domain:hostAttr> element contains the following child elements:

- A <domain:hostName> element that contains the fully qualified name of a host.

- Zero or more OPTIONAL <domain:hostAddr> elements that contain the IP addresses to be associated with the host. Each element MAY contain an "ip" attribute to identify the IP address format. Attribute value "v4" is used to note IPv4 address format. Attribute value "v6" is used to note IPv6 address format. If the "ip" attribute is not specified, "v4" is the default attribute value. IP address syntax requirements are described in Section 2.5 of the EPP host mapping [RFC5732].

Example host-object name server elements for domain example.com:

```
<domain:ns>
  <domain:hostObj>ns1.example.net</domain:hostObj>
  <domain:hostObj>ns2.example.net</domain:hostObj>
</domain:ns>
```

Example host-attribute name server elements for domain example.com:

```
<domain:ns>
  <domain:hostAttr>
    <domain:hostName>ns1.example.net</domain:hostName>
    <domain:hostAddr
     ip="v4">192.0.2.2</domain:hostAddr>
    <domain:hostAddr
     ip="v6">1080:0:0:0:8:800:200C:417A</domain:hostAddr>
  </domain:hostAttr>
  <domain:hostAttr>
    <domain:hostName>ns2.example.net</domain:hostName>
  </domain:hostAttr>
</domain:ns>
```

1.2. 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 protocol.

2. Object Attributes

An EPP domain 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.

2.1. Domain and Host Names

The syntax for domain and host names described in this document MUST conform to [RFC0952] and [RFC1123]. At the time of this writing, RFC 3490 [RFC3490] describes a standard to use certain ASCII name labels to represent non-ASCII name labels. These conformance requirements might change as a result of progressing work in developing standards for internationalized domain names. A server MAY restrict allowable domain names to a particular top-level domain, second-level domain, or other domain for which the server is authoritative. The trailing dot required when these names are stored in a DNS zone is implicit and MUST NOT be provided when exchanging host and domain names.

2.2. Contact and Client Identifiers

All EPP contacts are identified by a server-unique identifier. Contact identifiers are character strings with a specified minimum length, a specified maximum length, and a specified format. Contact identifiers use the "clIDType" client identifier syntax described in [RFC5730].

2.3. Status Values

A domain object MUST always have at least one associated status value. Status values can be set only by the client that sponsors a domain object and by the server on which the object resides. A client can change the status of a domain object using the EPP

<update> command. Each status value MAY be accompanied by a string of human-readable text that describes the rationale for the status applied to the object.

A client MUST NOT alter status values set by the server. A server MAY alter or override status values set by a client, subject to local server policies. The status of an object MAY change as a result of either a client-initiated transform command or an action performed by a server operator.

Status values that can be added or removed by a client are prefixed with "client". Corresponding status values that can be added or removed by a server are prefixed with "server". Status values that do not begin with either "client" or "server" are server-managed.

Status Value Descriptions:

- clientDeleteProhibited, serverDeleteProhibited

  Requests to delete the object MUST be rejected.

- clientHold, serverHold

  DNS delegation information MUST NOT be published for the object.

- clientRenewProhibited, serverRenewProhibited

  Requests to renew the object MUST be rejected.

- clientTransferProhibited, serverTransferProhibited

  Requests to transfer the object MUST be rejected.

- clientUpdateProhibited, serverUpdateProhibited

  Requests to update the object (other than to remove this status) MUST be rejected.

- inactive

  Delegation information has not been associated with the object. This is the default status when a domain object is first created and there are no associated host objects for the DNS delegation. This status can also be set by the server when all host-object associations are removed.

- ok

  This is the normal status value for an object that has no pending operations or prohibitions. This value is set and removed by the server as other status values are added or removed.

- pendingCreate, pendingDelete, pendingRenew, pendingTransfer, pendingUpdate

  A transform command has been processed for the object, but the action has not been completed by the server. Server operators can delay action completion for a variety of reasons, such as to allow for human review or third-party action. A transform command that is processed, but whose requested action is pending, is noted with response code 1001.

When the requested action has been completed, the pendingCreate, pendingDelete, pendingRenew, pendingTransfer, or pendingUpdate status value MUST be removed. 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.

"ok" status MUST NOT be combined with any other status.

"pendingDelete" status MUST NOT be combined with either "clientDeleteProhibited" or "serverDeleteProhibited" status.

"pendingRenew" status MUST NOT be combined with either "clientRenewProhibited" or "serverRenewProhibited" status.

"pendingTransfer" status MUST NOT be combined with either "clientTransferProhibited" or "serverTransferProhibited" status.

"pendingUpdate" status MUST NOT be combined with either "clientUpdateProhibited" or "serverUpdateProhibited" status.

The pendingCreate, pendingDelete, pendingRenew, pendingTransfer, and pendingUpdate status values MUST NOT be combined with each other.

Other status combinations not expressly prohibited MAY be used.

2.4. Dates and Times

Date and time attribute values MUST be represented in Universal Coordinated Time (UTC) using the Gregorian calendar. The extended date-time form using upper case "T" and "Z" characters defined in

[W3C.REC-xmlschema-2-20041028] MUST be used to represent date-time values, as XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

2.5. Validity Periods

A domain name object MAY have a specified validity period. If server policy supports domain-object validity periods, the validity period is defined when a domain object is created, and it MAY be extended by the EPP <renew> or <transfer> commands. As a matter of server policy, this specification does not define actions to be taken upon expiration of a domain object’s validity period.

Validity periods are measured in years or months with the appropriate units specified using the "unit" attribute. Valid values for the "unit" attribute are "y" for years and "m" for months. The minimum allowable period value is one (1). The maximum allowable value is ninety-nine decimal (99). A server MAY support a lower maximum value.

2.6. Authorization Information

Authorization information is associated with domain objects to facilitate transfer operations. Authorization information is assigned when a domain object is created, and it might be updated in the future. This specification describes password-based authorization information, though other mechanisms are possible.

2.7. Other DNS Resource Record Attributes

While the DNS allows many resource record types to be associated with a domain, this mapping only explicitly specifies elements that describe resource records used for domain delegation and resolution. Facilities to provision other domain-related resource record types can be developed by extending this mapping.

The provisioning method described in this mapping separates discrete data elements by data type. This method of data definition allows XML Schema processors to perform basic syntax-validation tasks, reducing ambiguity and the amount of parsing and syntax-checking work required of protocol processors. Provisioning and extension methods that aggregate data into opaque strings are possible, but such methods should not be used because they impose additional parsing, interpretation, and validation requirements on protocol processors.

3. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in [RFC5730]. The command mappings described here are specifically for use in provisioning and managing Internet domain names via EPP.

3.1. 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.

3.1.1. 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 <domain:check> element that identifies the domain namespace. The <domain:check> element contains the following child elements:

- One or more <domain:name> elements that contain the fully qualified names of the domain 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:      <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.org</domain:name>
C:      </domain: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 <domain:chkData> element that identifies the domain namespace. The <domain:chkData> element contains one or more <domain:cd> elements that contain the following child elements:

- A <domain:name> element that contains the fully qualified name of the queried domain 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 can not be provisioned.

- An OPTIONAL <domain: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:      <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="0">example.net</domain:name>
S:          <domain:reason>In use</domain:reason>
S:        </domain:cd>
S:        <domain:cd>
S:          <domain:name avail="1">example.org</domain:name>
S:        </domain:cd>
S:      </domain: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.

3.1.2. EPP <info> Command

The EPP <info> command is used to retrieve information associated with a domain object. The response to this command MAY vary depending on the identity of the querying client, use of authorization information, and server policy towards unauthorized clients. If the querying client is the sponsoring client, all available information MUST be returned. If the querying client is not the sponsoring client but the client provides valid authorization information, all available information MUST be returned. If the querying client is not the sponsoring client and the client does not provide valid authorization information, server policy determines which OPTIONAL elements are returned.

In addition to the standard EPP command elements, the <info> command MUST contain a <domain:info> element that identifies the domain namespace. The <domain:info> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object to be queried. An OPTIONAL "hosts" attribute is available to control return of information describing hosts related to the domain object. A value of "all" (the default, which MAY be absent) returns information describing both subordinate and delegated hosts. A value of "del" returns information describing only delegated hosts. A value of "sub" returns information describing only subordinate hosts. A value of "none" returns no information describing delegated or subordinate hosts.

- An OPTIONAL <domain:authInfo> element that contains authorization information associated with the domain object or authorization information associated with the domain object’s registrant or associated contacts. An OPTIONAL "roid" attribute MUST be used to identify the registrant or contact object if and only if the given authInfo is associated with a registrant or contact object, and not the domain object itself. If this element is not provided or if the authorization information is invalid, server policy determines if the command is rejected or if response information will be returned to the client.

Example <info> command without authorization 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:    <info>
C:      <domain:info
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name hosts="all">example.com</domain:name>
C:      </domain:info>
C:    </info>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

Example <info> command with authorization 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:    <info>
C:      <domain:info
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name hosts="all">example.com</domain:name>
C:        <domain:authInfo>
C:          <domain:pw>2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain: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 <domain:infData> element that identifies the domain namespace. Elements that are not OPTIONAL MUST be returned; OPTIONAL elements are returned based on client authorization and server policy. The <domain:infData> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object.

- A <domain:roid> element that contains the Repository Object IDentifier assigned to the domain object when the object was created.

- Zero or more OPTIONAL <domain:status> elements that contain the current status descriptors associated with the domain.

- If supported by the server, one OPTIONAL <domain:registrant> element and one or more OPTIONAL <domain:contact> elements that contain identifiers for the human or organizational social information objects associated with the domain object.

- An OPTIONAL <domain:ns> element that contains the fully qualified names of the delegated host objects or host attributes (name servers) associated with the domain object. See Section 1.1 for a description of the elements used to specify host objects or host attributes.

- Zero or more OPTIONAL <domain:host> elements that contain the fully qualified names of the subordinate host objects that exist under this superordinate domain object.

- A <domain:clID> element that contains the identifier of the sponsoring client.

- An OPTIONAL <domain:crID> element that contains the identifier of the client that created the domain object.

- An OPTIONAL <domain:crDate> element that contains the date and time of domain object creation.

- An OPTIONAL <domain:exDate> element that contains the date and time identifying the end of the domain object's registration period.

- An OPTIONAL <domain:upID> element that contains the identifier of the client that last updated the domain object. This element MUST NOT be present if the domain has never been modified.

- An OPTIONAL <domain:upDate> element that contains the date and time of the most recent domain-object modification. This element MUST NOT be present if the domain object has never been modified.

- An OPTIONAL <domain:trDate> element that contains the date and time of the most recent successful domain-object transfer. This element MUST NOT be provided if the domain object has never been transferred.

- An OPTIONAL <domain:authInfo> element that contains authorization information associated with the domain object. This element MUST only be returned if the querying client is the current sponsoring client or if the client supplied valid authorization information with the command.

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>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>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="tech">sh8013</domain:contact>
S:        <domain:ns>
S:          <domain:hostObj>ns1.example.com</domain:hostObj>
S:          <domain:hostObj>ns1.example.net</domain:hostObj>
S:        </domain:ns>
S:        <domain:host>ns1.example.com</domain:host>
S:        <domain:host>ns2.example.com</domain:host>
S:        <domain:clID>ClientX</domain:clID>
S:        <domain:crID>ClientY</domain:crID>
S:        <domain:crDate>1999-04-03T22:00:00.0Z</domain:crDate>
S:        <domain:upID>ClientX</domain:upID>
S:        <domain:upDate>1999-12-03T09:00:00.0Z</domain:upDate>
S:        <domain:exDate>2005-04-03T22:00:00.0Z</domain:exDate>
S:        <domain:trDate>2000-04-08T09:00:00.0Z</domain:trDate>
S:        <domain:authInfo>
S:          <domain:pw>2fooBAR</domain:pw>
S:        </domain:authInfo>
S:      </domain:infData>
S:    </resData>
S:    <trID>
S:      <clTRID>ABC-12345</clTRID>
S:      <svTRID>54322-XYZ</svTRID>
S:    </trID>
S:  </response>
S:</epp>
```

A server with a different information-return policy MAY provide less information in a response.

Example <info> response for an unauthorized 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:      <domain:infData
S:       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:clID>ClientX</domain:clID>
S:      </domain: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.

3.1.3. EPP <transfer> Query Command

The EPP <transfer> command provides a query operation that allows a client to determine the real-time status of pending and completed transfer requests. In addition to the standard EPP command elements, the <transfer> command MUST contain an "op" attribute with value "query", and a <domain:transfer> element that identifies the domain namespace. The <domain:transfer> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object to be queried.

- An OPTIONAL <domain:authInfo> element that contains authorization information associated with the domain object or authorization information associated with the domain object’s registrant or associated contacts. An OPTIONAL "roid" attribute MUST be used to identify the registrant or contact object if and only if the given authInfo is associated with a registrant or contact object, and not the domain object itself. If this element is not provided or if the authorization information is invalid, server policy determines if the command is rejected or if response information will be returned to the client.

Example <transfer> query 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="query">
C:      <domain:transfer
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:        <domain:authInfo>
C:          <domain:pw roid="JD1234-REP">2fooBAR</domain:pw>
C:        </domain:authInfo>
C:      </domain:transfer>
C:    </transfer>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <transfer> query command has been processed successfully, the EPP <resData> element MUST contain a child <domain:trnData> element that identifies the domain namespace. The <domain:trnData> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object.

- A <domain:trStatus> element that contains the state of the most recent transfer request.

- A <domain:reID> element that contains the identifier of the client that requested the object transfer.

- A <domain:reDate> element that contains the date and time that the transfer was requested.

- A <domain:acID> element that contains the identifier of the client that SHOULD act upon a PENDING transfer request. For all other status types, the value identifies the client that took the indicated action.

- A <domain:acDate> element that contains the date and time of a required or completed response. For a PENDING request, the value identifies the date and time by which a response is required before an automated response action will be taken by the server. For all other status types, the value identifies the date and time when the request was completed.

- An OPTIONAL <domain:exDate> element that contains the end of the domain object’s validity period if the <transfer> command caused or causes a change in the validity period.

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="1000">
S:      <msg>Command completed successfully</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>2000-06-06T22:00:00.0Z</domain:reDate>
S:        <domain:acID>ClientY</domain:acID>
S:        <domain:acDate>2000-06-11T22:00:00.0Z</domain:acDate>
S:        <domain:exDate>2002-09-08T22:00:00.0Z</domain:exDate>
S:      </domain:trnData>
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 <transfer> query command cannot be processed for any reason.

3.2. 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.

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.

Server operators SHOULD confirm that a client is authorized to perform a transform command on a given object. Any attempt to transform an object by an unauthorized client MUST be rejected, and the server MUST return a 2201 response code to the client to note that the client lacks privileges to execute the requested command.

3.2.1. EPP <create> Command

The EPP <create> command provides a transform operation that allows a client to create a domain object. In addition to the standard EPP command elements, the <create> command MUST contain a <domain:create> element that identifies the domain namespace. The <domain:create> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object to be created.

- An OPTIONAL <domain:period> element that contains the initial registration period of the domain object. A server MAY define a default initial registration period if not specified by the client.

- An OPTIONAL <domain:ns> element that contains the fully qualified names of the delegated host objects or host attributes (name servers) associated with the domain object to provide resolution services for the domain; see Section 1.1 for a description of the elements used to specify host objects or host attributes. A host object MUST be known to the server before the host object can be associated with a domain object.

- An OPTIONAL <domain:registrant> element that contains the identifier for the human or organizational social information (contact) object to be associated with the domain object as the object registrant. This object identifier MUST be known to the server before the contact object can be associated with the domain object. The EPP mapping for contact objects is described in [RFC5733].

- Zero or more OPTIONAL <domain:contact> elements that contain the identifiers for other contact objects to be associated with the domain object. Contact object identifiers MUST be known to the server before the contact object can be associated with the domain object.

- A <domain:authInfo> element that contains authorization information to be associated with the domain object. This mapping includes a password-based authentication mechanism, but the schema allows new mechanisms to be defined in new schemas.

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:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <create> command has been processed successfully, the EPP <resData> element MUST contain a child <domain:creData> element that identifies the domain namespace. The <domain:creData> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object.

- A <domain:crDate> element that contains the date and time of domain object creation.

- An OPTIONAL <domain:exDate> element that contains the date and time identifying the end of the domain object's registration period.

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:    <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.

3.2.2. EPP <delete> Command

The EPP <delete> command provides a transform operation that allows a client to delete a domain object. In addition to the standard EPP command elements, the <delete> command MUST contain a <domain:delete> element that identifies the domain namespace. The <domain:delete> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object to be deleted.

A domain object SHOULD NOT be deleted if subordinate host objects are associated with the domain object. For example, if domain "example.com" exists and host object "ns1.example.com" also exists, then domain "example.com" SHOULD NOT be deleted until host "ns1.example.com" has either been deleted or renamed to exist in a different superordinate domain. 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. Delegated and subordinate host objects associated with a domain object can be determined using the <info> query command for the domain object.

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:      <domain:delete
C:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
C:        <domain:name>example.com</domain:name>
C:      </domain: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.

3.2.3. EPP <renew> Command

The EPP <renew> command provides a transform operation that allows a client to extend the validity period of a domain object. In addition to the standard EPP command elements, the <renew> command MUST contain a <domain:renew> element that identifies the domain namespace. The <domain:renew> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object whose validity period is to be extended.

- A <domain:curExpDate> element that contains the date on which the current validity period ends. This value ensures that repeated <renew> commands do not result in multiple, unanticipated successful renewals.

- An OPTIONAL <domain:period> element that contains the number of units to be added to the registration period of the domain object. The number of units available MAY be subject to limits imposed by the server.

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:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <renew> command has been processed successfully, the EPP <resData> element MUST contain a child <domain:renData> element that identifies the domain namespace. The <domain:renData> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object.

- An OPTIONAL <domain:exDate> element that contains the date and time identifying the end of the domain object’s registration period.

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:    <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 <renew> command cannot be processed for any reason.

3.2.4. EPP <transfer> Command

The EPP <transfer> command provides a transform operation that allows a client to manage requests to transfer the sponsorship of a domain object. In addition to the standard EPP command elements, the <transfer> command MUST contain a <domain:transfer> element that identifies the domain namespace. The <domain:transfer> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object for which a transfer request is to be created, approved, rejected, or cancelled.

- An OPTIONAL <domain:period> element that contains the number of units to be added to the registration period of the domain object at completion of the transfer process. This element can only be used when a transfer is requested, and it MUST be ignored if used otherwise. The number of units available MAY be subject to limits imposed by the server.

- A <domain:authInfo> element that contains authorization information associated with the domain object or authorization information associated with the domain object’s registrant or associated contacts. An OPTIONAL "roid" attribute MUST be used to identify the registrant or contact object if and only if the given authInfo is associated with a registrant or contact object, and not the domain object itself.

Every EPP <transfer> command MUST contain an "op" attribute that identifies the transfer operation to be performed. Valid values, definitions, and authorizations for all attribute values are defined in [RFC5730].

Transfer of a domain object MUST implicitly transfer all host objects that are subordinate to the domain object. For example, if domain object "example.com" is transferred and host object "ns1.example.com" exists, the host object MUST be transferred as part of the "example.com" transfer process. Host objects that are subject to transfer when transferring a domain object are listed in the response to an EPP <info> command performed on the domain object.

Example <transfer> request 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:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <transfer> command has been processed successfully, the EPP <resData> element MUST contain a child <domain:trnData> element that identifies the domain namespace. The <domain:trnData> element contains the same child elements defined for a transfer query response.

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:    <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 <transfer> command can not be processed for any reason.

3.2.5. EPP <update> Command

The EPP <update> command provides a transform operation that allows a client to modify the attributes of a domain object. In addition to the standard EPP command elements, the <update> command MUST contain a <domain:update> element that identifies the domain namespace. The <domain:update> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object to be updated.

- An OPTIONAL <domain:add> element that contains attribute values to be added to the object.

- An OPTIONAL <domain:rem> element that contains attribute values to be removed from the object.

- An OPTIONAL <domain:chg> element that contains object attribute values to be changed.

At least one <domain:add>, <domain:rem>, or <domain:chg> 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 <domain:add> and <domain:rem> elements contain the following child elements:

- An OPTIONAL <domain:ns> element that contains the fully qualified names of the delegated host objects or host attributes (name servers) associated with the domain object to provide resolution services for the domain; see Section 1.1 for a description of the elements used to specify host objects or host attributes. A host object MUST be known to the server before the host object can be associated with a domain object. If host attributes are used to specify name servers, note that IP address elements are not needed to identify a name server that is being removed. IP address elements can safely be absent or ignored in this situation.

- Zero or more <domain:contact> elements that contain the identifiers for contact objects to be associated with or removed from the domain object. Contact object identifiers MUST be known to the server before the contact object can be associated with the domain object.

- Zero or more <domain:status> elements that contain status values to be applied to or removed from the object. When specifying a value to be removed, only the attribute value is significant; element text is not required to match a value for removal.

A <domain:chg> element contains the following child elements:

- A <domain:registrant> element that contains the identifier for the human or organizational social information (contact) object to be associated with the domain object as the object registrant. This object identifier MUST be known to the server before the contact object can be associated with the domain object. An empty element can be used to remove registrant information.

- A <domain:authInfo> element that contains authorization information associated with the domain object. This mapping includes a password-based authentication mechanism, but the schema allows new mechanisms to be defined in new schemas. A <domain: null> element can be used within the <domain:authInfo> element to remove authorization information.

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:add>
C:          <domain:ns>
C:            <domain:hostObj>ns2.example.com</domain:hostObj>
C:          </domain:ns>
C:          <domain:contact type="tech">mak21</domain:contact>
C:          <domain:status s="clientHold"
C:           lang="en">Payment overdue.</domain:status>
C:        </domain:add>
C:        <domain:rem>
C:          <domain:ns>
C:            <domain:hostObj>ns1.example.com</domain:hostObj>
C:          </domain:ns>
C:          <domain:contact type="tech">sh8013</domain:contact>
C:          <domain:status s="clientUpdateProhibited"/>
C:        </domain:rem>
C:        <domain:chg>
C:          <domain:registrant>sh8013</domain:registrant>
C:          <domain:authInfo>
C:            <domain:pw>2BARfoo</domain:pw>
C:          </domain:authInfo>
C:        </domain:chg>
C:      </domain: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.

3.3. Offline Review of Requested Actions

Commands are processed by a server in the order they are received from a client. Though an immediate response confirming receipt and processing of the command is produced by the server, a server operator MAY perform an offline review of requested transform commands before completing the requested action. In such situations, the response from the server MUST clearly note that the transform command has been received and processed but that the requested action is pending. The status of the corresponding object MUST clearly reflect processing of the pending action. The server MUST notify the client when offline processing of the action has been completed.

Examples describing a <create> command that requires offline review are included here. Note the result code and message returned in response to the <create> command.

```
   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>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:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

The status of the domain object after returning this response MUST include "pendingCreate". The server operator reviews the request offline, and informs the client of the outcome of the review either by queuing a service message for retrieval via the <poll> command or by using an out-of-band mechanism to inform the client of the request.

The service message MUST contain text that describes the notification in the child <msg> element of the response <msgQ> element. In addition, the EPP <resData> element MUST contain a child <domain:panData> element that identifies the domain namespace. The <domain:panData> element contains the following child elements:

- A <domain:name> element that contains the fully qualified name of the domain object. The <domain:name> element contains a REQUIRED "paResult" attribute. A positive boolean value indicates that the request has been approved and completed. A negative boolean value indicates that the request has been denied and the requested action has not been taken.

- A <domain:paTRID> element that contains the client transaction identifier and server transaction identifier returned with the original response to process the command. The client transaction identifier is OPTIONAL and will only be returned if the client provided an identifier with the original <create> command.

- A <domain:paDate> element that contains the date and time describing when review of the requested action was completed.

Example "review completed" service message:

```
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>1999-04-04T22:01:00.0Z</qDate>
S:      <msg>Pending action completed successfully.</msg>
S:    </msgQ>
S:    <resData>
S:      <domain:panData
S:       xmlns:domain="urn:ietf:params:xml:ns:domain-1.0">
S:        <domain:name paResult="1">example.com</domain:name>
S:        <domain:paTRID>
S:          <clTRID>ABC-12345</clTRID>
S:          <svTRID>54321-XYZ</svTRID>
```

```
   S:          </domain:paTRID>
   S:          <domain:paDate>1999-04-04T22:00:00.0Z</domain:paDate>
   S:        </domain:panData>
   S:      </resData>
   S:      <trID>
   S:        <clTRID>BCD-23456</clTRID>
   S:        <svTRID>65432-WXY</svTRID>
   S:      </trID>
   S:    </response>
   S:</epp>
```

4. 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:domain-1.0"
     xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
     xmlns:host="urn:ietf:params:xml:ns:host-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"/>
<import namespace="urn:ietf:params:xml:ns:epp-1.0"/>
<import namespace="urn:ietf:params:xml:ns:host-1.0"/>

<annotation>
 <documentation>
   Extensible Provisioning Protocol v1.0
   domain provisioning schema.
 </documentation>
</annotation>

<!--
Child elements found in EPP commands.
-->
<element name="check" type="domain:mNameType"/>
<element name="create" type="domain:createType"/>
<element name="delete" type="domain:sNameType"/>
<element name="info" type="domain:infoType"/>
<element name="renew" type="domain:renewType"/>
<element name="transfer" type="domain:transferType"/>
<element name="update" type="domain:updateType"/>
<!--
Child elements of the <create> command.
-->
<complexType name="createType">
 <sequence>
   <element name="name" type="eppcom:labelType"/>
   <element name="period" type="domain:periodType"
    minOccurs="0"/>
   <element name="ns" type="domain:nsType"
```

```
       minOccurs="0"/>
      <element name="registrant" type="eppcom:clIDType"
       minOccurs="0"/>
      <element name="contact" type="domain:contactType"
       minOccurs="0" maxOccurs="unbounded"/>
      <element name="authInfo" type="domain:authInfoType"/>
    </sequence>
   </complexType>

   <complexType name="periodType">
    <simpleContent>
      <extension base="domain:pLimitType">
        <attribute name="unit" type="domain:pUnitType"
         use="required"/>
      </extension>
    </simpleContent>
   </complexType>

   <simpleType name="pLimitType">
    <restriction base="unsignedShort">
      <minInclusive value="1"/>
      <maxInclusive value="99"/>
    </restriction>
   </simpleType>

   <simpleType name="pUnitType">
    <restriction base="token">
      <enumeration value="y"/>
      <enumeration value="m"/>
    </restriction>
   </simpleType>

   <complexType name="nsType">
    <choice>
      <element name="hostObj" type="eppcom:labelType"
       maxOccurs="unbounded"/>
      <element name="hostAttr" type="domain:hostAttrType"
       maxOccurs="unbounded"/>
    </choice>
   </complexType>
   <!--
   Name servers are either host objects or attributes.
   -->

   <complexType name="hostAttrType">
    <sequence>
      <element name="hostName" type="eppcom:labelType"/>
      <element name="hostAddr" type="host:addrType"
```

```
        minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
   </complexType>
   <!--
   If attributes, addresses are optional and follow the
   structure defined in the host mapping.
   -->

   <complexType name="contactType">
     <simpleContent>
       <extension base="eppcom:clIDType">
         <attribute name="type" type="domain:contactAttrType"/>
       </extension>
     </simpleContent>
   </complexType>

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

   <complexType name="authInfoType">
     <choice>
       <element name="pw" type="eppcom:pwAuthInfoType"/>
       <element name="ext" type="eppcom:extAuthInfoType"/>
     </choice>
   </complexType>

   <!--
   Child element of commands that require a single name.
   -->
   <complexType name="sNameType">
     <sequence>
       <element name="name" type="eppcom:labelType"/>
     </sequence>
   </complexType>
   <!--
   Child element of commands that accept multiple names.
   -->
   <complexType name="mNameType">
     <sequence>
       <element name="name" type="eppcom:labelType"
        maxOccurs="unbounded"/>
     </sequence>
   </complexType>
   <!--
   Child elements of the <info> command.
   -->
   <complexType name="infoType">
    <sequence>
      <element name="name" type="domain:infoNameType"/>
      <element name="authInfo" type="domain:authInfoType"
       minOccurs="0"/>
    </sequence>
   </complexType>

   <complexType name="infoNameType">
    <simpleContent>
      <extension base = "eppcom:labelType">
        <attribute name="hosts" type="domain:hostsType"
         default="all"/>
      </extension>
    </simpleContent>
   </complexType>

   <simpleType name="hostsType">
    <restriction base="token">
      <enumeration value="all"/>
      <enumeration value="del"/>
      <enumeration value="none"/>
      <enumeration value="sub"/>
    </restriction>
   </simpleType>

   <!--
   Child elements of the <renew> command.
   -->
   <complexType name="renewType">
    <sequence>
      <element name="name" type="eppcom:labelType"/>
      <element name="curExpDate" type="date"/>
      <element name="period" type="domain:periodType"
       minOccurs="0"/>
    </sequence>
   </complexType>

   <!--
   Child elements of the <transfer> command.
   -->
   <complexType name="transferType">
    <sequence>
      <element name="name" type="eppcom:labelType"/>
      <element name="period" type="domain:periodType"
```

```
         minOccurs="0"/>
        <element name="authInfo" type="domain:authInfoType"
         minOccurs="0"/>
    </sequence>
   </complexType>

   <!--
   Child elements of the <update> command.
   -->
   <complexType name="updateType">
    <sequence>
      <element name="name" type="eppcom:labelType"/>
      <element name="add" type="domain:addRemType"
       minOccurs="0"/>
      <element name="rem" type="domain:addRemType"
       minOccurs="0"/>
      <element name="chg" type="domain:chgType"
       minOccurs="0"/>
    </sequence>
   </complexType>

   <!--
   Data elements that can be added or removed.
   -->
   <complexType name="addRemType">
    <sequence>
      <element name="ns" type="domain:nsType"
       minOccurs="0"/>
      <element name="contact" type="domain:contactType"
       minOccurs="0" maxOccurs="unbounded"/>
      <element name="status" type="domain:statusType"
       minOccurs="0" maxOccurs="11"/>
    </sequence>
   </complexType>

   <!--
   Data elements that can be changed.
   -->
   <complexType name="chgType">
    <sequence>
      <element name="registrant" type="domain:clIDChgType"
       minOccurs="0"/>
      <element name="authInfo" type="domain:authInfoChgType"
       minOccurs="0"/>
    </sequence>
   </complexType>
   <!--
   Allow the registrant value to be nullified by changing the
   minLength restriction to "0".
   -->
   <simpleType name="clIDChgType">
    <restriction base="token">
      <minLength value="0"/>
      <maxLength value="16"/>
    </restriction>
   </simpleType>

   <!--
   Allow the authInfo value to be nullified by including an
   empty element within the choice.
   -->
   <complexType name="authInfoChgType">
    <choice>
      <element name="pw" type="eppcom:pwAuthInfoType"/>
      <element name="ext" type="eppcom:extAuthInfoType"/>
      <element name="null"/>
    </choice>
   </complexType>

   <!--
   Child response elements.
   -->
   <element name="chkData" type="domain:chkDataType"/>
   <element name="creData" type="domain:creDataType"/>
   <element name="infData" type="domain:infDataType"/>
   <element name="panData" type="domain:panDataType"/>
   <element name="renData" type="domain:renDataType"/>
   <element name="trnData" type="domain:trnDataType"/>

   <!--
   <check> response elements.
   -->
   <complexType name="chkDataType">
    <sequence>
      <element name="cd" type="domain:checkType"
       maxOccurs="unbounded"/>
    </sequence>
   </complexType>

   <complexType name="checkType">
    <sequence>
      <element name="name" type="domain:checkNameType"/>
      <element name="reason" type="eppcom:reasonType"
       minOccurs="0"/>
    </sequence>
   </complexType>

   <complexType name="checkNameType">
    <simpleContent>
      <extension base="eppcom:labelType">
        <attribute name="avail" type="boolean"
         use="required"/>
      </extension>
    </simpleContent>
   </complexType>

   <!--
   <create> response elements.
   -->
   <complexType name="creDataType">
    <sequence>
      <element name="name" type="eppcom:labelType"/>
      <element name="crDate" type="dateTime"/>
      <element name="exDate" type="dateTime"
       minOccurs="0"/>
    </sequence>
   </complexType>

   <!--
   <info> response elements.
   -->

   <complexType name="infDataType">
    <sequence>
      <element name="name" type="eppcom:labelType"/>
      <element name="roid" type="eppcom:roidType"/>
      <element name="status" type="domain:statusType"
       minOccurs="0" maxOccurs="11"/>
      <element name="registrant" type="eppcom:clIDType"
       minOccurs="0"/>
      <element name="contact" type="domain:contactType"
       minOccurs="0" maxOccurs="unbounded"/>
      <element name="ns" type="domain:nsType"
       minOccurs="0"/>
      <element name="host" type="eppcom:labelType"
       minOccurs="0" maxOccurs="unbounded"/>
      <element name="clID" type="eppcom:clIDType"/>
      <element name="crID" type="eppcom:clIDType"
       minOccurs="0"/>
      <element name="crDate" type="dateTime"
       minOccurs="0"/>
      <element name="upID" type="eppcom:clIDType"
```

```
       minOccurs="0"/>
      <element name="upDate" type="dateTime"
       minOccurs="0"/>
      <element name="exDate" type="dateTime"
       minOccurs="0"/>
      <element name="trDate" type="dateTime"
       minOccurs="0"/>
      <element name="authInfo" type="domain:authInfoType"
       minOccurs="0"/>
    </sequence>
   </complexType>

   <!--
   Status is a combination of attributes and an optional
   human-readable message that may be expressed in languages other
   than English.
   -->
   <complexType name="statusType">
    <simpleContent>
      <extension base="normalizedString">
        <attribute name="s" type="domain:statusValueType"
         use="required"/>
        <attribute name="lang" type="language"
         default="en"/>
      </extension>
    </simpleContent>
   </complexType>

   <simpleType name="statusValueType">
    <restriction base="token">
      <enumeration value="clientDeleteProhibited"/>
      <enumeration value="clientHold"/>
      <enumeration value="clientRenewProhibited"/>
      <enumeration value="clientTransferProhibited"/>
      <enumeration value="clientUpdateProhibited"/>
      <enumeration value="inactive"/>
      <enumeration value="ok"/>
      <enumeration value="pendingCreate"/>
      <enumeration value="pendingDelete"/>
      <enumeration value="pendingRenew"/>
      <enumeration value="pendingTransfer"/>
      <enumeration value="pendingUpdate"/>
      <enumeration value="serverDeleteProhibited"/>
      <enumeration value="serverHold"/>
      <enumeration value="serverRenewProhibited"/>
      <enumeration value="serverTransferProhibited"/>
      <enumeration value="serverUpdateProhibited"/>
    </restriction>
   </simpleType>

   <!--
   Pending action notification response elements.
   -->
   <complexType name="panDataType">
    <sequence>
      <element name="name" type="domain:paNameType"/>
      <element name="paTRID" type="epp:trIDType"/>
      <element name="paDate" type="dateTime"/>
    </sequence>
   </complexType>

   <complexType name="paNameType">
    <simpleContent>
      <extension base="eppcom:labelType">
        <attribute name="paResult" type="boolean"
         use="required"/>
      </extension>
    </simpleContent>
   </complexType>

   <!--
   <renew> response elements.
   -->
   <complexType name="renDataType">
   <sequence>
    <element name="name" type="eppcom:labelType"/>
    <element name="exDate" type="dateTime"
     minOccurs="0"/>
   </sequence>
   </complexType>

   <!--
   <transfer> response elements.
   -->
   <complexType name="trnDataType">
   <sequence>
    <element name="name" type="eppcom:labelType"/>
    <element name="trStatus" type="eppcom:trStatusType"/>
    <element name="reID" type="eppcom:clIDType"/>
    <element name="reDate" type="dateTime"/>
    <element name="acID" type="eppcom:clIDType"/>
    <element name="acDate" type="dateTime"/>
    <element name="exDate" type="dateTime"
     minOccurs="0"/>
   </sequence>
   </complexType>
```

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

5. 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 [RFC2781]. 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 in environments where parser encoding support incompatibility exists.

All date-time values presented via EPP MUST be expressed in Universal Coordinated Time using the Gregorian calendar. XML Schema allows use of time zone identifiers to indicate offsets from the zero meridian, but this option MUST NOT be used with EPP. The extended date-time form using upper case "T" and "Z" characters, defined in [W3C.REC-xmlschema-2-20041028], MUST be used to represent date-time values, as XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

This document requires domain and host name syntax as specified in [RFC0952] as updated by [RFC1123]. At the time of this writing, RFC 3490 [RFC3490] describes a standard to use certain ASCII name labels to represent non-ASCII name labels. These conformance requirements might change as a result of progressing work in developing standards for internationalized domain names.

6. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. Two URI assignments have been registered by the IANA.

Registration request for the domain namespace:

URI: urn:ietf:params:xml:ns:domain-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 domain XML schema:

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

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

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

7. Security Considerations

Authorization information as described in Section 2.6 is REQUIRED to create a domain object. This information is used in some query and transfer operations as an additional means of determining client authorization to perform the command. Failure to protect authorization information from inadvertent disclosure can result in unauthorized transfer operations and unauthorized information release. Both client and server MUST ensure that authorization information is stored and exchanged with high-grade encryption mechanisms to provide privacy services.

The object mapping described in this document does not provide any other security services or introduce any additional considerations beyond those described by [RFC5730] or those caused by the protocol layers used by EPP.

8. Acknowledgements

RFC 3731 is a product of the PROVREG working group, which suggested improvements and provided many invaluable comments. The author wishes to acknowledge the efforts of WG chairs Edward Lewis and Jaap Akkerhuis for their process and editorial contributions. RFC 4931 and this document are individual submissions, based on the work done in RFC 3731.

Specific suggestions that have been incorporated into this document were provided by Joe Abley, Chris Bason, Eric Brunner-Williams, Jordyn Buchanan, Dave Crocker, Ayesha Damaraju, Anthony Eden, Sheer El-Showk, Klaus Malorny, Dan Manley, Michael Mealling, Patrick Mevzek, Asbjorn Steira, Bruce Tonkin, and Rick Wesson.

9. References

9.1. Normative References

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[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.

[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.

[W3C.REC-xml-20040204]
Sperberg-McQueen, C., Maler, E., Yergeau, F., Paoli, J., and T. Bray, "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]
Maloney, M., Thompson, H., Mendelsohn, N., and D. Beech, "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]
Malhotra, A. and P. Biron, "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>.

9.2. Informative References

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC4931] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", RFC 4931, May 2007.

Appendix A. Changes from RFC 4931

1. Changed "This document obsoletes RFC 3731" to "This document obsoletes RFC 4931".

2. Replaced references to RFC 3731 with references to 4931.

3. Replaced references to RFC 4930 with references to 5730.

4. Replaced references to RFC 4932 with references to 5732.

5. Replaced references to RFC 4933 with references to 5733.

6. Updated description of inactive status in Section 2.3.

7. Fixed example host names in the Section 1.1 and Section 3.2.1 examples.

8. Changed "but such methods SHOULD NOT be used" to "but such methods should not be used" in Section 2.7.

9. Added "Other notification methods MAY be used in addition to the required service message" in Section 3.2.

10. Added 2201 response code text in Section 3.2.

11. Added BSD license text to XML schema section.

Author's Address

Scott Hollenbeck
VeriSign, Inc.
21345 Ridgetop Circle
Dulles, VA 20166-6503
US

EMail: shollenbeck@verisign.com

==

Network Working Group
Request for Comments: 5732
STD: 69
Obsoletes: 4932
Category: Standards Track

S. Hollenbeck
VeriSign, Inc.
August 2009

Extensible Provisioning Protocol (EPP) Host Mapping

Abstract

This document describes an Extensible Provisioning Protocol (EPP) mapping for the provisioning and management of Internet host names stored in a shared central repository. Specified in XML, the mapping defines EPP command syntax and semantics as applied to host names. This document obsoletes RFC 4932.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes an Internet host name mapping for version 1.0 of the Extensible Provisioning Protocol (EPP). This mapping is specified using the Extensible Markup Language (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]. This document obsoletes RFC 4932 [RFC4932].

[RFC5730] provides a complete description of EPP command and response structures. A thorough understanding of the base protocol specification is necessary to understand the mapping described in this document.

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

1.1. Relationship of Host Objects and Domain Objects

This document assumes that host name objects have a subordinate relationship to a superordinate domain name object. For example, host name "ns1.example.com" has a subordinate relationship to domain name "example.com". EPP actions (such as object transfers) that do not preserve this relationship MUST be explicitly disallowed.

A host name object can be created in a repository for which no superordinate domain name object exists. For example, host name "ns1.example.com" can be created in the ".example" repository so that DNS domains in ".example" can be delegated to the host. Such hosts are described as "external" hosts in this specification since the name of the host does not belong to the namespace of the repository in which the host is being used for delegation purposes.

Whether a host is external or internal relates to the repository in which the host is being used for delegation purposes. An internal host is subordinate if the name of the host belongs to the domain within the repository in which the host is being used for delegation purposes. For example, host ns1.example1.com is a subordinate host of domain example1.com, but it is not a subordinate host of domain example2.com. ns1.example1.com can be used as a name server for example2.com. In this case, ns1.example1.com MUST be treated as an internal host, subject to the rules governing operations on subordinate hosts within the same repository.

1.2. 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 protocol.

2. Object Attributes

An EPP host 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.

2.1. Host Names

The syntax for host names described in this document MUST conform to [RFC0952] as updated by [RFC1123]. At the time of this writing, RFC 3490 [RFC3490] describes a standard to use certain ASCII name labels to represent non-ASCII name labels. These conformance requirements might change in the future as a result of progressing work in developing standards for internationalized host names.

2.2. Client Identifiers

All EPP clients are identified by a server-unique identifier. Client identifiers conform to the "clIDType" syntax described in [RFC5730].

2.3. Status Values

A host object MUST always have at least one associated status value. Status values MAY be set only by the client that sponsors a host object and by the server on which the object resides. A client can change the status of a host object using the EPP <update> command. Each status value MAY be accompanied by a string of human-readable text that describes the rationale for the status applied to the object.

A client MUST NOT alter status values set by the server. A server MAY alter or override status values set by a client, subject to local server policies. The status of an object MAY change as a result of either a client-initiated transform command or an action performed by a server operator.

Status values that can be added or removed by a client are prefixed with "client". Corresponding status values that can be added or removed by a server are prefixed with "server". Status values that do not begin with either "client" or "server" are server-managed.

Status Value Descriptions:

- clientDeleteProhibited, serverDeleteProhibited

  Requests to delete the object MUST be rejected.

- clientUpdateProhibited, serverUpdateProhibited

  Requests to update the object (other than to remove this status) MUST be rejected.

- linked

  The host object has at least one active association with another object, such as a domain object. Servers SHOULD provide services to determine existing object associations.

- ok

  This is the normal status value for an object that has no pending operations or prohibitions. This value is set and removed by the server as other status values are added or removed.

- pendingCreate, pendingDelete, pendingTransfer, pendingUpdate

  A transform command has been processed for the object (or in the case of a <transfer> command, for the host object’s superordinate domain object), but the action has not been completed by the server. Server operators can delay action completion for a variety of reasons, such as to allow for human review or third-party action. A transform command that is processed, but whose requested action is pending, is noted with response code 1001.

When the requested action has been completed, the pendingCreate, pendingDelete, pendingTransfer, or pendingUpdate status value MUST be removed. 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.

"ok" status MAY only be combined with "linked" status.

"linked" status MAY be combined with any status.

"pendingDelete" status MUST NOT be combined with either "clientDeleteProhibited" or "serverDeleteProhibited" status.

"pendingUpdate" status MUST NOT be combined with either "clientUpdateProhibited" or "serverUpdateProhibited" status.

The pendingCreate, pendingDelete, pendingTransfer, and pendingUpdate status values MUST NOT be combined with each other.

Other status combinations not expressly prohibited MAY be used.

2.4. Dates and Times

Date and time attribute values MUST be represented in Universal Coordinated Time (UTC) using the Gregorian calendar. The extended date-time form using upper case "T" and "Z" characters defined in [W3C.REC-xmlschema-2-20041028] MUST be used to represent date-time values, as XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

2.5. IP Addresses

The syntax for IPv4 addresses described in this document MUST conform to [RFC0791]. The syntax for IPv6 addresses described in this document MUST conform to [RFC4291]. Practical considerations for publishing IPv6 address information in zone files are documented in [RFC2874] and [RFC3596]. A server MAY reject IP addresses that have not been allocated for public use by IANA. When a host object is provisioned for use as a DNS name server, IP addresses SHOULD be required only as needed to generate DNS glue records.

3. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in [RFC5730]. The command mappings described here are specifically for use in provisioning and managing Internet host names via EPP.

3.1. EPP Query Commands

EPP provides two commands to retrieve host information: <check> to determine if a host object can be provisioned within a repository, and <info> to retrieve detailed information associated with a host object.

3.1.1. 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 <host:check> element that identifies the host namespace. The <host:check> element contains the following child elements:

- One or more <host:name> elements that contain the fully qualified names of the host 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:      <host:check
C:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
C:        <host:name>ns1.example.com</host:name>
C:        <host:name>ns2.example.com</host:name>
C:        <host:name>ns3.example.com</host:name>
C:      </host: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 <host:chkData> element that identifies the host namespace. The <host:chkData> element contains one or more <host:cd> elements that contain the following child elements:

- A <host:name> element that contains the fully qualified name of the queried host 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.

- An OPTIONAL <host: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:      <host:chkData
S:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
S:        <host:cd>
S:          <host:name avail="1">ns1.example.com</host:name>
S:        </host:cd>
S:        <host:cd>
S:          <host:name avail="0">ns2.example2.com</host:name>
S:          <host:reason>In use</host:reason>
S:        </host:cd>
S:        <host:cd>
S:          <host:name avail="1">ns3.example3.com</host:name>
S:        </host:cd>
S:      </host: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.

3.1.2. EPP <info> Command

The EPP <info> command is used to retrieve information associated with a host object. In addition to the standard EPP command elements, the <info> command MUST contain a <host:info> element that identifies the host namespace. The <host:info> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object for which information is requested.

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:      <host:info
C:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
C:        <host:name>ns1.example.com</host:name>
C:      </host: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 <host:infData> element that identifies the host namespace. The <host:infData> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object.

- A <host:roid> element that contains the Repository Object IDentifier assigned to the host object when the object was created.

- One or more <host:status> elements that describe the status of the host object.

- Zero or more <host:addr> elements that contain the IP addresses associated with the host object.

- A <host:clID> element that contains the identifier of the sponsoring client.

- A <host:crID> element that contains the identifier of the client that created the host object.

- A <host:crDate> element that contains the date and time of host-object creation.

- A <host:upID> element that contains the identifier of the client that last updated the host object. This element MUST NOT be present if the host object has never been modified.

- A <host:upDate> element that contains the date and time of the most recent host-object modification. This element MUST NOT be present if the host object has never been modified.

- A <host:trDate> element that contains the date and time of the most recent successful host-object transfer. This element MUST NOT be provided if the host object has never been transferred. Note that host objects MUST NOT be transferred directly; host objects MUST be transferred implicitly when the host object's superordinate domain object is transferred. Host objects that are subject to transfer when transferring a domain object are listed in the response to an EPP <info> command performed on the domain object.

Example <info> 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:      <host:infData
S:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
S:        <host:name>ns1.example.com</host:name>
S:        <host:roid>NS1_EXAMPLE1-REP</host:roid>
S:        <host:status s="linked"/>
S:        <host:status s="clientUpdateProhibited"/>
S:        <host:addr ip="v4">192.0.2.2</host:addr>
S:        <host:addr ip="v4">192.0.2.29</host:addr>
S:        <host:addr ip="v6">1080:0:0:0:8:800:200C:417A</host:addr>
S:        <host:clID>ClientY</host:clID>
S:        <host:crID>ClientX</host:crID>
S:        <host:crDate>1999-04-03T22:00:00.0Z</host:crDate>
```

```
   S:          <host:upID>ClientX</host:upID>
   S:          <host:upDate>1999-12-03T09:00:00.0Z</host:upDate>
   S:          <host:trDate>2000-04-08T09:00:00.0Z</host:trDate>
   S:        </host: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.

3.1.3. EPP <transfer> Query Command

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

3.2. EPP Transform Commands

EPP provides three commands to transform host objects: <create> to create an instance of a host object, <delete> to delete an instance of a host object, and <update> to change information associated with a host object. This document does not define host-object mappings for the EPP <renew> and <transfer> commands.

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.

Server operators SHOULD confirm that a client is authorized to perform a transform command on a given object. Any attempt to transform an object by an unauthorized client MUST be rejected, and the server MUST return a 2201 response code to the client to note that the client lacks privileges to execute the requested command.

3.2.1. EPP <create> Command

The EPP <create> command provides a transform operation that allows a client to create a host object. In addition to the standard EPP command elements, the <create> command MUST contain a <host:create> element that identifies the host namespace. The <host:create> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object to be created.

- Zero or more <host:addr> elements that contain the IP addresses to be associated with the host. Each element MAY contain an "ip" attribute to identify the IP address format. Attribute value "v4" is used to note IPv4 address format. Attribute value "v6" is used to note IPv6 address format. If the "ip" attribute is not specified, "v4" is the default attribute value.

Hosts can be provisioned for use as name servers in the Domain Name System (DNS), described in [RFC1034] and [RFC1035]. Hosts provisioned as name servers might be subject to server-operator policies that require or prohibit specification of IP addresses, depending on the name of the host and the namespace in which the server will be used as a name server. When provisioned for use as a name server, IP addresses are REQUIRED only as needed to produce DNS glue records. For example, if the server is authoritative for the "com" namespace and the name of the server is "ns1.example.net", the server is not required to produce DNS glue records for the name server, and IP addresses for the server are not required by the DNS.

If the host name exists in a namespace for which the server is authoritative, then the superordinate domain of the host MUST be known to the server before the host object can be created.

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:      <host:create
C:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
C:        <host:name>ns1.example.com</host:name>
C:        <host:addr ip="v4">192.0.2.2</host:addr>
C:        <host:addr ip="v4">192.0.2.29</host:addr>
C:        <host:addr ip="v6">1080:0:0:0:8:800:200C:417A</host:addr>
C:      </host: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 <host:creData> element that identifies the host namespace. The <host:creData> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object.

- A <host:crDate> element that contains the date and time of host-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:      <host:creData
   S:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
   S:        <host:name>ns1.example.com</host:name>
   S:        <host:crDate>1999-04-03T22:00:00.0Z</host:crDate>
   S:      </host:creData>
   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 <create> command cannot be processed for any reason.

3.2.2. EPP <delete> Command

The EPP <delete> command provides a transform operation that allows a client to delete a host object. In addition to the standard EPP command elements, the <delete> command MUST contain a <host:delete> element that identifies the host namespace. The <host:delete> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object to be deleted.

A host name object SHOULD NOT be deleted if the host object is associated with any other object. For example, if the host object is associated with a domain object, the host object SHOULD NOT be deleted until the existing association has been broken. Deleting a host object without first breaking existing associations can cause DNS resolution failure for domain objects that refer to the deleted host object.

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:      <host:delete
C:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
C:        <host:name>ns1.example.com</host:name>
C:      </host: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.

3.2.3. EPP <renew> Command

Renewal semantics do not apply to host objects, so there is no mapping defined for the EPP <renew> command.

3.2.4. EPP <transfer> Command

Transfer semantics do not directly apply to host objects, so there is no mapping defined for the EPP <transfer> command. Host objects are subordinate to an existing superordinate domain object and, as such, they are subject to transfer when a domain object is transferred.

3.2.5. EPP <update> Command

The EPP <update> command provides a transform operation that allows a client to modify the attributes of a host object. In addition to the standard EPP command elements, the <update> command MUST contain a <host:update> element that identifies the host namespace. The <host:update> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object to be updated.

- An OPTIONAL <host:add> element that contains attribute values to be added to the object.

- An OPTIONAL <host:rem> element that contains attribute values to be removed from the object.

- An OPTIONAL <host:chg> element that contains object attribute values to be changed.

At least one <host:add>, <host:rem>, or <host:chg> 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 <host:add> and <host:rem> elements contain the following child elements:

- One or more <host:addr> elements that contain IP addresses to be associated with or removed from the host object. IP address restrictions described in the <create> command mapping apply here as well.

- One or more <host:status> elements that contain status values to be associated with or removed from the object. When specifying a value to be removed, only the attribute value is significant; element text is not required to match a value for removal.

A <host:chg> element contains the following child elements:

- A <host:name> element that contains a new fully qualified host name by which the host object will be known.

Host name changes MAY require the addition or removal of IP addresses to be accepted by the server. IP address association MAY be subject to server policies for provisioning hosts as name servers.

Host name changes can have an impact on associated objects that refer to the host object. A host name change SHOULD NOT require additional updates of associated objects to preserve existing associations, with one exception: changing an external host object that has associations with objects that are sponsored by a different client. Attempts to update such hosts directly MUST fail with EPP error code 2305. The change can be provisioned by creating a new external host with a new name and any needed new attributes, and subsequently updating the other objects sponsored by the client.

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:      <host:update
C:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
C:        <host:name>ns1.example.com</host:name>
C:        <host:add>
C:          <host:addr ip="v4">192.0.2.22</host:addr>
C:          <host:status s="clientUpdateProhibited"/>
C:        </host:add>
C:        <host:rem>
C:          <host:addr ip="v6">1080:0:0:0:8:800:200C:417A</host:addr>
C:        </host:rem>
C:        <host:chg>
C:          <host:name>ns2.example.com</host:name>
C:        </host:chg>
C:      </host: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 could not be processed for any reason.

3.3. Offline Review of Requested Actions

Commands are processed by a server in the order they are received from a client. Though an immediate response confirming receipt and processing of the command is produced by the server, a server operator MAY perform an offline review of requested transform commands before completing the requested action. In such situations, the response from the server MUST clearly note that the transform command has been received and processed, but the requested action is pending. The status of the corresponding object MUST clearly reflect processing of the pending action. The server MUST notify the client when offline processing of the action has been completed.

Examples describing a <create> command that requires offline review are included here. Note the result code and message returned in response to the <create> command.

```
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:      <host:creData
S:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
S:        <host:name>ns1.example.com</host:name>
S:        <host:crDate>1999-04-03T22:00:00.0Z</host:crDate>
S:      </host:creData>
S:    </resData>
```

```
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54322-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

The status of the host object after returning this response MUST include "pendingCreate". The server operator reviews the request offline and informs the client of the outcome of the review either by queuing a service message for retrieval via the <poll> command or by using an out-of-band mechanism to inform the client of the request.

The service message MUST contain text that describes the notification in the child <msg> element of the response <msgQ> element. In addition, the EPP <resData> element MUST contain a child <host:panData> element that identifies the host namespace. The <host:panData> element contains the following child elements:

- A <host:name> element that contains the fully qualified name of the host object. The <host:name> element contains a REQUIRED "paResult" attribute. A positive boolean value indicates that the request has been approved and completed. A negative boolean value indicates that the request has been denied and the requested action has not been taken.

- A <host:paTRID> element that contains the client transaction identifier and server transaction identifier returned with the original response to process the command. The client transaction identifier is OPTIONAL and will only be returned if the client provided an identifier with the original <create> command.

- A <host:paDate> element that contains the date and time describing when review of the requested action was completed.

Example "review completed" service message:

```
   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>1999-04-04T22:01:00.0Z</qDate>
   S:      <msg>Pending action completed successfully.</msg>
   S:    </msgQ>
   S:    <resData>
   S:      <host:panData
   S:       xmlns:host="urn:ietf:params:xml:ns:host-1.0">
   S:        <host:name paResult="1">ns1.example.com</host:name>
   S:        <host:paTRID>
   S:          <clTRID>ABC-12345</clTRID>
   S:          <svTRID>54322-XYZ</svTRID>
   S:        </host:paTRID>
   S:        <host:paDate>1999-04-04T22:00:00.0Z</host:paDate>
   S:      </host:panData>
   S:    </resData>
   S:    <trID>
   S:      <clTRID>BCD-23456</clTRID>
   S:      <svTRID>65432-WXY</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

4. 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:host-1.0"
       xmlns:host="urn:ietf:params:xml:ns:host-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"/>
 <import namespace="urn:ietf:params:xml:ns:epp-1.0"/>

 <annotation>
   <documentation>
     Extensible Provisioning Protocol v1.0
     host provisioning schema.
   </documentation>
 </annotation>

<!--
Child elements found in EPP commands.
-->
 <element name="check" type="host:mNameType"/>
 <element name="create" type="host:createType"/>
 <element name="delete" type="host:sNameType"/>
 <element name="info" type="host:sNameType"/>
 <element name="update" type="host:updateType"/>

<!--
Child elements of the <create> command.
-->
 <complexType name="createType">
```

```
        <sequence>
          <element name="name" type="eppcom:labelType"/>
          <element name="addr" type="host:addrType"
           minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </complexType>

      <complexType name="addrType">
        <simpleContent>
          <extension base="host:addrStringType">
            <attribute name="ip" type="host:ipType"
             default="v4"/>
          </extension>
        </simpleContent>
      </complexType>

      <simpleType name="addrStringType">
        <restriction base="token">
          <minLength value="3"/>
          <maxLength value="45"/>
        </restriction>
      </simpleType>

      <simpleType name="ipType">
        <restriction base="token">
          <enumeration value="v4"/>
          <enumeration value="v6"/>
        </restriction>
      </simpleType>

     <!--
     Child elements of the <delete> and <info> commands.
     -->
      <complexType name="sNameType">
        <sequence>
          <element name="name" type="eppcom:labelType"/>
        </sequence>
      </complexType>

     <!--
     Child element of commands that accept multiple names.
     -->
      <complexType name="mNameType">
        <sequence>
          <element name="name" type="eppcom:labelType"
           maxOccurs="unbounded"/>
        </sequence>
      </complexType>
```

```
   <!--
   Child elements of the <update> command.
   -->
    <complexType name="updateType">
      <sequence>
        <element name="name" type="eppcom:labelType"/>
        <element name="add" type="host:addRemType"
         minOccurs="0"/>
        <element name="rem" type="host:addRemType"
         minOccurs="0"/>
        <element name="chg" type="host:chgType"
         minOccurs="0"/>
      </sequence>
    </complexType>

   <!--
   Data elements that can be added or removed.
   -->
    <complexType name="addRemType">
      <sequence>
        <element name="addr" type="host:addrType"
         minOccurs="0" maxOccurs="unbounded"/>
        <element name="status" type="host:statusType"
         minOccurs="0" maxOccurs="7"/>
      </sequence>
    </complexType>

   <!--
   Data elements that can be changed.
   -->
    <complexType name="chgType">
      <sequence>
        <element name="name" type="eppcom:labelType"/>
      </sequence>
    </complexType>

   <!--
   Child response elements.
   -->
    <element name="chkData" type="host:chkDataType"/>
    <element name="creData" type="host:creDataType"/>
    <element name="infData" type="host:infDataType"/>
    <element name="panData" type="host:panDataType"/>

   <!--
   <check> response elements.
   -->
    <complexType name="chkDataType">
```

```
      <sequence>
        <element name="cd" type="host:checkType"
         maxOccurs="unbounded"/>
      </sequence>
    </complexType>

    <complexType name="checkType">
      <sequence>
        <element name="name" type="host:checkNameType"/>
        <element name="reason" type="eppcom:reasonType"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="checkNameType">
      <simpleContent>
        <extension base="eppcom:labelType">
          <attribute name="avail" type="boolean"
           use="required"/>
        </extension>
      </simpleContent>
    </complexType>

   <!--
   <create> response elements.
   -->
    <complexType name="creDataType">
      <sequence>
        <element name="name" type="eppcom:labelType"/>
        <element name="crDate" type="dateTime"/>
      </sequence>
    </complexType>

   <!--
   <info> response elements.
   -->
    <complexType name="infDataType">
      <sequence>
        <element name="name" type="eppcom:labelType"/>
        <element name="roid" type="eppcom:roidType"/>
        <element name="status" type="host:statusType"
         maxOccurs="7"/>
        <element name="addr" type="host:addrType"
         minOccurs="0" maxOccurs="unbounded"/>
        <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="trDate" type="dateTime"
         minOccurs="0"/>
      </sequence>
    </complexType>

   <!--
   Status is a combination of attributes and an optional human-readable
   message that may be expressed in languages other than English.
   -->
    <complexType name="statusType">
      <simpleContent>
        <extension base="normalizedString">
          <attribute name="s" type="host:statusValueType"
           use="required"/>
          <attribute name="lang" type="language"
           default="en"/>
        </extension>
      </simpleContent>
    </complexType>

    <simpleType name="statusValueType">
      <restriction base="token">
        <enumeration value="clientDeleteProhibited"/>
        <enumeration value="clientUpdateProhibited"/>
        <enumeration value="linked"/>
        <enumeration value="ok"/>
        <enumeration value="pendingCreate"/>
        <enumeration value="pendingDelete"/>
        <enumeration value="pendingTransfer"/>
        <enumeration value="pendingUpdate"/>
        <enumeration value="serverDeleteProhibited"/>
        <enumeration value="serverUpdateProhibited"/>
      </restriction>
    </simpleType>

   <!--
   Pending action notification response elements.
   -->
    <complexType name="panDataType">
      <sequence>
        <element name="name" type="host:paNameType"/>
        <element name="paTRID" type="epp:trIDType"/>
        <element name="paDate" type="dateTime"/>
      </sequence>
    </complexType>
```

```
     <complexType name="paNameType">
       <simpleContent>
         <extension base="eppcom:labelType">
           <attribute name="paResult" type="boolean"
            use="required"/>
         </extension>
       </simpleContent>
     </complexType>

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

5. 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 [RFC2781]. 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 in environments where parser encoding support incompatibility exists.

All date-time values presented via EPP MUST be expressed in Universal Coordinated Time using the Gregorian calendar. XML Schema allows use of time zone identifiers to indicate offsets from the zero meridian, but this option MUST NOT be used with EPP. The extended date-time form using upper case "T" and "Z" characters defined in [W3C.REC-xmlschema-2-20041028] MUST be used to represent date-time values, as XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

The syntax for domain and host names described in this document MUST conform to [RFC0952] and [RFC1123]. At the time of this writing, RFC 3490 [RFC3490] describes a standard to use certain ASCII name labels to represent non-ASCII name labels. These conformance requirements might change as a result of progressing work in developing standards for internationalized host names.

6. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. Two URI assignments have been registered by the IANA.

Registration request for the host namespace:

URI: urn:ietf:params:xml:ns:host-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 host XML schema:

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

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

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

7. Security Considerations

The object mapping described in this document does not provide any security services or introduce any additional considerations beyond those described by [RFC5730] or those caused by the protocol layers used by EPP.

8. Acknowledgements

RFC 3732 is a product of the PROVREG working group, which suggested improvements and provided many invaluable comments. The author wishes to acknowledge the efforts of WG chairs Edward Lewis and Jaap Akkerhuis for their process and editorial contributions. RFC 4932 and this document are individual submissions, based on the work done in RFC 3732.

Specific suggestions that have been incorporated into this document were provided by Chris Bason, Jordyn Buchanan, Dave Crocker, Anthony Eden, Sheer El-Showk, Klaus Malorny, Dan Manley, Michael Mealling, Patrick Mevzek, and Rick Wesson.

9. References

9.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[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.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

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

[W3C.REC-xml-20040204]
Sperberg-McQueen, C., Maler, E., Yergeau, F., Paoli, J., and T. Bray, "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]
Maloney, M., Thompson, H., Mendelsohn, N., and D. Beech, "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]
Malhotra, A. and P. Biron, "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>.

9.2. Informative References

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC2874] Crawford, M. and C. Huitema, "DNS Extensions to Support IPv6 Address Aggregation and Renumbering", RFC 2874, July 2000.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.

[RFC4932] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Host Mapping", RFC 4932, May 2007.

Appendix A. Changes from RFC 4932

1. Changed "This document obsoletes RFC 3732" to "This document obsoletes RFC 4932".

2. Replaced references to RFC 1886 with references to 3596.

3. Removed references to RFC 3152 since both it and 1886 have been obsoleted by 3596.

4. Replaced references to RFC 3732 with references to 4932.

5. Replaced references to RFC 4930 with references to 5730.

6. Added "Other notification methods MAY be used in addition to the required service message" in Section 3.2.

7. Added 2201 response code text in Section 3.2.

8. Added BSD license text to XML schema section.

Author's Address

Scott Hollenbeck
VeriSign, Inc.
21345 Ridgetop Circle
Dulles, VA 20166-6503
US

EMail: shollenbeck@verisign.com

==

Network Working Group   S. Hollenbeck
Request for Comments: 5733   VeriSign, Inc.
STD: 69   August 2009
Obsoletes: 4933
Category: Standards Track

Extensible Provisioning Protocol (EPP) Contact Mapping

Abstract

This document describes an Extensible Provisioning Protocol (EPP) mapping for the provisioning and management of individual or organizational social information identifiers (known as "contacts") stored in a shared central repository. Specified in Extensible Markup Language (XML), the mapping defines EPP command syntax and semantics as applied to contacts. This document obsoletes RFC 4933.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes a personal and organizational identifier mapping for version 1.0 of the Extensible Provisioning Protocol (EPP). This mapping is specified using the Extensible Markup Language (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]. This document obsoletes RFC 4933 [RFC4933].

[RFC5730] provides a complete description of EPP command and response structures. A thorough understanding of the base protocol specification is necessary to understand the mapping described in this document.

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

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 [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 protocol.

2. Object Attributes

An EPP contact 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.

2.1. Contact and Client Identifiers

All EPP contacts are identified by a server-unique identifier. Contact identifiers are character strings with a specified minimum length, a specified maximum length, and a specified format. Contact identifiers use the "clIDType" client identifier syntax described in [RFC5730].

2.2. Status Values

A contact object MUST always have at least one associated status value. Status values can be set only by the client that sponsors a contact object and by the server on which the object resides. A client can change the status of a contact object using the EPP <update> command. Each status value MAY be accompanied by a string of human-readable text that describes the rationale for the status applied to the object.

A client MUST NOT alter status values set by the server. A server MAY alter or override status values set by a client, subject to local server policies. The status of an object MAY change as a result of either a client-initiated transform command or an action performed by a server operator.

Status values that can be added or removed by a client are prefixed with "client". Corresponding status values that can be added or removed by a server are prefixed with "server". Status values that do not begin with either "client" or "server" are server-managed.

Status Value Descriptions:

- clientDeleteProhibited, serverDeleteProhibited

  Requests to delete the object MUST be rejected.

- clientTransferProhibited, serverTransferProhibited

  Requests to transfer the object MUST be rejected.

- clientUpdateProhibited, serverUpdateProhibited

  Requests to update the object (other than to remove this status) MUST be rejected.

- linked

  The contact object has at least one active association with another object, such as a domain object. Servers SHOULD provide services to determine existing object associations.

- ok

  This is the normal status value for an object that has no pending operations or prohibitions. This value is set and removed by the server as other status values are added or removed.

- pendingCreate, pendingDelete, pendingTransfer, pendingUpdate

  A transform command has been processed for the object, but the action has not been completed by the server. Server operators can delay action completion for a variety of reasons, such as to allow for human review or third-party action. A transform command that is processed, but whose requested action is pending, is noted with response code 1001.

When the requested action has been completed, the pendingCreate, pendingDelete, pendingTransfer, or pendingUpdate status value MUST be removed. 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.

"ok" status MAY only be combined with "linked" status.

"linked" status MAY be combined with any status.

"pendingDelete" status MUST NOT be combined with either "clientDeleteProhibited" or "serverDeleteProhibited" status.

"pendingTransfer" status MUST NOT be combined with either "clientTransferProhibited" or "serverTransferProhibited" status.
"pendingUpdate" status MUST NOT be combined with either "clientUpdateProhibited" or "serverUpdateProhibited" status.

The pendingCreate, pendingDelete, pendingTransfer, and pendingUpdate status values MUST NOT be combined with each other.

Other status combinations not expressly prohibited MAY be used.

2.3. Individual and Organizational Names

Individual and organizational names associated with a contact are represented using character strings. These strings have a specified minimum length and a specified maximum length. Individual and organizational names MAY be provided in either UTF-8 [RFC3629] or a subset of UTF-8 that can be represented in 7-bit ASCII, depending on local needs.

2.4. Address

Every contact has associated postal-address information. A postal address contains OPTIONAL street information, city information, OPTIONAL state/province information, an OPTIONAL postal code, and a country identifier. Address information MAY be provided in either UTF-8 or a subset of UTF-8 that can be represented in 7-bit ASCII, depending on local needs.

2.4.1. Street, City, and State or Province

Contact street, city, and state or province information is represented using character strings. These strings have a specified minimum length and a specified maximum length.

2.4.2. Postal Code

Contact postal codes are represented using character strings. These strings have a specified minimum length and a specified maximum length.

2.4.3. Country

Contact country identifiers are represented using two-character identifiers specified in [ISO3166-1].

2.5. Telephone Numbers

Contact telephone number structure is derived from structures defined in [ITU.E164.2005]. Telephone numbers described in this mapping are character strings that MUST begin with a plus sign ("+", ASCII value 0x002B), followed by a country code defined in [ITU.E164.2005], followed by a dot (".", ASCII value 0x002E), followed by a sequence of digits representing the telephone number. An optional "x" attribute is provided to note telephone extension information.

2.6. Email Addresses

Email address syntax is defined in [RFC5322]. This mapping does not prescribe minimum or maximum lengths for character strings used to represent email addresses.

2.7. Dates and Times

Date and time attribute values MUST be represented in Universal Coordinated Time (UTC) using the Gregorian calendar. The extended date-time form using upper case "T" and "Z" characters defined in

[W3C.REC-xmlschema-2-20041028] MUST be used to represent date-time values, as XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

2.8. Authorization Information

Authorization information is associated with contact objects to facilitate transfer operations. Authorization information is assigned when a contact object is created, and it might be updated in the future. This specification describes password-based authorization information, though other mechanisms are possible.

2.9. Disclosure of Data Elements and Attributes

The EPP core protocol requires a server operator to announce data-collection policies to clients; see Section 2.4 of [RFC5730]. In conjunction with this disclosure requirement, this mapping includes data elements that allow a client to identify elements that require exceptional server-operator handling to allow or restrict disclosure to third parties.

A server operator announces a default disclosure policy when establishing a session with a client. When an object is created or updated, the client can specify contact attributes that require exceptional disclosure handling using an OPTIONAL <contact:disclose> element. Once set, disclosure preferences can be reviewed using a contact-information query. A server operator MUST reject any transaction that requests disclosure practices that do not conform to the announced data-collection policy with a 2308 error response code.

If present, the <contact:disclose> element MUST contain a "flag" attribute. The "flag" attribute contains an XML Schema boolean value. A value of "true" or "1" (one) notes a client 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.

The <contact:disclose> element MUST contain at least one of the following child elements:

```
<contact:name type="int"/>
<contact:name type="loc"/>
<contact:org type="int"/>
<contact:org type="loc"/>
<contact:addr type="int"/>
<contact:addr type="loc"/>
```

```
   <contact:voice/>
   <contact:fax/>
   <contact:email/>
```

Example <contact:disclose> element, flag="0":

```
   <contact:disclose flag="0">
    <contact:email/>
    <contact:voice/>
   </contact:disclose>
```

In this example, the contact email address and voice telephone number cannot be disclosed. All other elements are subject to disclosure in accordance with the server’s data-collection policy.

Example <contact:disclose> element, flag="1":

```
   <contact:disclose flag="1">
    <contact:name type="int"/>
    <contact:org type="int"/>
    <contact:addr type="int"/>
   </contact:disclose>
```

In this example, the internationalized contact name, organization, and address information can be disclosed. All other elements are subject to disclosure in accordance with the server’s data-collection policy.

Client-identification features provided by the EPP <login> command and contact-authorization information are used to determine if a client is authorized to perform contact-information query commands. These features also determine if a client is authorized to receive data that is otherwise marked for non-disclosure in a query response.

3. EPP Command Mapping

A detailed description of the EPP syntax and semantics can be found in [RFC5730]. The command mappings described here are specifically for use in provisioning and managing contact objects via EPP.

3.1. EPP Query Commands

EPP provides three commands to retrieve contact information: <check> to determine if a contact object can be provisioned within a repository, <info> to retrieve detailed information associated with a contact object, and <transfer> to retrieve information regarding the transfer status of the contact object.

3.1.1. 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 <contact:check> element that identifies the contact namespace. The <contact:check> element contains the following child elements:

- One or more <contact:id> elements that contain the server-unique identifier of the contact 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:      <contact:check
C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
C:        <contact:id>sh8013</contact:id>
C:        <contact:id>sah8013</contact:id>
C:        <contact:id>8013sah</contact:id>
C:      </contact: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 <contact:chkData> element that identifies the contact namespace. The <contact:chkData> element contains one or more <contact:cd> elements that contain the following child elements:

- A <contact: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.

- An OPTIONAL <contact: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:      <contact:chkData
S:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
S:        <contact:cd>
S:          <contact:id avail="1">sh8013</contact:id>
S:        </contact:cd>
S:        <contact:cd>
S:          <contact:id avail="0">sah8013</contact:id>
S:          <contact:reason>In use</contact:reason>
S:        </contact:cd>
S:        <contact:cd>
S:          <contact:id avail="1">8013sah</contact:id>
S:        </contact:cd>
S:      </contact: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.

3.1.2. EPP <info> Command

The EPP <info> command is used to retrieve information associated with a contact object. In addition to the standard EPP command elements, the <info> command MUST contain a <contact:info> element that identifies the contact namespace. The <contact:info> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier of the contact object to be queried.

- An OPTIONAL <contact:authInfo> element that contains authorization information associated with the contact object. If this element is not provided or if the authorization information is invalid, server policy determines if the command is rejected or if response information will be returned to the client.

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:      <contact:info
C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
C:        <contact:id>sh8013</contact:id>
C:        <contact:authInfo>
C:          <contact:pw>2fooBAR</contact:pw>
C:        </contact:authInfo>
C:      </contact: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 <contact:infData> element that identifies the contact namespace. The <contact:infData> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier of the contact object.

- A <contact:roid> element that contains the Repository Object IDentifier assigned to the contact object when the object was created.

- One or more <contact:status> elements that describe the status of the contact object.

- One or two <contact: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 <contact:postalInfo> element contains the following child elements:

  - A <contact:name> element that contains the name of the individual or role represented by the contact.

  - An OPTIONAL <contact:org> element that contains the name of the organization with which the contact is affiliated.

  - A <contact:addr> element that contains address information associated with the contact. A <contact:addr> element contains the following child elements:

    - One, two, or three OPTIONAL <contact:street> elements that contain the contact’s street address.

    - A <contact:city> element that contains the contact’s city.

    - An OPTIONAL <contact:sp> element that contains the contact’s state or province.

    - An OPTIONAL <contact:pc> element that contains the contact’s postal code.

    - A <contact:cc> element that contains the contact’s country code.

- An OPTIONAL <contact:voice> element that contains the contact’s voice telephone number.

- An OPTIONAL <contact:fax> element that contains the contact’s facsimile telephone number.

- A <contact:email> element that contains the contact’s email address.

- A <contact:clID> element that contains the identifier of the sponsoring client.

- A <contact:crID> element that contains the identifier of the client that created the contact object.

- A <contact:crDate> element that contains the date and time of contact-object creation.

- A <contact:upID> element that contains the identifier of the client that last updated the contact object. This element MUST NOT be present if the contact has never been modified.

- A <contact:upDate> element that contains the date and time of the most recent contact-object modification. This element MUST NOT be present if the contact object has never been modified.

- A <contact:trDate> element that contains the date and time of the most recent successful contact-object transfer. This element MUST NOT be provided if the contact object has never been transferred.

- A <contact:authInfo> element that contains authorization information associated with the contact object. This element MUST NOT be provided if the querying client is not the current sponsoring client.

- An OPTIONAL <contact:disclose> element that identifies elements that require exceptional server-operator handling to allow or restrict disclosure to third parties. See Section 2.9 for a description of the child elements contained within the <contact:disclose> element.

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>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <contact:infData
S:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
S:        <contact:id>sh8013</contact:id>
S:        <contact:roid>SH8013-REP</contact:roid>
S:        <contact:status s="linked"/>
S:        <contact:status s="clientDeleteProhibited"/>
S:        <contact:postalInfo type="int">
```

```
   S:            <contact:name>John Doe</contact:name>
   S:            <contact:org>Example Inc.</contact:org>
   S:            <contact:addr>
   S:              <contact:street>123 Example Dr.</contact:street>
   S:              <contact:street>Suite 100</contact:street>
   S:              <contact:city>Dulles</contact:city>
   S:              <contact:sp>VA</contact:sp>
   S:              <contact:pc>20166-6503</contact:pc>
   S:              <contact:cc>US</contact:cc>
   S:            </contact:addr>
   S:          </contact:postalInfo>
   S:          <contact:voice x="1234">+1.7035555555</contact:voice>
   S:          <contact:fax>+1.7035555556</contact:fax>
   S:          <contact:email>jdoe@example.com</contact:email>
   S:          <contact:clID>ClientY</contact:clID>
   S:          <contact:crID>ClientX</contact:crID>
   S:          <contact:crDate>1999-04-03T22:00:00.0Z</contact:crDate>
   S:          <contact:upID>ClientX</contact:upID>
   S:          <contact:upDate>1999-12-03T09:00:00.0Z</contact:upDate>
   S:          <contact:trDate>2000-04-08T09:00:00.0Z</contact:trDate>
   S:          <contact:authInfo>
   S:            <contact:pw>2fooBAR</contact:pw>
   S:          </contact:authInfo>
   S:          <contact:disclose flag="0">
   S:            <contact:voice/>
   S:            <contact:email/>
   S:          </contact:disclose>
   S:        </contact: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.

3.1.3. EPP <transfer> Query Command

The EPP <transfer> command provides a query operation that allows a client to determine the real-time status of pending and completed transfer requests. In addition to the standard EPP command elements, the <transfer> command MUST contain an "op" attribute with value "query", and a <contact:transfer> element that identifies the contact namespace. The <contact:transfer> element MUST contain the following child elements:

- A <contact:id> element that contains the server-unique identifier of the contact object to be queried.

- An OPTIONAL <contact:authInfo> element that contains authorization information associated with the contact object. If this element is not provided or if the authorization information is invalid, server policy determines if the command is rejected or if response information will be returned to the client.

Example <transfer> query 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="query">
C:      <contact:transfer
C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
C:        <contact:id>sh8013</contact:id>
C:        <contact:authInfo>
C:          <contact:pw>2fooBAR</contact:pw>
C:        </contact:authInfo>
C:      </contact:transfer>
C:    </transfer>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <transfer> query command has been processed successfully, the EPP <resData> element MUST contain a child <contact:trnData> element that identifies the contact namespace. The <contact:trnData> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier for the queried contact.

- A <contact:trStatus> element that contains the state of the most recent transfer request.

- A <contact:reID> element that contains the identifier of the client that requested the object transfer.

- A <contact:reDate> element that contains the date and time that the transfer was requested.

- A <contact:acID> element that contains the identifier of the client that SHOULD act upon a PENDING transfer request. For all other status types, the value identifies the client that took the indicated action.

- A <contact:acDate> element that contains the date and time of a required or completed response. For a pending request, the value identifies the date and time by which a response is required before an automated response action SHOULD be taken by the server. For all other status types, the value identifies the date and time when the request was completed.

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="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <resData>
S:      <contact:trnData
S:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
S:        <contact:id>sh8013</contact:id>
S:        <contact:trStatus>pending</contact:trStatus>
S:        <contact:reID>ClientX</contact:reID>
S:        <contact:reDate>2000-06-06T22:00:00.0Z</contact:reDate>
S:        <contact:acID>ClientY</contact:acID>
S:        <contact:acDate>2000-06-11T22:00:00.0Z</contact:acDate>
S:      </contact:trnData>
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 <transfer> query command cannot be processed for any reason.

3.2. EPP Transform Commands

EPP provides four commands to transform contact-object information: <create> to create an instance of a contact object, <delete> to delete an instance of a contact object, <transfer> to manage contact-object sponsorship changes, and <update> to change information associated with a contact object. This document does not define a mapping for the EPP <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.

Server operators SHOULD confirm that a client is authorized to perform a transform command on a given object. Any attempt to transform an object by an unauthorized client MUST be rejected, and the server MUST return a 2201 response code to the client to note that the client lacks privileges to execute the requested command.

3.2.1. EPP <create> Command

The EPP <create> command provides a transform operation that allows a client to create a contact object. In addition to the standard EPP command elements, the <create> command MUST contain a <contact:create> element that identifies the contact namespace. The <contact:create> element contains the following child elements:

- A <contact:id> element that contains the desired server-unique identifier for the contact to be created.

- One or two <contact: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 <contact:postalInfo> element contains the following child elements:

  o A <contact:name> element that contains the name of the individual or role represented by the contact.

  o An OPTIONAL <contact:org> element that contains the name of the organization with which the contact is affiliated.

      o A <contact:addr> element that contains address information associated with the contact. A <contact:addr> element contains the following child elements:

         * One, two, or three OPTIONAL <contact:street> elements that contain the contact’s street address.

         * A <contact:city> element that contains the contact’s city.

         * An OPTIONAL <contact:sp> element that contains the contact’s state or province.

         * An OPTIONAL <contact:pc> element that contains the contact’s postal code.

         * A <contact:cc> element that contains the contact’s country code.

   - An OPTIONAL <contact:voice> element that contains the contact’s voice telephone number.

   - An OPTIONAL <contact:fax> element that contains the contact’s facsimile telephone number.

   - A <contact:email> element that contains the contact’s email address.

   - A <contact:authInfo> element that contains authorization information to be associated with the contact object. This mapping includes a password-based authentication mechanism, but the schema allows new mechanisms to be defined in new schemas.

   - An OPTIONAL <contact:disclose> element that allows a client to identify elements that require exceptional server-operator handling to allow or restrict disclosure to third parties. See Section 2.9 for a description of the child elements contained within the <contact: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:      <contact:create
   C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
   C:        <contact:id>sh8013</contact:id>
   C:        <contact:postalInfo type="int">
   C:            <contact:name>John Doe</contact:name>
   C:            <contact:org>Example Inc.</contact:org>
   C:            <contact:addr>
   C:              <contact:street>123 Example Dr.</contact:street>
   C:              <contact:street>Suite 100</contact:street>
   C:              <contact:city>Dulles</contact:city>
   C:              <contact:sp>VA</contact:sp>
   C:              <contact:pc>20166-6503</contact:pc>
   C:              <contact:cc>US</contact:cc>
   C:            </contact:addr>
   C:          </contact:postalInfo>
   C:          <contact:voice x="1234">+1.7035555555</contact:voice>
   C:          <contact:fax>+1.7035555556</contact:fax>
   C:          <contact:email>jdoe@example.com</contact:email>
   C:          <contact:authInfo>
   C:            <contact:pw>2fooBAR</contact:pw>
   C:          </contact:authInfo>
   C:          <contact:disclose flag="0">
   C:            <contact:voice/>
   C:            <contact:email/>
   C:          </contact:disclose>
   C:        </contact: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 <contact:creData> element that identifies the contact namespace. The <contact:creData> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier for the created contact.

- A <contact:crDate> element that contains the date and time of contact-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:      <contact:creData
   S:         xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
   S:          <contact:id>sh8013</contact:id>
   S:          <contact:crDate>1999-04-03T22:00:00.0Z</contact:crDate>
   S:        </contact: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.

3.2.2. EPP <delete> Command

The EPP <delete> command provides a transform operation that allows a client to delete a contact object. In addition to the standard EPP command elements, the <delete> command MUST contain a <contact:delete> element that identifies the contact namespace. The <contact:delete> element MUST contain the following child element:

- A <contact:id> element that contains the server-unique identifier of the contact object to be deleted.

A contact object SHOULD NOT be deleted if it is associated with other known objects. An associated contact 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:      <contact:delete
   C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
   C:        <contact:id>sh8013</contact:id>
   C:      </contact: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.

3.2.3. EPP <renew> Command

Renewal semantics do not apply to contact objects, so there is no mapping defined for the EPP <renew> command.

3.2.4. EPP <transfer> Command

The EPP <transfer> command provides a transform operation that allows a client to manage requests to transfer the sponsorship of a contact object. In addition to the standard EPP command elements, the <transfer> command MUST contain a <contact:transfer> element that identifies the contact namespace. The <contact:transfer> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier of the contact object for which a transfer request is to be created, approved, rejected, or cancelled.

- A <contact:authInfo> element that contains authorization information associated with the contact object.

Every EPP <transfer> command MUST contain an "op" attribute that identifies the transfer operation to be performed, as defined in [RFC5730].

Example <transfer> request 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:      <contact:transfer
C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
C:        <contact:id>sh8013</contact:id>
C:        <contact:authInfo>
C:          <contact:pw>2fooBAR</contact:pw>
C:        </contact:authInfo>
C:      </contact:transfer>
C:    </transfer>
C:    <clTRID>ABC-12345</clTRID>
C:  </command>
C:</epp>
```

When a <transfer> command has been processed successfully, the EPP <resData> element MUST contain a child <contact:trnData> element that identifies the contact namespace. The <contact:trnData> element contains the same child elements defined for a <transfer> query response.

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:      <contact:trnData
S:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
S:        <contact:id>sh8013</contact:id>
S:        <contact:trStatus>pending</contact:trStatus>
S:        <contact:reID>ClientX</contact:reID>
S:        <contact:reDate>2000-06-08T22:00:00.0Z</contact:reDate>
S:        <contact:acID>ClientY</contact:acID>
S:        <contact:acDate>2000-06-13T22:00:00.0Z</contact:acDate>
S:      </contact:trnData>
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 <transfer> command cannot be processed for any reason.

3.2.5. EPP <update> Command

The EPP <update> command provides a transform operation that allows a client to modify the attributes of a contact object. In addition to the standard EPP command elements, the <update> command MUST contain a <contact:update> element that identifies the contact namespace. The <contact:update> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier of the contact object to be updated.

- An OPTIONAL <contact:add> element that contains attribute values to be added to the object.

- An OPTIONAL <contact:rem> element that contains attribute values to be removed from the object.

- An OPTIONAL <contact:chg> element that contains object attribute values to be changed.

At least one <contact:add>, <contact:rem>, or <contact:chg> 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 <contact:add> and <contact:rem> elements contain the following child elements:

- One or more <contact:status> elements that contain status values to be associated with or removed from the object. When specifying a value to be removed, only the attribute value is significant; element text is not required to match a value for removal.

A <contact:chg> element contains the following OPTIONAL child elements. At least one child element MUST be present:

- One or two <contact: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 <contact:postalInfo> element contains the following OPTIONAL child elements:

  - A <contact:name> element that contains the name of the individual or role represented by the contact.

  - A <contact:org> element that contains the name of the organization with which the contact is affiliated.

  - A <contact:addr> element that contains address information associated with the contact. A <contact:addr> element contains the following child elements:

    * One, two, or three OPTIONAL <contact:street> elements that contain the contact’s street address.

    * A <contact:city> element that contains the contact’s city.

    * An OPTIONAL <contact:sp> element that contains the contact’s state or province.

    * An OPTIONAL <contact:pc> element that contains the contact’s postal code.

    * A <contact:cc> element that contains the contact’s country code.

- A <contact:voice> element that contains the contact’s voice telephone number.

- A <contact:fax> element that contains the contact’s facsimile telephone number.

- A <contact:email> element that contains the contact’s email address.

- A <contact:authInfo> element that contains authorization information associated with the contact object. This mapping includes a password-based authentication mechanism, but the schema allows new mechanisms to be defined in new schemas.

- A <contact:disclose> element that allows a client to identify elements that require exceptional server-operator handling to allow or restrict disclosure to third parties. See Section 2.9 for a description of the child elements contained within the <contact: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:      <contact:update
C:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
C:        <contact:id>sh8013</contact:id>
C:        <contact:add>
C:          <contact:status s="clientDeleteProhibited"/>
C:        </contact:add>
C:        <contact:chg>
C:          <contact:postalInfo type="int">
C:            <contact:org/>
C:            <contact:addr>
C:              <contact:street>124 Example Dr.</contact:street>
C:              <contact:street>Suite 200</contact:street>
C:              <contact:city>Dulles</contact:city>
C:              <contact:sp>VA</contact:sp>
C:              <contact:pc>20166-6503</contact:pc>
C:              <contact:cc>US</contact:cc>
C:            </contact:addr>
C:          </contact:postalInfo>
C:          <contact:voice>+1.7034444444</contact:voice>
C:          <contact:fax/>
C:          <contact:authInfo>
C:            <contact:pw>2fooBAR</contact:pw>
C:          </contact:authInfo>
C:          <contact:disclose flag="1">
C:            <contact:voice/>
C:            <contact:email/>
C:          </contact:disclose>
C:        </contact:chg>
C:      </contact: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.

3.3. Offline Review of Requested Actions

Commands are processed by a server in the order they are received from a client. Though an immediate response confirming receipt and processing of the command is produced by the server, a server operator MAY perform an offline review of requested transform commands before completing the requested action. In such situations, the response from the server MUST clearly note that the transform command has been received and processed but that the requested action is pending. The status of the corresponding object MUST clearly reflect processing of the pending action. The server MUST notify the client when offline processing of the action has been completed.

Examples describing a <create> command that requires offline review are included here. Note the result code and message returned in response to the <create> command.

```
   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:      <contact:creData
   S:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
   S:        <contact:id>sh8013</contact:id>
   S:        <contact:crDate>1999-04-03T22:00:00.0Z</contact:crDate>
   S:      </contact:creData>
   S:    </resData>
   S:    <trID>
   S:      <clTRID>ABC-12345</clTRID>
   S:      <svTRID>54321-XYZ</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

The status of the contact object after returning this response MUST include "pendingCreate". The server operator reviews the request offline and informs the client of the outcome of the review either by queuing a service message for retrieval via the <poll> command or by using an out-of-band mechanism to inform the client of the request.

The service message MUST contain text that describes the notification in the child <msg> element of the response <msgQ> element. In addition, the EPP <resData> element MUST contain a child <contact:panData> element that identifies the contact namespace. The <contact:panData> element contains the following child elements:

- A <contact:id> element that contains the server-unique identifier of the contact object. The <contact:id> element contains a REQUIRED "paResult" attribute. A positive boolean value indicates that the request has been approved and completed. A negative boolean value indicates that the request has been denied and the requested action has not been taken.

- A <contact:paTRID> element that contains the client transaction identifier and server transaction identifier returned with the original response to process the command. The client transaction identifier is OPTIONAL and will only be returned if the client provided an identifier with the original <create> command.

- A <contact:paDate> element that contains the date and time describing when review of the requested action was completed.

Example "review completed" service message:

```
   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>1999-04-04T22:01:00.0Z</qDate>
   S:      <msg>Pending action completed successfully.</msg>
   S:    </msgQ>
   S:    <resData>
   S:      <contact:panData
   S:       xmlns:contact="urn:ietf:params:xml:ns:contact-1.0">
   S:        <contact:id paResult="1">sh8013</contact:id>
   S:        <contact:paTRID>
   S:          <clTRID>ABC-12345</clTRID>
   S:          <svTRID>54321-XYZ</svTRID>
   S:        </contact:paTRID>
   S:        <contact:paDate>1999-04-04T22:00:00.0Z</contact:paDate>
   S:      </contact:panData>
   S:    </resData>
   S:    <trID>
   S:      <clTRID>BCD-23456</clTRID>
   S:      <svTRID>65432-WXY</svTRID>
   S:    </trID>
   S:  </response>
   S:</epp>
```

4. 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:contact-1.0"
           xmlns:contact="urn:ietf:params:xml:ns:contact-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"/>
    <import namespace="urn:ietf:params:xml:ns:epp-1.0"/>

    <annotation>
      <documentation>
        Extensible Provisioning Protocol v1.0
        contact provisioning schema.
      </documentation>
    </annotation>

   <!--
   Child elements found in EPP commands.
   -->
    <element name="check" type="contact:mIDType"/>
    <element name="create" type="contact:createType"/>
    <element name="delete" type="contact:sIDType"/>
    <element name="info" type="contact:authIDType"/>
    <element name="transfer" type="contact:authIDType"/>
    <element name="update" type="contact:updateType"/>

   <!--
   Utility types.
   -->
    <simpleType name="ccType">
      <restriction base="token">
        <length value="2"/>
      </restriction>
    </simpleType>
    <complexType name="e164Type">
      <simpleContent>
        <extension base="contact:e164StringType">
          <attribute name="x" type="token"/>
        </extension>
      </simpleContent>
    </complexType>

    <simpleType name="e164StringType">
      <restriction base="token">
        <pattern value="(\+[0-9]{1,3}\.[0-9]{1,14})?"/>
        <maxLength value="17"/>
      </restriction>
    </simpleType>

    <simpleType name="pcType">
      <restriction base="token">
        <maxLength value="16"/>
      </restriction>
    </simpleType>

    <simpleType name="postalLineType">
       <restriction base="normalizedString">
         <minLength value="1"/>
         <maxLength value="255"/>
       </restriction>
    </simpleType>

    <simpleType name="optPostalLineType">
       <restriction base="normalizedString">
         <maxLength value="255"/>
       </restriction>
    </simpleType>

   <!--
   Child elements of the <create> command.
   -->
    <complexType name="createType">
      <sequence>
        <element name="id" type="eppcom:clIDType"/>
        <element name="postalInfo" type="contact: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="authInfo" type="contact:authInfoType"/>
        <element name="disclose" type="contact:discloseType"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="postalInfoType">
      <sequence>
        <element name="name" type="contact:postalLineType"/>
        <element name="org" type="contact:optPostalLineType"
         minOccurs="0"/>
        <element name="addr" type="contact:addrType"/>
      </sequence>
      <attribute name="type" type="contact:postalInfoEnumType"
       use="required"/>
    </complexType>

    <simpleType name="postalInfoEnumType">
      <restriction base="token">
        <enumeration value="loc"/>
        <enumeration value="int"/>
      </restriction>
    </simpleType>

    <complexType name="addrType">
      <sequence>
        <element name="street" type="contact:optPostalLineType"
         minOccurs="0" maxOccurs="3"/>
        <element name="city" type="contact:postalLineType"/>
        <element name="sp" type="contact:optPostalLineType"
         minOccurs="0"/>
        <element name="pc" type="contact:pcType"
         minOccurs="0"/>
        <element name="cc" type="contact:ccType"/>
      </sequence>
    </complexType>

    <complexType name="authInfoType">
      <choice>
        <element name="pw" type="eppcom:pwAuthInfoType"/>
        <element name="ext" type="eppcom:extAuthInfoType"/>
      </choice>
    </complexType>

    <complexType name="discloseType">
      <sequence>
        <element name="name" type="contact:intLocType"
         minOccurs="0" maxOccurs="2"/>
        <element name="org" 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"/>
      </sequence>
      <attribute name="flag" type="boolean" use="required"/>
    </complexType>

    <complexType name="intLocType">
      <attribute name="type" type="contact:postalInfoEnumType"
       use="required"/>
    </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> and <transfer> commands.
   -->
    <complexType name="authIDType">
      <sequence>
        <element name="id" type="eppcom:clIDType"/>
        <element name="authInfo" type="contact:authInfoType"
         minOccurs="0"/>
      </sequence>
    </complexType>

   <!--
   Child elements of the <update> command.
   -->
```

```
    <complexType name="updateType">
      <sequence>
        <element name="id" type="eppcom:clIDType"/>
        <element name="add" type="contact:addRemType"
         minOccurs="0"/>
        <element name="rem" type="contact:addRemType"
         minOccurs="0"/>
        <element name="chg" type="contact:chgType"
         minOccurs="0"/>
      </sequence>
    </complexType>

   <!--
   Data elements that can be added or removed.
   -->
    <complexType name="addRemType">
      <sequence>
        <element name="status" type="contact:statusType"
         maxOccurs="7"/>
      </sequence>
    </complexType>

   <!--
   Data elements that can be changed.
   -->
    <complexType name="chgType">
      <sequence>
        <element name="postalInfo" type="contact: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="authInfo" type="contact:authInfoType"
         minOccurs="0"/>
        <element name="disclose" type="contact:discloseType"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="chgPostalInfoType">
      <sequence>
        <element name="name" type="contact:postalLineType"
         minOccurs="0"/>
        <element name="org" type="contact:optPostalLineType"
         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="contact:infDataType"/>
    <element name="panData" type="contact:panDataType"/>
    <element name="trnData" type="contact:trnDataType"/>

   <!--
   <check> response elements.
   -->
    <complexType name="chkDataType">
      <sequence>
        <element name="cd" type="contact:checkType"
         maxOccurs="unbounded"/>
      </sequence>
    </complexType>

    <complexType name="checkType">
      <sequence>
        <element name="id" type="contact:checkIDType"/>
        <element name="reason" type="eppcom:reasonType"
         minOccurs="0"/>
      </sequence>
    </complexType>

    <complexType name="checkIDType">
      <simpleContent>
        <extension base="eppcom:clIDType">
          <attribute name="avail" type="boolean"
           use="required"/>
        </extension>
      </simpleContent>
    </complexType>

   <!--
   <create> response elements.
   -->
    <complexType name="creDataType">
      <sequence>
```

```
        <element name="id" type="eppcom:clIDType"/>
        <element name="crDate" type="dateTime"/>
      </sequence>
    </complexType>

   <!--
   <info> response elements.
   -->
    <complexType name="infDataType">
      <sequence>
        <element name="id" type="eppcom:clIDType"/>
        <element name="roid" type="eppcom:roidType"/>
        <element name="status" type="contact:statusType"
         maxOccurs="7"/>
        <element name="postalInfo" type="contact: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="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="trDate" type="dateTime"
         minOccurs="0"/>
        <element name="authInfo" type="contact:authInfoType"
         minOccurs="0"/>
        <element name="disclose" type="contact:discloseType"
         minOccurs="0"/>
      </sequence>
    </complexType>

   <!--
   Status is a combination of attributes and an optional human-readable
   message that may be expressed in languages other than English.
   -->
    <complexType name="statusType">
      <simpleContent>
        <extension base="normalizedString">
          <attribute name="s" type="contact:statusValueType"
           use="required"/>
          <attribute name="lang" type="language"
           default="en"/>
        </extension>
      </simpleContent>
    </complexType>

    <simpleType name="statusValueType">
      <restriction base="token">
        <enumeration value="clientDeleteProhibited"/>
        <enumeration value="clientTransferProhibited"/>
        <enumeration value="clientUpdateProhibited"/>
        <enumeration value="linked"/>
        <enumeration value="ok"/>
        <enumeration value="pendingCreate"/>
        <enumeration value="pendingDelete"/>
        <enumeration value="pendingTransfer"/>
        <enumeration value="pendingUpdate"/>
        <enumeration value="serverDeleteProhibited"/>
        <enumeration value="serverTransferProhibited"/>
        <enumeration value="serverUpdateProhibited"/>
      </restriction>
    </simpleType>

   <!--
   Pending action notification response elements.
   -->
    <complexType name="panDataType">
      <sequence>
        <element name="id" type="contact:paCLIDType"/>
        <element name="paTRID" type="epp:trIDType"/>
        <element name="paDate" type="dateTime"/>
      </sequence>
    </complexType>

    <complexType name="paCLIDType">
      <simpleContent>
        <extension base="eppcom:clIDType">
          <attribute name="paResult" type="boolean"
           use="required"/>
        </extension>
      </simpleContent>
    </complexType>

   <!--
   <transfer> response elements.
   -->
    <complexType name="trnDataType">
      <sequence>
        <element name="id" type="eppcom:clIDType"/>
        <element name="trStatus" type="eppcom:trStatusType"/>
        <element name="reID" type="eppcom:clIDType"/>
        <element name="reDate" type="dateTime"/>
        <element name="acID" type="eppcom:clIDType"/>
        <element name="acDate" type="dateTime"/>
      </sequence>
    </complexType>

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

5. 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 [RFC2781]. 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 in environments where parser encoding support incompatibility exists.

All date-time values presented via EPP MUST be expressed in Universal Coordinated Time using the Gregorian calendar. The XML Schema allows use of time zone identifiers to indicate offsets from the zero meridian, but this option MUST NOT be used with EPP. The extended date-time form using upper case "T" and "Z" characters defined in [W3C.REC-xmlschema-2-20041028] MUST be used to represent date-time values, as the XML Schema does not support truncated date-time forms or lower case "T" and "Z" characters.

Humans, organizations, and other entities often need to represent social information in both a commonly understood character set and a locally optimized character set. This specification provides features allowing representation of social information in both a subset of UTF-8 for broad readability and unrestricted UTF-8 for local optimization.

6. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688]. Two URI assignments have been registered by the IANA.

Registration request for the contact namespace:

URI: urn:ietf:params:xml:ns:contact-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 contact XML schema:

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

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

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

7. Security Considerations

Authorization information as described in Section 2.8 is REQUIRED to create a contact object. This information is used in some query and transfer operations as an additional means of determining client authorization to perform the command. Failure to protect authorization information from inadvertent disclosure can result in unauthorized transfer operations and unauthorized information release. Both client and server MUST ensure that authorization information is stored and exchanged with high-grade encryption mechanisms to provide privacy services.

The object mapping described in this document does not provide any other security services or introduce any additional considerations beyond those described by [RFC5730] or those caused by the protocol layers used by EPP.

8. Acknowledgements

RFC 3733 is a product of the PROVREG working group, which suggested improvements and provided many invaluable comments. The author wishes to acknowledge the efforts of WG chairs Edward Lewis and Jaap Akkerhuis for their process and editorial contributions. RFC 4933 and this document are individual submissions, based on the work done in RFC 3733.

Specific suggestions that have been incorporated into this document were provided by Chris Bason, Eric Brunner-Williams, Jordyn Buchanan, Robert Burbidge, Dave Crocker, Ayesha Damaraju, Anthony Eden, Sheer El-Showk, Dipankar Ghosh, Klaus Malorny, Dan Manley, Michael Mealling, Patrick Mevzek, Asbjorn Steira, and Rick Wesson.

9. References

9.1. Normative References

[ISO3166-1]
International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO Standard 3166, November 2006.

[ITU.E164.2005]
International Telecommunication Union, "The international public telecommunication numbering plan", ITU-T Recommendation E.164, February 2005.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

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

[W3C.REC-xml-20040204]
Sperberg-McQueen, C., Maler, E., Yergeau, F., Paoli, J., and T. Bray, "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]
Maloney, M., Thompson, H., Mendelsohn, N., and D. Beech, "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]
Malhotra, A. and P. Biron, "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>.

9.2. Informative References

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC4933] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Contact Mapping", RFC 4933, May 2007.

Appendix A. Changes from RFC 4933

1. Changed "This document obsoletes RFC 3733" to "This document obsoletes RFC 4933".

2. Replaced references to RFC 0822 with references to 5322.

3. Replaced references to RFC 3733 with references to 4933.

4. Replaced references to RFC 4930 with references to 5730.

5. Updated reference to ISO 3166-1.

6. Removed pendingRenew status from Section 2.2 because this document does not define a mapping for the EPP <renew> command.

7. Modified text in Section 3.2.2 to include 2305 response code.

8. Updated Section 5.

9. Added "Other notification methods MAY be used in addition to the required service message" in Section 3.2.

10. Added 2201 response code text in Section 3.2.

11. Added BSD license text to XML schema section.

Author’s Address

Scott Hollenbeck
VeriSign, Inc.
21345 Ridgetop Circle
Dulles, VA 20166-6503
US

EMail: shollenbeck@verisign.com

==

Network Working Group   S. Hollenbeck
Request for Comments: 5734   VeriSign, Inc.
STD: 69   August 2009
Obsoletes: 4934
Category: Standards Track

Extensible Provisioning Protocol (EPP) Transport over TCP

Abstract

This document describes how an Extensible Provisioning Protocol (EPP) session is mapped onto a single Transmission Control Protocol (TCP) connection. This mapping requires use of the Transport Layer Security (TLS) protocol to protect information exchanged between an EPP client and an EPP server. This document obsoletes RFC 4934.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes how the Extensible Provisioning Protocol (EPP) is mapped onto a single client-server TCP connection. Security services beyond those defined in EPP are provided by the Transport Layer Security (TLS) Protocol [RFC2246]. EPP is described in [RFC5730]. TCP is described in [RFC0793]. This document obsoletes RFC 4934 [RFC4934].

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 [RFC2119].

2. Session Management

Mapping EPP session management facilities onto the TCP service is straightforward. An EPP session first requires creation of a TCP connection between two peers, one that initiates the connection request and one that responds to the connection request. The initiating peer is called the "client", and the responding peer is called the "server". An EPP server MUST listen for TCP connection requests on a standard TCP port assigned by IANA.

The client MUST issue an active OPEN call, specifying the TCP port number on which the server is listening for EPP connection attempts. The EPP server MUST return an EPP <greeting> to the client after the TCP session has been established.

An EPP session is normally ended by the client issuing an EPP <logout> command. A server receiving an EPP <logout> command MUST end the EPP session and close the TCP connection with a CLOSE call. A client MAY end an EPP session by issuing a CLOSE call.

A server MAY limit the life span of an established TCP connection. EPP sessions that are inactive for more than a server-defined period MAY be ended by a server issuing a CLOSE call. A server MAY also close TCP connections that have been open and active for longer than a server-defined period.

3. Message Exchange

With the exception of the EPP server greeting, EPP messages are initiated by the EPP client in the form of EPP commands. An EPP server MUST return an EPP response to an EPP command on the same TCP connection that carried the command. If the TCP connection is closed after a server receives and successfully processes a command but before the response can be returned to the client, the server MAY attempt to undo the effects of the command to ensure a consistent state between the client and the server. EPP commands are idempotent, so processing a command more than once produces the same net effect on the repository as successfully processing the command once.

An EPP client streams EPP commands to an EPP server on an established TCP connection. A client MUST NOT distribute commands from a single EPP session over multiple TCP connections. A client MAY establish multiple TCP connections to support multiple EPP sessions with each session mapped to a single connection. A server SHOULD limit a client to a maximum number of TCP connections based on server capabilities and operational load.

EPP describes client-server interaction as a command-response exchange where the client sends one command to the server and the server returns one response to the client. A client might be able to realize a slight performance gain by pipelining (sending more than one command before a response for the first command is received) commands with TCP transport, but this feature does not change the basic single command, single response operating mode of the core protocol.

Each EPP data unit MUST contain a single EPP message. Commands MUST be processed independently and in the same order as sent from the client.

A server SHOULD impose a limit on the amount of time required for a client to issue a well-formed EPP command. A server SHOULD end an EPP session and close an open TCP connection if a well-formed command is not received within the time limit.

A general state machine for an EPP server is described in Section 2 of [RFC5730]. General client-server message exchange using TCP transport is illustrated in Figure 1.

```
                       Client                  Server
                  |                                     |
                  |                Connect              |
                  | >>------------------------------->> |
                  |                                     |
                  |             Send Greeting           |
                  | <<-------------------------------<< |
                  |                                     |
                  |             Send <login>            |
                  | >>------------------------------->> |
                  |                                     |
                  |             Send Response           |
                  | <<-------------------------------<< |
                  |                                     |
                  |             Send Command            |
                  | >>------------------------------->> |
                  |                                     |
                  |             Send Response           |
                  | <<-------------------------------<< |
                  |                                     |
                  |            Send Command X           |
                  | >>------------------------------->> |
                  |                                     |
                  |    Send Command Y                   |
                  | >>---------------+                  |
                  |                  |                  |
                  |                  |                  |
                  |            Send Response X          |
                  | <<---------------(---------------<< |
                  |                  |                  |
                  |                  |                  |
                  |                  +--------------->> |
                  |                                     |
                  |            Send Response Y          |
                  | <<-------------------------------<< |
                  |                                     |
                  |             Send <logout>           |
                  | >>------------------------------->> |
                  |                                     |
                  |     Send Response & Disconnect      |
                  | <<-------------------------------<< |
                  |                                     |
```

Figure 1: TCP Client-Server Message Exchange

4. Data Unit Format

The EPP data unit contains two fields: a 32-bit header that describes the total length of the data unit, and the EPP XML instance. The length of the EPP XML instance is determined by subtracting four octets from the total length of the data unit. A receiver must successfully read that many octets to retrieve the complete EPP XML instance before processing the EPP message.

EPP Data Unit Format (one tick mark represents one bit position):

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Total Length                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         EPP XML Instance                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-//-+-+-+-+-+-+-+-+-+-+-+
```

Total Length (32 bits): The total length of the EPP data unit measured in octets in network (big endian) byte order. The octets contained in this field MUST be included in the total length calculation.

EPP XML Instance (variable length): The EPP XML instance carried in the data unit.

5. Transport Considerations

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

- TCP includes features to provide reliability, flow control, ordered delivery, and congestion control. Section 1.5 of RFC 793 [RFC0793] describes these features in detail; congestion control principles are described further in RFC 2581 [RFC2581] and RFC 2914 [RFC2914]. TCP is a connection-oriented protocol, and Section 2 of this document describes how EPP sessions are mapped to TCP connections.

- Sections 2 and 3 of this document describe how the stateful nature of EPP is preserved through managed sessions and controlled message exchanges.

- Section 3 of this document notes that command pipelining is possible with TCP, though batch-oriented processing (combining multiple EPP commands in a single data unit) is not permitted.

- Section 4 of this document describes features to frame data units by explicitly specifying the number of octets used to represent a data unit.

6. Internationalization Considerations

This document does not introduce or present any internationalization or localization issues.

7. IANA Considerations

System port number 700 has been assigned by the IANA for mapping EPP onto TCP.

User port number 3121 (which was used for development and test purposes) has been reclaimed by the IANA.

8. Security Considerations

EPP as-is provides only simple client authentication services using identifiers and plain text passwords. A passive attack is sufficient to recover client identifiers and passwords, allowing trivial command forgery. Protection against most other common attacks MUST be provided by other layered protocols.

When layered over TCP, the Transport Layer Security (TLS) Protocol version 1.0 [RFC2246] or its successors (such as TLS 1.2 [RFC5246]), using the latest version supported by both parties, MUST be used to provide integrity, confidentiality, and mutual strong client-server authentication. Implementations of TLS often contain a weak cryptographic mode that SHOULD NOT be used to protect EPP. Clients and servers desiring high security SHOULD instead use TLS with cryptographic algorithms that are less susceptible to compromise.

Authentication using the TLS Handshake Protocol confirms the identity of the client and server machines. EPP uses an additional client identifier and password to identify and authenticate the client’s user identity to the server, supplementing the machine authentication provided by TLS. The identity described in the client certificate and the identity described in the EPP client identifier can differ, as a server can assign multiple user identities for use from any particular client machine. Acceptable certificate identities MUST be negotiated between client operators and server operators using an out-of-band mechanism. Presented certificate identities MUST match negotiated identities before EPP service is granted.

There is a risk of login credential compromise if a client does not properly identify a server before attempting to establish an EPP session. Before sending login credentials to the server, a client needs to confirm that the server certificate received in the TLS handshake is an expected certificate for the server. A client also needs to confirm that the greeting received from the server contains expected identification information. After establishing a TLS session and receiving an EPP greeting on a protected TCP connection, clients MUST compare the certificate subject and/or subjectAltName to expected server identification information and abort processing if a mismatch is detected. If certificate validation is successful, the client then needs to ensure that the information contained in the received certificate and greeting is consistent and appropriate. As described above, both checks typically require an out-of-band exchange of information between client and server to identify expected values before in-band connections are attempted.

EPP TCP servers are vulnerable to common TCP denial-of-service attacks including TCP SYN flooding. Servers SHOULD take steps to minimize the impact of a denial-of-service attack using combinations of easily implemented solutions, such as deployment of firewall technology and border router filters to restrict inbound server access to known, trusted clients.

9. TLS Usage Profile

The client should initiate a connection to the server and then send the TLS Client Hello to begin the TLS handshake. When the TLS handshake has finished, the client can then send the first EPP message.

TLS implementations are REQUIRED to support the mandatory cipher suite specified in the implemented version:

- TLS 1.0 [RFC2246]: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS 1.1 [RFC4346]: TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS 1.2 [RFC5246]: TLS_RSA_WITH_AES_128_CBC_SHA

This document is assumed to apply to future versions of TLS, in which case the mandatory cipher suite for the implemented version MUST be supported.

Mutual client and server authentication using the TLS Handshake Protocol is REQUIRED. Signatures on the complete certification path for both client machine and server machine MUST be validated as part of the TLS handshake. Information included in the client and server certificates, such as validity periods and machine names, MUST also be validated. A complete description of the issues associated with certification path validation can be found in RFC 5280 [RFC5280]. EPP service MUST NOT be granted until successful completion of a TLS handshake and certificate validation, ensuring that both the client machine and the server machine have been authenticated and cryptographic protections are in place.

If the client has external information as to the expected identity of the server, the server name check MAY be omitted. For instance, a client may be connecting to a machine whose address and server name are dynamic, but the client knows the certificate that the server will present. In such cases, it is important to narrow the scope of acceptable certificates as much as possible in order to prevent man-in-the-middle attacks. In special cases, it might be appropriate for the client to simply ignore the server's identity, but it needs to be understood that this leaves the connection open to active attack.

During the TLS negotiation, the EPP client MUST check its understanding of the server name / IP address against the server's identity as presented in the server Certificate message in order to prevent man-in-the-middle attacks. In this section, the client's understanding of the server's identity is called the "reference identity". Checking is performed according to the following rules in the specified order:

o If the reference identity is a server name:

   * If a subjectAltName extension of the dNSName [CCITT.X509.1988] type is present in the server's certificate, then it SHOULD be used as the source of the server's identity. Matching is performed as described in Section 7.2 of [RFC5280], with the exception that wildcard matching (see below) is allowed for dNSName type. If the certificate contains multiple names (e.g., more than one dNSName field), then a match with any one of the fields is considered acceptable.

   * The '*' (ASCII 42) wildcard character is allowed in subjectAltName values of type dNSName, and then only as the left-most (least significant) DNS label in that value. This wildcard matches any left-most DNS label in the server name. That is, the subject *.example.com matches the server names a.example.com and b.example.com, but does not match example.com or a.b.example.com.

   * The server’s identity MAY also be verified by comparing the reference identity to the Common Name (CN) [RFC4519] value in the leaf Relative Distinguished Name (RDN) of the subjectName field of the server’s certificate. This comparison is performed using the rules for comparison of DNS names in bullet 1 above (including wildcard matching). Although the use of the Common Name value is existing practice, it is deprecated, and Certification Authorities are encouraged to provide subjectAltName values instead. Note that the TLS implementation may represent DNs in certificates according to X.500 or other conventions. For example, some X.500 implementations order the RDNs in a DN using a left-to-right (most significant to least significant) convention instead of LDAP’s right-to-left convention.

- If the reference identity is an IP address:

   * The iPAddress subjectAltName SHOULD be used by the client for comparison. In such a case, the reference identity MUST be converted to the "network byte order" octet string representation. For IP Version 4 (as specified in RFC 791 [RFC0791]), the octet string will contain exactly four octets. For IP Version 6 (as specified in RFC 2460 [RFC2460]), the octet string will contain exactly sixteen octets. This octet string is then compared against subjectAltName values of type iPAddress. A match occurs if the reference identity octet string and value octet strings are identical.

If the server identity check fails, user-oriented clients SHOULD either notify the user (clients MAY give the user the opportunity to continue with the EPP session in this case) or close the transport connection and indicate that the server’s identity is suspect. Automated clients SHOULD return or log an error indicating that the server’s identity is suspect and/or SHOULD close the transport connection. Automated clients MAY provide a configuration setting that disables this check, but MUST provide a setting which enables it.

During the TLS negotiation, the EPP server MUST verify that the client certificate matches the reference identity previously negotiated out of band, as specified in Section 8. The server should match the entire subject name or the subjectAltName as described in RFC 5280. The server MAY enforce other restrictions on the subjectAltName, for example if it knows that a particular client is always connecting from a particular hostname / IP address.

All EPP messages MUST be sent as TLS "application data". It is possible that multiple EPP messages are contained in one TLS record, or that an EPP message is transferred in multiple TLS records.

When no data is received from a connection for a long time (where the application decides what "long" means), a server MAY close the connection. The server MUST attempt to initiate an exchange of close_notify alerts with the client before closing the connection. Servers that are unprepared to receive any more data MAY close the connection after sending the close_notify alert, thus generating an incomplete close on the client side.

10. Acknowledgements

RFC 3734 is a product of the PROVREG working group, which suggested improvements and provided many invaluable comments. The author wishes to acknowledge the efforts of WG chairs Edward Lewis and Jaap Akkerhuis for their process and editorial contributions. RFC 4934 and this document are individual submissions, based on the work done in RFC 3734.

Specific suggestions that have been incorporated into this document were provided by Chris Bason, Randy Bush, Patrik Faltstrom, Ned Freed, James Gould, Dan Manley, and John Immordino.

11. References

11.1. Normative References

[CCITT.X509.1988]
International Telephone and Telegraph Consultative Committee, "Information Technology - Open Systems Interconnection - The Directory: Authentication Framework", CCITT Recommendation X.509, November 1988.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC4519] Sciberras, A., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, June 2006.

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

11.2. Informative References

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4934] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Transport Over TCP", RFC 4934, May 2007.

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

Appendix A. Changes from RFC 4934

1. Changed "This document obsoletes RFC 3734" to "This document obsoletes RFC 4934".

2. Replaced references to RFC 3280 with references to 5280.

3. Replaced references to RFC 3734 with references to 4934.

4. Updated references to RFC 4346 and TLS 1.1 with references to 5246 and TLS 1.2.

5. Replaced references to RFC 4930 with references to 5730.

6. Added clarifying TLS Usage Profile section and included references.

7. Moved the paragraph that begins with "Mutual client and server authentication" from the Security Considerations section to the TLS Usage Profile section.

Author’s Address

Scott Hollenbeck
VeriSign, Inc.
21345 Ridgetop Circle
Dulles, VA 20166-6503
US

EMail: shollenbeck@verisign.com