                  Extensible Provisioning Protocol (EPP)

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

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.

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 [XML] and XML Schema notation as described in [XMLS-1] and [XMLS-
    2]. EPP meets and exceeds the requirements for a generic registry
    registrar protocol as described in [RFC3375].

    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.

This 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 is provided only to illustrate element
relationships and is not a REQUIRED feature of this protocol.

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 |       +-----------------+
        |    |    |               |                 |
        |    |    |       +--------------+           |  <login>
        |    |    |       | Prepare Fail |           |  Received
        |    |    +-----|   Response    |           V
        |    |      Send +--------------+     +-----------------+
        |    |      2501           ^          |   Processing    |
        |    |      Response       |          |    <login>      |
        |    |                   +---------|                 |
        |    |              Auth Fail +-----------------+
        |    |                                   |
        |    |                                   | Auth OK
        |    |                                   V
        |    |              Timeout        +-----------------+
        |    +--------------------------|   Waiting for   |
        |                               |    Command      |
        | Send x5xx                     +-----------------+
        | Response   +-----------------+  Send     ^   |
      +-----------|     Prepare     | Response |   | Command
        |         |     Response    |---------+   | Received
        |         +-----------------+             V
        |                  ^              +-----------------+
        | Command |                      |   Processing    |
        | Processed +---------|          |    Command      |
        +-----------------+              +-----------------+
```
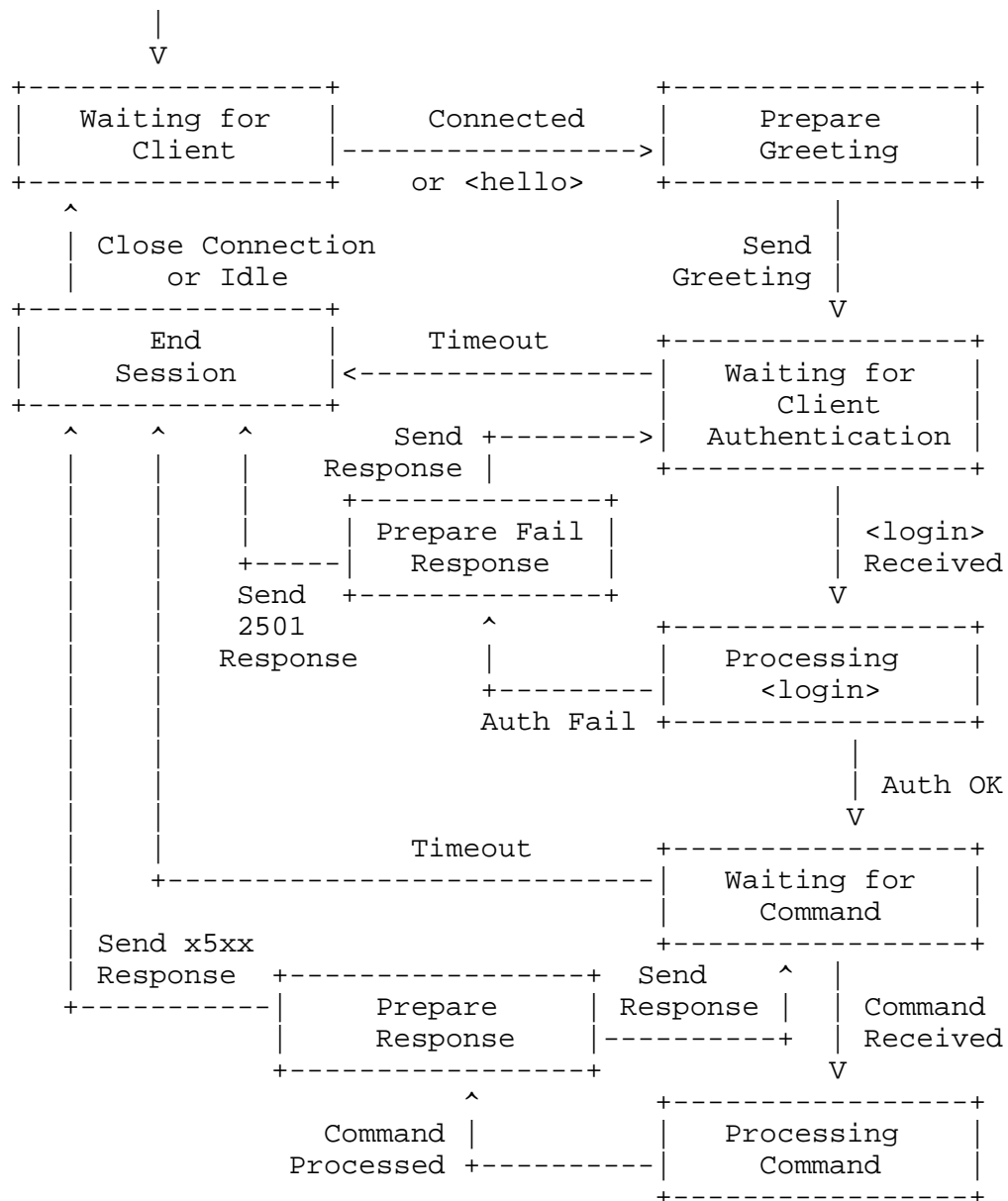
                 Figure 1: EPP Server State Machine

   EPP commands fall into three categories: session management commands,
   query commands, and data 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 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 specific strings used to associate URIs and namespaces
(such as the string "foo" in "xmlns:foo") in EPP are illustrative and
are not needed for interoperability.

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 2279 [RFC2279]) 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.

Normative section 4.3.3 and non-normative appendix F of [XML]
describe use of a BOM to identify the character encoding in the
absence of an XML declaration or encapsulating headers. Appendix F
includes a BOM to represent UTF-8 encoding, though section 4.3.3
notes that a BOM is not needed to identify UTF-8 encoding. Section
4.3.3 was later amended (see [XMLE]) to clarify that a BOM MAY be
used to identify UTF-8 encoding. 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
   [RFC793] or SCTP [RFC2960] that provides congestion avoidance that
   follows RFC 2914 [RFC2914], or if it maps onto a protocol such as
   SMTP [RFC2821] or 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, the namespace used
   within the protocol, and the location of the protocol schema. 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"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
        epp-1.0.xsd">
   </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 by sending a <hello> to a server. Use of this
   element is essential in a connection-less environment where a server
   can not 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: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
   C: epp-1.0.xsd">
   C:
   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 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 [RFC3066].

   - 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 [P3P]
   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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <info>
C:      <obj:info xmlns:obj="urn:ietf:params:xml:ns:obj"
C:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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) 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:

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

   - 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:

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

   - 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:      xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:      epp-1.0.xsd">
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:     xsi:schemaLocation="urn:ietf:params:xml:ns:ext ext.xsd">
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:   xsi:schemaLocation="urn:ietf:params:xml:ns:object object.xsd">
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:   xsi:schemaLocation="urn:ietf:params:xml:ns:object object.xsd">
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:     xsi:schemaLocation="urn:ietf:params:xml:ns:object object.xsd">
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>
```

An 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 a name server name 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 execution. 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 SASL mechanism presented in [RFC2595] 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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 action MUST be open to all
authorized 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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 action MUST be open
to all authorized clients.

2.9.2. Query Commands

EPP provides four commands to retrieve object information: <check> to
determine if an object can be provisioned within a repository, <info>
to retrieve detailed information associated with a known object,
<poll> to receive service notifications from the server, and
<transfer> to retrieve object transfer status information.

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 identify 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: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
   C: epp-1.0.xsd">
   C: <command>
   C: <obj:check xmlns:obj="urn:ietf:params:xml:ns:obj"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
   C: <obj:name>example1</obj:name>
   C: <obj:name>example2</obj:name>
   C: <obj:name>example3</obj:name>
   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 and the location of the
   object schema. 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> elements 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 can not be provisioned.

   - An OPTIONAL <obj:reason> element that MAY be provided when an
      object can not be provisioned. If present, this element contains
      server-specific text to help explain why the object can not 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: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   S: xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
   S: epp-1.0.xsd">
   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: xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
   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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <info>
C:      <obj:info xmlns:obj="urn:ietf:params:xml:ns:obj"
C:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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 location of the
object schema 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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 MUST be created for all clients affected by an action on an object. For example, <transfer> actions MUST be reported to both the client that requests an object transfer and the client that has the authority to approve or reject the transfer request.

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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj-1.0
S:       obj-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <poll op="ack" msgID="12345"/>
C:    <clTRID>ABC-12346</clTRID>
C:  </command>
C:</epp>
```

A <poll> acknowledgement response notes the number of messages
remaining in the queue and the ID of the next message available for
retrieval.

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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
S:  <response>
S:    <result code="1000">
S:      <msg>Command completed successfully</msg>
S:    </result>
S:    <msgQ count="4" id="12346"/>
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <transfer op="query">
C:      <obj:transfer xmlns:obj="urn:ietf:params:xml:ns:obj"
C:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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 and the location
   of the object schema. 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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, <update> to change information associated with
   an object, and <transfer> to manage changes in client sponsorship of
   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, 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: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
   C: epp-1.0.xsd">
   C: <command>
   C: <create>
   C: <obj:create xmlns:obj="urn:ietf:params:xml:ns:obj"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
   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 and the location of the
object schema. 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <delete>
C:      <obj:delete xmlns:obj="urn:ietf:params:xml:ns:obj"
C:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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 and the location of the
object schema. 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
   C: epp-1.0.xsd">
   C: <command>
   C: <renew>
   C: <obj:renew xmlns:obj="urn:ietf:params:xml:ns:obj"
   C: xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
   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 and the location of the
   object schema. 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: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   S: xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
   S: epp-1.0.xsd">
   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:         xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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
by queuing a service message for retrieval via the <poll> command.
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <transfer op="request">
C:      <obj:transfer xmlns:obj="urn:ietf:params:xml:ns:obj"
C:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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 and the location of the
object schema. 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
C:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
C:     epp-1.0.xsd">
C:  <command>
C:    <update>
C:      <obj:update xmlns:obj="urn:ietf:params:xml:ns:obj"
C:       xsi:schemaLocation="urn:ietf:params:xml:ns:obj obj.xsd">
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 and the location of the
object schema. 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:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
S:     xsi:schemaLocation="urn:ietf:params:xml:ns:epp-1.0
S:     epp-1.0.xsd">
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 [RFC2821]. 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 has been 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. Specific result
codes are listed in the table below.

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
[RFC3066].

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 the
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 can not be executed due to a
sequencing or context error. For example, a <logout> command can not
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 can not 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 required 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 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 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 can not 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 can not 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 can not 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 can not 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 schemas. 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"
          schemaLocation="eppcom-1.0.xsd"/>

  <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 [RFC3066].

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 [RFC3339]
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.

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
      and confidentiality.

      EPP uses a variant of the PLAIN SASL mechanism described in [RFC2595]
      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.

8. Acknowledgements

      This document was originally written as an individual submission
      Internet-Draft. The provreg working group later adopted it as a
      working group document and provided many invaluable comments and
      suggested improvements. The author wishes to acknowledge the efforts
      of WG chairs Edward Lewis and Jaap Akkerhuis for their process and
      editorial contributions.

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.

   [RFC2279]   Yergeau, F., "UTF-8, a transformation format of ISO
               10646", RFC 2279, January 1998.

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

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

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

   [RFC3066]   Alvestrand, H., "Tags for the Identification of
               Languages", BCP 47, RFC 3066, January 2001.

   [RFC3339]   Klyne, G. and C. Newman, "Date and Time on the Internet:
               Timestamps", RFC 3339, July 2002.

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

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

   [XML]   Editor T. Bray et al.: "Extensible Markup Language (XML)
               1.0 (Second Edition)", W3C Recommendation 6 October 2000.

   [XMLE]   "XML 1.0 Second Edition Specification Errata", E22, 25
               July 2001, http://www.w3.org/XML/xml-V10-2e-errata#E22.

   [XMLS-1]   Editors H. Thompson et al.: "XML Schema Part 1:
               Structures", W3C Recommendation 2 May 2001.

   [XMLS-2]   Editors P. Biron, A. Malhotra: "XML Schema Part 2:
               Datatypes", W3C Recommendation 2 May 2001.

9.2. Informative References

   [P3P]   Editor M. Marchiori: "The Platform for Privacy Preferences
               1.0 (P3P1.0) Specification", W3C Recommendation 16 April
               2002.

   [RFC793]   Postel, J., "Transmission Control Protocol", STD 7, RFC
               793, September 1981.

   [RFC2595]   Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC
               2595, June 1999.

   [RFC2821]   Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC
               2821, April 2001.

   [RFC2960]   Stewart, R., Xie, Q., Morneault, K., Sharp, C.,
               Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M.,
               Zhang, L. and V. Paxson, "Stream Control Transmission
               Protocol", RFC 2960, October 2000.

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

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 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.

3.2.5. 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

   Mandatory 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 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 sub-section 2.9
   "Standalone Document Declaration" and section 5 "Conformance" of
   [XML].

   Published specification: This document.

   Applications which 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 and email address for further information: See the "Author's
   Address" section of this document.

   Intended usage: COMMON

   Author/Change controller: IETF

Author's Address

   Scott Hollenbeck
   VeriSign Global Registry Services
   21345 Ridgetop Circle
   Dulles, VA 20166-6503
   USA

   EMail: shollenbeck@verisign.com

Full Copyright Statement

Intellectual Property

Acknowledgement