Network Working Group
Internet-Draft
Expires: August 26, 2001

M.T. Rose
Invisible Worlds, Inc.
G. Klyne
Content Technologies Limited
D.H. Crocker
Brandenburg Consulting
February 25, 2001

The APEX Presence Service
draft-mrose-apex-presence-03

Status of this Memo

Copyright Notice

Abstract

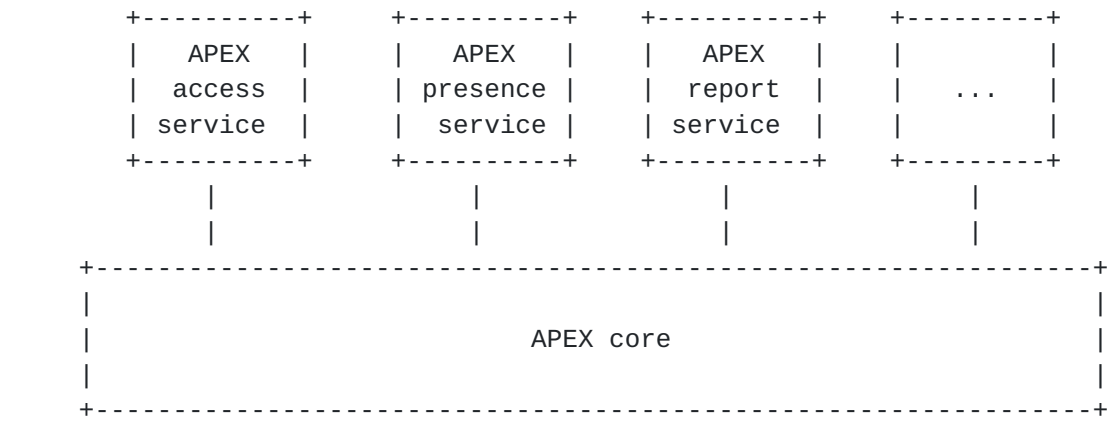
This memo describes the APEX presence service, addressed as the well-known endpoint "apex=presence". The presence service is used to manage presence information for APEX endpoints.

Table of Contents

1. Introduction

This memo describes a presence service that is built upon the APEX[1] "relaying mesh". The APEX presence service is used to manage presence information for APEX endpoints.

APEX, at its core, provides a best-effort datagram service. Within an administrative domain, all relays must be able to handle messages for any endpoint within that domain. APEX services are logically defined as endpoints but given their ubiquitous semantics they do not necessarily need to be associated with a single physical endpoint. As such, they may be provisioned co-resident with each relay within an administrative domain, even though they are logically provided on top of the relaying mesh, i.e.,



That is, applications communicate with an APEX service by exchanging data with a "well-known endpoint" (WKE).

APEX applications communicate with the presence service by exchanging data with the well-known endpoint "apex=presence" in the corresponding administrative domain, e.g., "apex=presence@example.com" is the endpoint associated with the presence service in the "example.com" administrative domain.

Note that within a single administrative domain, the presence service makes use of the APEX access[3] service in order to determine if an originator is allowed to view or manage presence information.

2. Management of Presence Information

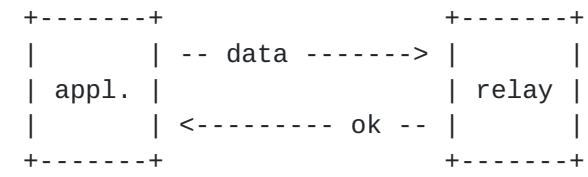
Management of presence information falls into three categories:

- applications may update the presence entry associated with an endpoint;
- applications may subscribe to receive presence information about an endpoint; and,
- applications may find out who is subscribed to receive presence information.

Each is now described in turn.

2.1 Update of Presence Information

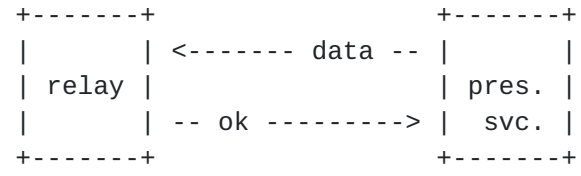
When an application wants to modify the presence entry associated with an endpoint, it sends a publish operation to the service, e.g.,




```
  C: <data content='#Content'>
        <originator identity='fred@example.com' />
        <recipient identity='apex=presence@example.com' />
        <data-content Name='Content'>
            <publish publisher='fred@example.com' transID='1'
                     timeStamp='14 May 2000 13:30:00 -0800'>
                <presence publisher='fred@example.com'
                       lastUpdate='14 May 2000 13:02:00 -0800'
                       publisherInfo='http://www.example.com/fred/'>
                    <tuple
                      destination='apex:fred/appl=im@example.com'
                      availableUntil='14 May 2000 14:02:00 -0800' />
                    <tuple destination='mailto:fred@flintstone.com'
                      availableUntil='31 Dec 2525 23:59:59 -0800' />
                </presence>
            </publish>
        </data-content>
     </data>
  S: <ok />
```

Note that this example uses the subaddress-specification convention of RFC 2846[4] (e.g., "fred/appl=im") to denote multiplexing of traffic for a particular endpoint. Of course, popular applications may have their own URI method assigned to them (e.g., "im:fred@example.com").

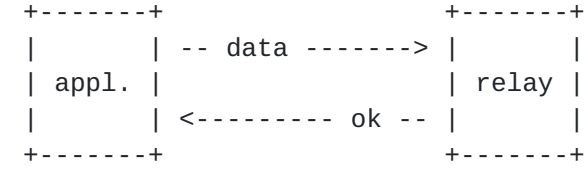
The service immediately responds with a reply operation containing the same transaction-identifier, e.g.,




```
C: <data content='#Content'>
       <originator identity='apex=presence@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'>
           <reply code='250' transID='1' />
       </data-content>
   </data>
S: <ok />
```

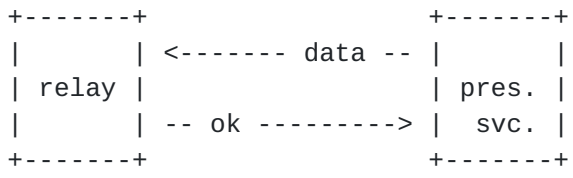
2.2 Distribution of Presence Information

When an application wants to (periodically) receive the presence entry associated with an endpoint, it sends a subscribe operation to the service, e.g.,




```
C: <data content='#Content'>
       <originator identity='wilma@example.com' />
       <recipient identity='apex=presence@example.com' />
       <data-content Name='Content'>
           <subscribe publisher='fred@example.com' duration='86400'
                      transID='100' />
       </data-content>
   </data>
S: <ok />
```

The service immediately responds with a publish operation containing the same transaction-identifier, e.g.,

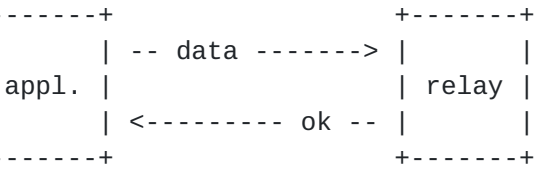



```
C: <data content='#Content'>
       <originator identity='apex=presence@example.com' />
       <recipient identity='wilma@example.com' />
       <data-content Name='Content'>
           <publish publisher='fred@example.com' transID='100'
                    timeStamp='14 May 2000 13:30:00 -0800'>
               <presence publisher='fred@example.com'
                      lastUpdate='14 May 2000 13:02:00 -0800'
                      publisherInfo='http://www.example.com/fred/'>
                   <tuple
                     destination='apex:fred/appl=im@example.com'
                     availableUntil='14 May 2000 14:02:00 -0800' />
               </presence>
           </publish>
       </data-content>
   </data>
S: <ok />
```

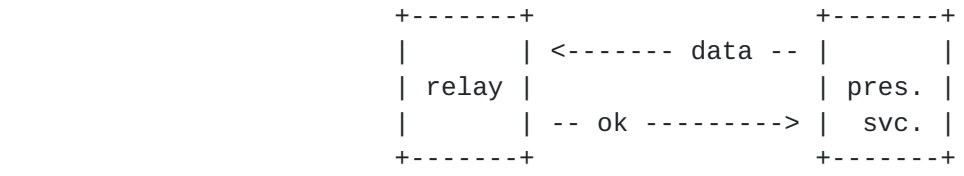
Subsequently, for up to the specified "duration", the service sends new publish operations whenever there are any changes to the endpoint's presence information. If the "duration" is zero-valued, a one time poll of the presence information is achieved; otherwise, at the end of the "duration", a terminate operation is sent.

Note that Step 5 of Section 4.4 requires that the "lastUpdate" attribute of a presence entry be supplied in order to update that entry; accordingly, applications must successfully retrieve an publish entry prior to trying to update that entry. This is usually accomplished by subscribing with a zero-valued duration.

Either the subscriber or the service may cancel a subscription by sending a terminate operation, e.g.,

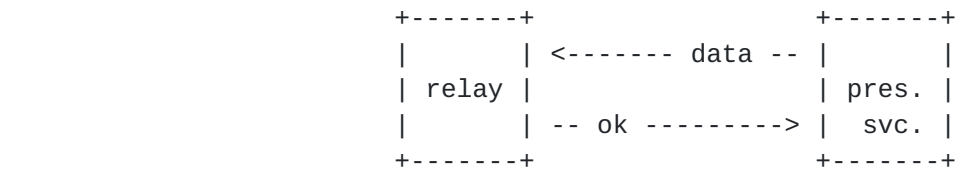



```
C: <data content='#Content'>
       <originator identity='wilma@example.com' />
       <recipient identity='apex=presence@example.com' />
       <data-content Name='Content'>
           <terminate transID='100' />
       </data-content>
   </data>
S: <ok />
```




```
C: <data content='#Content'>
       <originator identity='apex=presence@example.com' />
       <recipient identity='wilma@example.com' />
       <data-content Name='Content'>
           <reply code='250' transID='100' />
       </data-content>
   </data>
S: <ok />
```

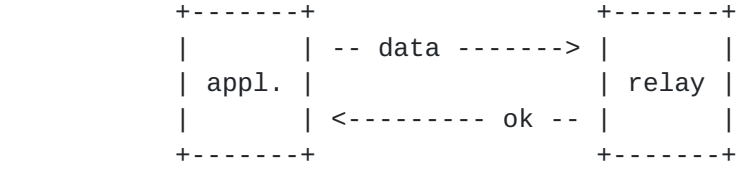
or




```
C: <data content='#Content'>
       <originator identity='apex=presence@example.com' />
       <recipient identity='wilma@example.com' />
       <data-content Name='Content'>
           <terminate transID='100' />
       </data-content>
   </data>
S: <ok />
```

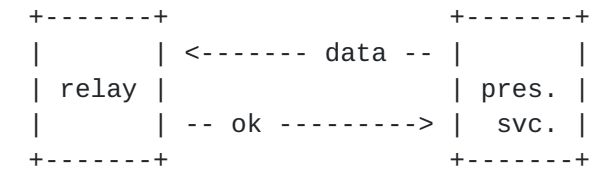
2.3 Distribution of Watcher Information

When an application wants to (periodically) receive notices about endpoints that are subscribed to receive presence information, it sends a watch operation to the service, e.g.,



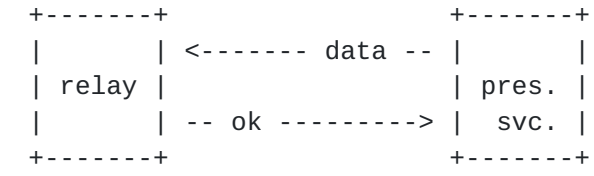

```
C: <data content='#Content'>
       <originator identity='fred@example.com' />
       <recipient identity='apex=presence@example.com' />
       <data-content Name='Content'>
           <watch publisher='fred@example.com' duration='86400'
                  transID='2' />
       </data-content>
   </data>
S: <ok />
```

The service immediately responds with a reply operation containing the same transaction-identifier, e.g.,




```
C: <data content='#Content'>
       <originator identity='apex=presence@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'
           <reply code='250' transID='2' />
       </data-content>
   </data>
S: <ok />
```

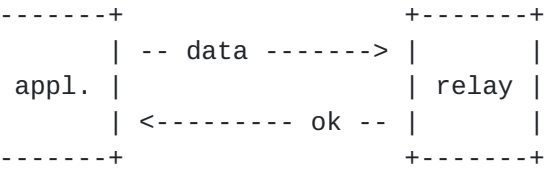
For each current subscriber, the service immediately sends a notify operation containing the same transaction-identifier, e.g.,



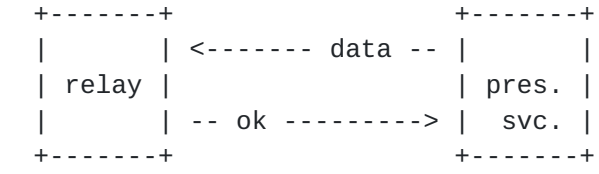

```
C: <data content='#Content'>
       <originator identity='apex=presence@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'>
           <notify subscriber='wilma@example.com' transID='2' />
       </data-content>
   </data>
S: <ok />
```

Subsequently, for up to the specified "duration", the service sends new notify operations whenever an application subscribes successfully. If the "duration" is zero-valued, a one time poll of the watcher information is achieved; otherwise, at the end of the "duration", a terminate operation is sent.

Either the watcher or the service may cancel the request by sending a terminate operation, e.g.,

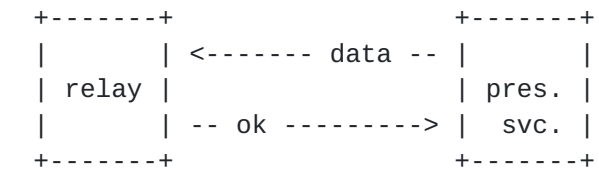



```
  C: <data content='#Content'>
        <originator identity='fred@example.com' />
        <recipient identity='apex=presence@example.com' />
        <data-content Name='Content'>
            <terminate transID='2' />
        </data-content>
     </data>
  S: <ok />
```




```
  C: <data content='#Content'>
        <originator identity='apex=presence@example.com' />
        <recipient identity='fred@example.com' />
        <data-content Name='Content'>
            <reply code='250' transID='2' />
         </data-content>
     </data>
  S: <ok />
```

or




```
  C: <data content='#Content'>
        <originator identity='apex=presence@example.com' />
        <recipient identity='fred@example.com' />
        <data-content Name='Content'>
            <terminate transID='2' />
         </data-content>
     </data>
  S: <ok />
```

3. Format of Presence Entries

Each administrative domain is responsible for maintaining a "presence entry" for each of its endpoints (regardless of whether those endpoints are currently attached to the relaying mesh).

Section 6 defines the syntax for presence entries. Each presence entry has a "publisher" attribute, a "lastUpdate" attribute, a "publisherInfo" attribute, and contains one or more "tuple" elements:

- the "publisher" attribute specifies the endpoint associated with the presence entry;
- the "lastUpdate" attribute specifies the date and time that the service last updated the presence entry;
- the "publisherInfo" attribute specifies arbitrary information about the publisher (using a URI); and,
- each "tuple" element specifies information about an entity associated with the endpoint.

Each "tuple" element has a "destination" attribute, an "availableUntil" attribute, a "tupleInfo" attribute, and contains zero or more "capability" elements:

- the "destination" attribute identifies the entity as a URI (e.g., "apex:fred/appl=im@example.com" or "mailto:fred@flintstone.com");
- the "availableUntil" attribute specifies the latest date and time that the entity is capable of receiving messages;
- the "tupleInfo" attribute specifies arbitrary information about the entity (using a URI); and,
- each "capability" element contains a specification as to the kinds of content the entity is capable of receiving.

Each "capability" element contains arbitrary character data formatted according to the standard indicated in the element's "baseline" attribute.

4. The Presence Service

Section 5 contains the APEX service registration for the presence service:

- Within an administrative domain, the service is addressed using the well-known endpoint of "apex=presence".
- Section 6 defines the syntax of the operations exchanged with the service.
- A consumer of the service initiates communications by sending data containing the subscribe, watch, or publish operation.
- In addition to replying to these operations, the service may also initiate communications by sending data containing the terminate, publish, or notify operations.

An implementation of the service must maintain information about both presence entries and in-progress operations in persistent storage.

Consult Section 6.1.1 of [1] for a discussion on the properties of long-lived transaction-identifiers.

4.1 Use of XML and MIME

Section 4.1 of [1] describes how arbitrary MIME content is exchanged as a BEEP[2] payload. For example, to transmit:

```
<data content='...'>
     <originator identity='apex=presence@example.com' />
    <recipient identity='fred@example.com' />
</data>
```

where "..." refers to: <reply code='250' transID='1' />

then the corresponding BEEP message might look like this:

```
C: MSG 1 1 . 42 1234
C: Content-Type: multipart/related; boundary="boundary";
C:               start="<1@example.com>";
C:               type="application/beep+xml"
C:
C: --boundary
C: Content-Type: application/beep+xml
C: Content-ID: <1@example.com>
C:
C: <data content='cid:2@example.com'>
C:     <originator identity='fred@example.com' />
C:     <recipient identity='apex=presence@example.com' />
C: </data>
C: --boundary
C: Content-Type: application/beep+xml
C: Content-ID: <2@example.com>
C:
C: <reply code='250' transID='1' />
C: --boundary--
C: END
```

or this:

```
C: MSG 1 1 . 42 1234
C: Content-Type: application/beep+xml
C:
C: <data content='#Content'>
C:     <originator identity='fred@example.com' />
C:     <recipient identity='apex=presence@example.com' />
C:     <data-content Name='Content'>
C:         <reply code='250' transID='1' />
C:     </data-content>
C: </data>
C: END
```

4.2 The Subscribe Operation

When an application wants to (periodically) receive the presence entry associated with an endpoint, it sends a "subscribe" element to the service.

The "subscribe" element has a "publisher" attribute, a "duration" attribute, a "transID" attribute, and no content:

- the "publisher" attribute specifies the endpoint associated with the presence entry;
- the "transID" attribute specifies the transaction-identifier associated with this operation; and,
- the "duration" attribute specifies the maximum number of seconds for which the originator is interested in receiving updated presence information.

When the service receives a "subscribe" element, we refer to the "publisher" attribute of that element as the "subject", and the service performs these steps:

1. If the subject is outside of this administrative domain, a "reply" element having code 553 is sent to the originator.

2. If the subject does not refer to a valid endpoint, a "reply" element having code 550 is sent to the originator.

3. If the subject's access entry does not contain a "presence:subscribe" token for the originator, a "reply" element having code 537 is sent to the originator.

4. If the originator already has an in-progress subscribe operation for the subject, then the previous subscribe operation is silently terminated, and processing continues.

5. If the "transID" attribute refers to an in-progress subscribe or watch operation for the originator, a "reply" element having code 555 is sent to the originator.

6. Otherwise:

    1. A "publish" element, corresponding to the subject's presence entry, is immediately sent to the originator.

    2. For each endpoint currently watching subscribers to the subject's presence information, a "notify" element is immediately as sent (c.f., Step 6.3 of Section 4.6).

    3. For up to the amount of time indicated by the "duration" attribute of the "subscribe" element, if the subject's presence entry changes, an updated "presence" element is sent to the originator using the publish operation (Section 4.4). Finally, when the amount of time indicated by the "duration" attribute expires, a terminate operation (Section 4.5) is sent to the originator.

    Note that if the duration is zero-valued, then the subscribe operation is making a one-time poll of the presence information. Accordingly, Step 6.3 above does not occur.

Regardless of whether a "publish" or "reply" element is sent to the originator, the "transID" attribute is identical to the value found in the "subscribe" element sent by the originator.

4.3 The Watch Operation

When an application wants to (periodically) receive notices about endpoints that are subscribed to receive presence information, it sends a "watch" element to the service.

The "watch" element has a "publisher" attribute, a "duration" attribute, a "transID" attribute, and no content:

- the "publisher" attribute specifies the endpoint associated with the presence entry;
- the "transID" attribute specifies the transaction-identifier associated with this operation; and,
- the "duration" attribute specifies the maximum number of seconds for which the originator is interested in watching subscribers.

When the service receives a "watch" element, we refer to the "publisher" attribute of that element as the "subject", and the service performs these steps:

1. If the subject is outside of this administrative domain, a "reply" element having code 553 is sent to the originator.

2. If the subject does not refer to a valid endpoint, a "reply" element having code 550 is sent to the originator.

3. If the subject's access entry does not contain a "presence:watch" token for the originator, a "reply" element having code 537 is sent to the originator.

4. If the originator already has an in-progress watch operation for the subject, then the previous watch operation is silently terminated, and processing continues.

5. If the "transID" attribute refers to an in-progress subscribe or watch operation for the originator, a "reply" element having code 555 is sent to the originator.

6. Otherwise:

    1. A "reply" element having code 250 is sent to the originator.

    2. For each endpoint currently subscribing to the subject's presence information, a "notify" element is immediately sent to the originator (c.f., Section 4.6). Finally, when the amount of time indicated by the "duration" attribute expires, a terminate operation (Section 4.5) is sent to the originator.

    3. For up to the amount of time indicated by the "duration" attribute of the "watch" element, whenever a subscribe operation succeeds, a "notify" element is sent to the originator. Finally, when the amount of time indicated by the "duration" attribute expires, the a terminate operation (Section 4.5) is sent to the originator.

    Note that if the duration is zero-valued, then the watch operation is making a one-time poll of the presence information. Accordingly, Step 6.3 above does not occur.

Regardless of whether a "notify" or "reply" element is sent to the originator, the "transID" attribute is identical to the value found in the "presence" element sent by the originator.

4.4 The Publish Operation

When an application wants to modify the presence entry associated with an endpoint, it sends a "publish" element to the service. In addition, the service sends a "publish" element to endpoints that have subscribed to see presence information (c.f., Section 4.2).

The "publish" element has a "publisher" attribute, a "transID" attribute, a "timeStamp" attribute, and contains a "presence" element:

- the "publisher" attribute specifies the endpoint to be associated with the presence entry;
- the "transID" attribute specifies the transaction-identifier associated with this operation;
- the "timeStamp" attribute specifies the application's notion of the current date and time; and,
- the "presence" element contains the desired presence entry for the endpoint.

When the service sends a "publish" element, the "transID" attribute specifies the transaction-identifier associated with the subscribe operation that caused this "publish" element to be sent, and the "timeStamp" attribute specifies the service's notion of the current date and time. No reply is sent by the receiving endpoint.

When the service receives a "publish" element, we refer to the "publisher" attribute of that element as the "subject", and the service performs these steps:

1. If the "publisher" attribute of the "publish" element doesn't match the "publisher" attribute of the "presence" element contained in the "publish" element, a "reply" element having code 503 is sent to the originator.

2. If the subject is outside of this administrative domain, a "reply" element having code 553 is sent to the originator.

3. If the subject does not refer to a valid endpoint, a "reply" element having code 550 is sent to the originator.

4. If the subject's access entry does not contain a "presence:publish" token for the originator, a "reply" element having code 537 is sent to the originator.

5. If the "lastUpdate" attribute of the "publish" element is not semantically identical to the "lastUpdate" attribute of the subject's presence entry, a "reply" element having code 555 is sent to the originator. (This allows a simple mechanism for atomic updates.)

6. Otherwise:

    1. The subject's presence entry is updated from the "publish" element.

    2. The "lastUpdate" attribute of the presence entry is set to the service's notion of the current date and time.

    3. A "reply" element having code 250 is sent to the originator.

When sending the "reply" element, the "transID" attribute is identical to the value found in the "publish" element sent by the originator.

4.5 The Terminate Operation

When an application no longer wishes to subscribe to presence information or to watch endpoints that are subscribed to receive presence information, it sends a "terminate" element to the service; similarly, when the service no longer considers an application to be subscribing or watching, a "terminate" element is sent to the application.

The "terminate" element contains only a "transID" attribute that specifies the transaction-identifier associated an in-progress subscribe or watch operation. Section 9.1 of [1] defines the syntax for the "terminate" element.

When the service receives a "terminate" element, it performs these steps:

1. If the transaction-identifier does not refer to a previous subscribe or watch operation for the originator, an "error" element having code 550 is returned.

2. Otherwise, the previous subscribe or watch operation for the originator is terminated, and a "reply" element having code 250 is sent to the originator.

Note that following a terminate operation, the originator may receive further presence or watcher updates. Although the service will send no further updates after processing a terminate operation and sending the reply operation, earlier updates may be in transit.

4.6 The Notify Operation

The service sends a "notify" element to endpoints that are watching other endpoints subscribed to presence information (c.f., Section 4.3).

The "notify" element has a "subscriber" attribute, a "transID" attribute, and no content:

- the "subscriber" attribute specifies the endpoint that is subscribed to presence information; and,
- the "transID" attribute specifies the transaction-identifier associated with the watch operation that caused this "notify" element to be sent. No reply is sent by the receiving endpoint.

4.7 The Reply Operation

While processing operations, the service may respond with a "reply" element. Consult Sections 10.2 and 6.1.2 of [1], respectively, for the syntax and semantics of the reply operation.

5. Registration: The Presence Service

Well-Known Endpoint: apex=presence

Syntax of Messages Exchanged: c.f., Section 6

Sequence of Messages Exchanged: c.f., Section 4

Access Control Tokens: presence:subscribe, presence:watch, presence:publish

Contact Information: c.f., the "Authors' Addresses" section of this memo

6. The Presence Service DTD

```
<!--
  DTD for the APEX presence service, as of 2000-12-12


  Refer to this DTD as:

    <!ENTITY % APEXPRESENCE PUBLIC "-//Blocks//DTD APEX PRESENCE//EN"
           "http://xml.resource.org/profiles/APEX/apex-presence.dtd">
    %APEXPRESENCE;
  -->


<!ENTITY % APEXCORE PUBLIC "-//Blocks//DTD APEX CORE//EN"
           "http://xml.resource.org/profiles/APEX/apex-core.dtd">
%APEXCORE;


<!--
  Synopsis of the APEX presence service

    service WKE: apex=presence

    message exchanges:

        consumer initiates    service replies
        ==================    ================
        subscribe             publish or reply
        terminate             reply
        watch                 reply
        publish               reply

        service initiates     consumer replies
        =================     ================
        terminate             (nothing)
        publish               (nothing)
        notify                (nothing)


    access control:

        token                 target
        ==================    ======
        presence:subscribe    for "publisher" of "subscribe" element
        presence:watch        for "publisher" of "watch" element
        presence:publish      for "publisher" of "publish" element
  -->
```

```
   <!ELEMENT subscribe   EMPTY>
   <!ATTLIST subscribe
             publisher   %ENDPOINT;        #REQUIRED
             transID     %UNIQID;          #REQUIRED
             duration    %SECONDS;         #REQUIRED>

   <!ELEMENT watch       EMPTY>
   <!ATTLIST watch
             publisher   %ENDPOINT;        #REQUIRED
             transID     %UNIQID;          #REQUIRED
             duration    %SECONDS;         #REQUIRED>


   <!-- publisher attributes must match in publish and presence -->
   <!ELEMENT publish     (presence)>
   <!ATTLIST publish
             publisher   %ENDPOINT;        #REQUIRED
             transID     %UNIQID;          #REQUIRED
             timeStamp   %TIMESTAMP;       #REQUIRED>

   <!ELEMENT notify      EMPTY>
   <!ATTLIST notify
             subscriber  %ENDPOINT;        #REQUIRED
             transID     %UNIQID;          #REQUIRED>


   <!--
     presence entries
     -->

   <!ELEMENT presence    (tuple+)>
   <!ATTLIST presence
             publisher   %ENDPOINT;        #REQUIRED
             lastUpdate  %TIMESTAMP;       #REQUIRED
             publisherInfo
                         %URI;             "">

   <!ELEMENT tuple       (capability*)>
   <!ATTLIST tuple
             destination %URI;             #REQUIRED
             availableUntil
                         %TIMESTAMP;       #REQUIRED
             tupleInfo %URI;               "">

   <!-- e.g., baseline='rfc2533' -->
   <!ELEMENT capability (#PCDATA)>
   <!ATTLIST capability
             baseline    NMTOKEN           #REQUIRED>
```

7. Security Considerations

Consult Section [1]'s Section 11 for a discussion of security issues.

References

[1] Rose, M.T., Klyne, G. and D.H. Crocker, "The Application Exchange Core", draft-mrose-apex-core-03 (work in progress), February 2001.

[2] Rose, M.T., "The Blocks Extensible Exchange Protocol Core", draft-ietf-beep-framework-11 (work in progress), January 2001.

[3] Rose, M.T., Klyne, G. and D.H. Crocker, "The APEX Access Service", draft-mrose-apex-access-02 (work in progress), December 2000.

[4] Allocchio, C., "GSTN Address Element Extensions in E-mail Services", RFC 2846, June 2000.

Authors' Addresses

Marshall T. Rose
Invisible Worlds, Inc.
1179 North McDowell Boulevard
Petaluma, CA 94954-6559
US

Phone: +1 707 789 3700
EMail: mrose@invisible.net
URI: http://invisible.net/

Graham Klyne
Content Technologies Limited
1220 Parkview
Arlington Business Park
Theale, Reading RG7 4SA
UK

Phone: +44 118 930 1300
EMail: gk@acm.org

David H. Crocker
Brandenburg Consulting
675 Spruce Drive
Sunnyvale, CA 94086
US

Phone: +1 408 246 8253
EMail: dcrocker@brandenburg.com
URI: http://www.brandenburg.com/

Appendix A. Acknowledgements

The authors gratefully acknowledge the contributions of: Neil Cook, Eric Dixon, Darren New, and Scott Pead.

Appendix B. Changes from IMXP

- s/IMXP/APEX/g
- Clarify the notion of co-residence for APEX services.
- Change data's originator from an attribute to an element.
- CPIM mapping moved to another document.

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.