Network Working Group
Request for Comments: 3341
Category: Standards Track

M. Rose
Dover Beach Consulting, Inc.
G. Klyne
Clearswift Corporation
D. Crocker
Brandenburg InternetWorking

July 2002

The Application Exchange (APEX) Access Service

Status of this Memo

Copyright Notice

Abstract

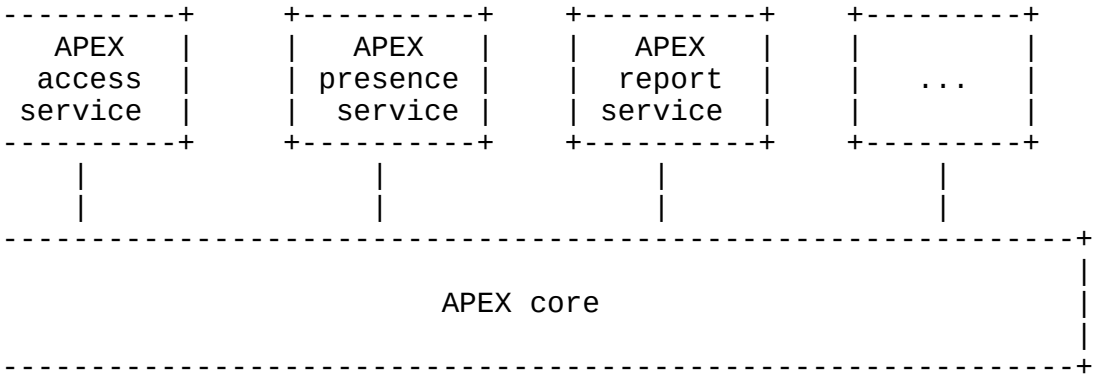
This memo describes the Application Exchange (APEX) access service, addressed as the well-known endpoint "apex=access". The access service is used to control use of both the APEX "relaying mesh" and other APEX services.

Table of Contents

1. Introduction

This memo describes an access service that is built upon the APEX [1] "relaying mesh". The APEX access service is used to control use of both the relaying mesh and other APEX services.

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

Note that within a single administrative domain, the relaying mesh makes use of the APEX access service in order to determine if an originator is allowed to transmit data to a recipient (c.f., Step 5.3 of Section 4.4.4.1 of [1]).

2. Use and Management of Access Information

Access information is organized around access entries, each of which contains:

- an owner: an APEX address with which the entry is associated;
- an actor: an APEX address that is granted permission to perform some action in the context of the owner;
- a list of actions; and,
- a timestamp indicating when the service last created or modified the access entry.

The access entry for a given owner controls access to a potentially large range of different APEX services, such as data delivery, access control, and presence information. In addition, Section 4.5 of [1] discusses APEX access policies that govern such activities as peer authentication, message relaying, and so on.

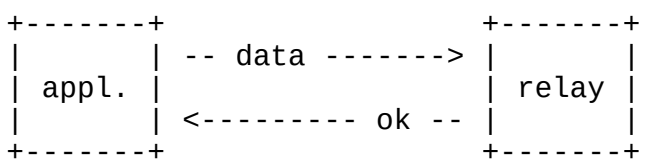
Management of access information falls into three categories:

- applications may query the access service to see if one or more actions are allowed;
- applications may retrieve access information associated with an owner/actor combination; and,
- applications may modify (i.e., create, replace, or delete) access information associated with an owner/actor combination.

Each is now described in turn.

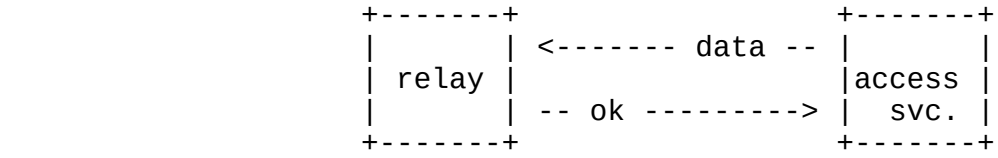
2.1 Querying Access Information

When an application wants to determine whether one or more actions are allowed for an owner/actor combination, it sends a "query" element to the service, e.g.,



```
C: <data content='#Content'>
       <originator identity='fred@example.com' />
       <recipient identity='apex=access@example.com' />
       <data-content Name='Content'>
           <query owner='fred@example.com' transID='1'
                  actor='barney@example.com'
                  actions='core:data presence:subscribe' />
       </data-content>
   </data>
S: <ok />
```

The service immediately responds with either an allow or deny operation containing the same transaction-identifier, where "allow" means that all of the actions listed in the query are permitted, e.g.,



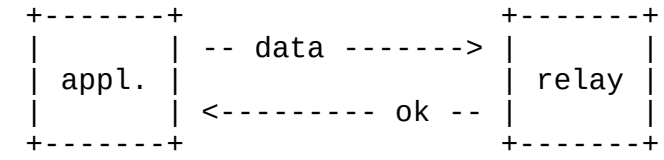

```
C: <data content='#Content'>
       <originator identity='apex=access@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'>
           <allow transID='1' />
       </data-content>
   </data>
S: <ok />
```

or

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

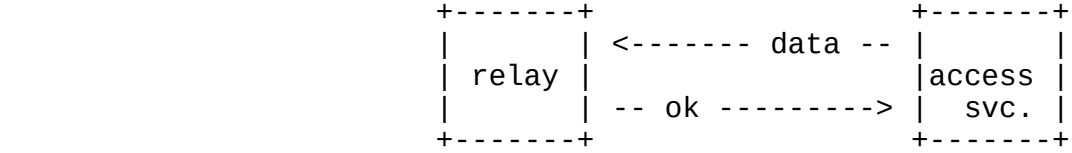
2.2 Retrieval of Access Information

When an application wants to retrieve the access entry associated with an owner/actor combination (typically in preparation for updating that access information), it sends a "get" element to the service, e.g.,




```
C: <data content='#Content'>
       <originator identity='fred@example.com' />
       <recipient identity='apex=access@example.com' />
       <data-content Name='Content'>
           <get transID='2'
                owner='fred@example.com'
                actor='*@example.com' />
       </data-content>
   </data>
S: <ok />
```

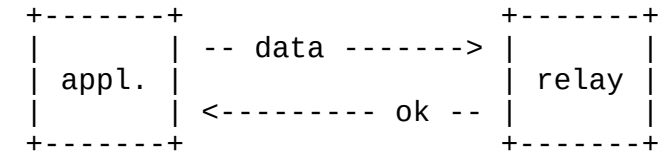
The service immediately responds with a set operation containing the access entry and the same transaction-identifier, e.g.,




```
C: <data content='#Content'>
       <originator identity='apex=access@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'>
           <set transID='2'>
               <access owner='fred@example.com'
                       actor='*@example.com'
                       actions='core:data presence:subscribe'
                       lastUpdate='2000-05-14T13:02:00-08:00' />
           </set>
       </data-content>
   </data>
S: <ok />
```

2.3 Update of Access Information

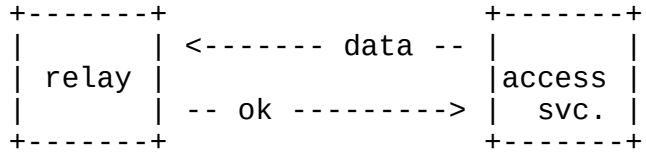
When an application wants to create or modify an access entry associated with an owner/actor combination, it sends a "set" element to the service containing the new access entry, e.g.,




```
C: <data content='#Content'>
       <originator identity='wilma@example.com' />
       <recipient identity='apex=access@example.com' />
       <data-content Name='Content'>
           <set transID='1'>
               <access owner='fred@example.com'
                       actor='*@example.com'
                       actions='core:data presence:subscribe'
                       lastUpdate='2000-05-14T13:02:00-08:00' />
           </set>
       </data-content>
   </data>
S: <ok />
```

Note that Step 4 of Section 4.4 requires that the "lastUpdate" attribute of an access entry be supplied in order to update that entry; accordingly, applications must successfully retrieve an access entry prior to trying to modify that entry. (Naturally, administrators should ensure that applications authorized to modify an access entry are also authorized to retrieve that entry.)

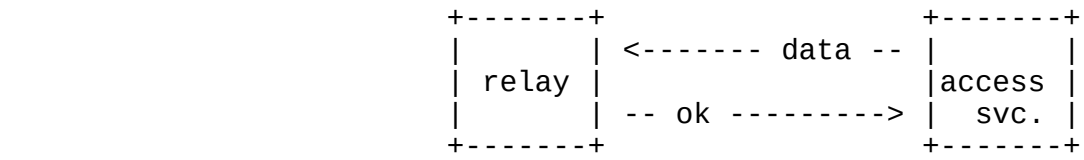
The service immediately responds with a reply operation containing the same transaction-identifier, e.g.,




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

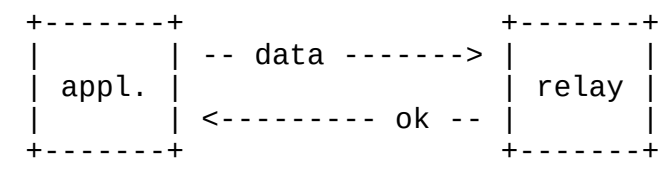
Note that Steps 6.2 and 9.2 of Section 4.4 require that the access service update the "lastUpdate" attribute of an access entry when it is created or modified.

The service also immediately sends a set operation to the owner attribute associated with the access entry, e.g.,



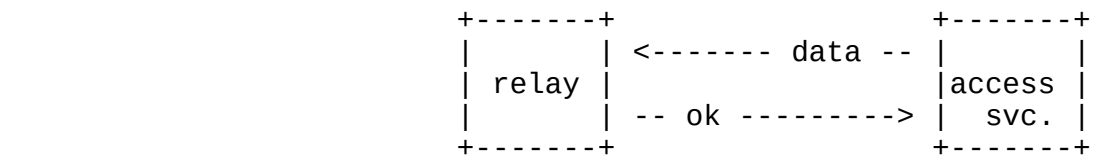

```
C: <data content='#Content'>
       <originator identity='apex=access@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'>
           <set transID='1'>
               <access owner='fred@example.com'
                       actor='*@example.com'
                       actions='core:data presence:subscribe'
                       lastUpdate='2000-05-14T23:02:00-08:00' />
           </set>
       </data-content>
   </data>
S: <ok />
```

When an application wants to delete the access entry associated with an owner/actor combination, it sends a "set" element to the service omitting the permitted actions, e.g.,



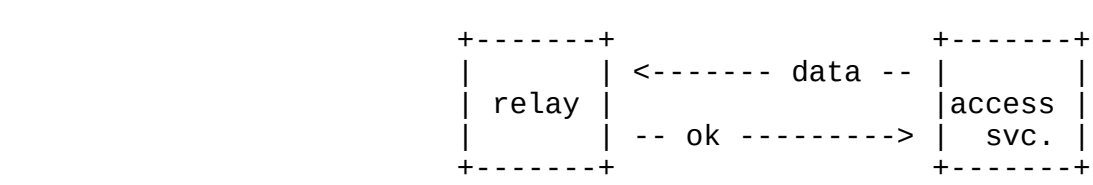

```
C: <data content='#Content'>
       <originator identity='wilma@example.com' />
       <recipient identity='apex=access@example.com' />
       <data-content Name='Content'>
           <set transID='2'>
               <access owner='fred@example.com'
                       actor='*@example.com'
                       lastUpdate='2000-05-14T13:02:00-08:00' />
           </set>
       </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=access@example.com' />
       <recipient identity='wilma@example.com' />
       <data-content Name='Content'>
           <reply code='250' transID='2' />
       </data-content>
   </data>
S: <ok />
```

The service also immediately sends a set operation to the owner attribute associated with the access entry, e.g.,




```
C: <data content='#Content'>
       <originator identity='apex=access@example.com' />
       <recipient identity='fred@example.com' />
       <data-content Name='Content'>
           <set transID='2'>
               <access owner='fred@example.com'
                       actor='*@example.com'
                       lastUpdate='2000-05-14T13:02:00-08:00' />
           </set>
       </data-content>
   </data>
S: <ok />
```

Because there are no actions associated with this access entry, the owner knows that the entry has been deleted.

Note that because access control supported limited wildcarding of actors, deleting an access entry for a particular owner/actor combination, may modify, rather than remove, permission. Because of this, a special action, "all:none", is used.

For example, consider these two access entries:

```
<access owner='fred@example.com'
        actor='barney@example.com'
        actions='core:data presence:subscribe presence:watch'
        lastUpdate='2000-05-14T13:20:00-08:00' />

<access owner='fred@example.com'
        actor='*@example.com'
        actions='core:data'
        lastUpdate='2000-05-14T13:20:00-08:00' />
```

Deleting the first access entry will not remove all permissions for for the actor "barney@example.com".

Instead, the first access entry should be modified thusly:

```
<access owner='fred@example.com'
        actor='barney@example.com'
        actions='all:none'
        lastUpdate='2000-05-14T13:20:00-08:00' />
```

3. Format of Access Entries

Each administrative domain is responsible for maintaining one or more "access entries" for each of its endpoints and associated subaddresses (regardless of whether those addresses are currently attached to the relaying mesh).

A separate access entry is required for each actor or group of actors for whom access permission is specified. Section 6 defines the syntax for access entries. Each access entry has an "owner" attribute, an "actor" attribute, an "actions" attribute, a "lastUpdate" attribute, and no content:

- the "owner" attribute specifies the address (endpoint or subaddress) associated with the access entry;
- the "actor" attribute specifies an entity or group of entities for whom access permissions are specified, as described below;
- the "actions" attribute specifies the permissions granted to the actor in the context of the owner; and,
- the "lastUpdate" attribute specifies the date and time that the service last created or modified the access entry.

An action is specified as a service/operation pair, e.g., the action "presence:publish" refers to the "publish" operation of the "presence" service. Two service values are reserved:

- "all" is used to refer to all services, e.g., "all:data"; and,
- "core" is used to refer to the service implemented by the relaying mesh, e.g., the "core:data" permission is consulted by the relaying mesh (c.f., Step 5.3 of Section 4.4.4.1 of [1]).

Further, two operation values are reserved:

- "all" is used to refer to all operations, e.g., "presence:all"; and,
- "none" is used to refer to no operations whatsoever, e.g., "all:none".

An actor is an APEX address and is specified using the "entity" syntax specified in Section 2.2 of [1]. However, both the "local" and "domain" parts may contain limited wildcarding:

- The "local" part is either:
  - a literal string (e.g., "fred");
  - a subaddress wildcard (e.g., "fred/*" or "apex=pubsub/*"); or,
  - the value "apex=*", specifying all APEX services;
  - the value "*", specifying any address other than an APEX service.
- The "domain" part is either:
  - a FQDN (e.g., "example.com");
  - a domain wildcard (e.g., "*.example.com"); or,
  - the value "*", specifying all administrative domains.

Note that in the case of a domain wildcard, the wildcard itself matches zero or more subdomains, e.g., "*.example.com" matches "example.com", "foo.example.com", "bar.foo.example.com", and so on.)

The following default entries are provided for each owner, but are overridden by an explicitly supplied entry with the same actor value:

```
actor='local@domain'  actions='all:all'
actor='apex=*@domain' actions='all:all'
actor='apex=*@*'      actions='core:data'
actor='*@*'           actions='all:none'
```

where "local@domain" specifies the owner associated with the access entry.

For example, the explicit entry

```
actor='*@*'           actions='core:data'
```

allows endpoints from any domain to use the relaying mesh to send data to the owner, but does not override the default entry for "apex=*@domain", which allows all APEX services in the owner's domain access to all actions.

APEX endpoint names can legitimately contain the character '*', but access entries use '*' to indicate wildcarding. Accordingly, the two-character sequence '*' is used to avoid ambiguity in the "actor" attribute. Similarly, to explicitly specify an endpoint name containing '\' in the "actor" attribute, the two-character sequence '\\' is used.

Note that this convention is used only for the "actor" attribute of the "get" operation and of the "access" entry that appears in the "set" operation; however, this convention is not used in the "query" operation, as this operation does not allow wildcarding.

For example, to specify the endpoint named as "a\b*c@example.com" in the "get" operation or in an "access" entry, the string "a\\b*c@example.com" is used; but in the "query" operation, the string "a\b*c@example.com" is used. (Of course, as name allocation is a local matter, these complications can be avoided by the simple expedient of not using endpoint names containing '*' or '\'.)

3.1 Finding the Appropriate Entry: Matching Owners and Actors

The use of actor wildcarding makes it possible for several access entries to apply for a given owner/actor combination. When determining which access entry to use when responding to the query operation, the algorithm is:

- Consider only those access entries that are associated with the given owner.

- Consider only those access entries in which the actor value matches the actor address in the query. If the wildcard character ('*') is present, then it a match is possible only if each wildcard character can be replaced with a non-empty character sequence (one or more characters) to obtain a value identical to the address in the query.

- Order those remaining access entries:

  * Use the exactness of the match with the domain part of the actor value as the primary key; and,

  * Use the exactness of the match with the local part of the actor value as the secondary key.

- When matching with the domain part, an exact match is the best match; otherwise, the shorter the wildcard match, the higher the priority.

  For example, if the actor's domain is "bar.foo.example.com", a match against an entry of "*.foo.example.com" is better than a match against an entry of "*.example.com".

- When matching with the local part, an exact match is the best match; otherwise, the shorter the wildcard match, the higher the priority. This is true regardless of whether the wildcarding is for subaddress or service. (Note that a local part with a wildcard subaddress does not have a non-empty match with the same local part without a subaddress.)

For example, consider these access entries:

```
<access owner='fred@example.com'
        actor='wilma@example.com'
        actions='all:all'
        lastUpdate='2000-05-14T13:20:00-08:00' />
<access owner='fred@example.com'
        actor='mr.slate@example.com'
        actions='core:data'
        lastUpdate='2000-05-14T13:20:00-08:00' />
<access owner='fred/appl=wb@example.com'
        actor='barney/appl=wb@example.com'
        actions='core:data'
        lastUpdate='2000-05-14T13:20:00-08:00' />
<access owner='fred@example.com'
        actor='*@example.com'
        actions='core:data presence:subscribe presence:watch'
        lastUpdate='2000-05-14T13:20:00-08:00' />
```

```
<access owner='fred@example.com'
        actor='*@*'
        actions='core:data'
        lastUpdate='2000-05-14T13:20:00-08:00' />
```

Briefly:

- For addresses within the "example.com" administrative domain:

  * "fred", "wilma", and all APEX services within the "example.com" administrative domain are allowed access to all operations for "fred@example.com";

  * "mr.slate" is allowed access only to send data through the relaying mesh to "fred@example.com";

  * "barney/appl=wb" is allowed access only to send data to "fred/appl=wb", a subaddress of "fred@example.com"; and,

  * any other address within the "example.com" administrative domain is allowed access to send data and invoke the "subscribe" and "watch" operations of the APEX presence service with respect to "fred@example.com".

- For any address outside the "example.com" administrative domain, the address is allowed access to send data, regardless of whether it is an APEX service.

Note that although the four default entries are always available, the explicit entry for actor "*@*" overrides the corresponding default entry.

3.2 Creating and Updating Access Entries

The get and set operations are provided as a basic mechanism for creating and updating access rules, for which no special wildcard processing is performed.

The actor value for an access entry may contain limited wildcard characters which have special significance only when performing a query operation (cf., Section 3.1). For the purposes of retrieving and updating entries, actor values are treated simply as literal names.

4. The Access Service

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

- Within an administrative domain, the service is addressed using the well-known endpoint of "apex=access".
- Section 6 defines the syntax of the operations exchanged with the service.
- A consumer of the service initiates communications by sending data containing a query, get, or set operation.
- The service replies to these operations.
- When an access entry is changed, the service sends a notification to the owner associated with the changed entry.

An implementation of the service must maintain information about access entries 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='fred@example.com' />
    <recipient identity='apex=access@example.com' />
</data>
```

where "..." refers to:

```
<query owner='fred@example.com' transID='1'
       actor='barney@example.com'
       actions='core:data presence:subscribe' />
```

then the corresponding BEEP message might look like this:

```
C: MSG 1 2 . 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=access@example.com' />
C: </data>
C: --boundary
C: Content-Type: application/beep+xml
C: Content-ID: <2@example.com>
C:
C: <query owner='fred@example.com' transID='1'
C:        actor='barney@example.com'
C:        actions='core:data presence:subscribe' />
C: --boundary--
C: END
```

or this:

```
C: MSG 1 1 . 42 267
C: Content-Type: application/beep+xml
C:
C: <data content='#Content'>
C:     <originator identity='fred@example.com' />
C:     <recipient identity='apex=access@example.com' />
C:     <data-content Name='Content'>
C:         <query owner='fred@example.com' transID='1'
C:                actor='barney@example.com'
C:                actions='core:data presence:subscribe' />
C:     </data-content>
C: </data>
C: END
```

4.2 The Query Operation

When an application wants to see if a particular operation is allowed, it sends a "query" element to the service.

The "query" element has an "owner" attribute, an "actor" attribute, an "actions" attribute, a "transID" attribute, and no content:

- the "owner" attribute specifies the address associated with the access entry;
- the "actor" attribute specifies the address (without wildcarding) for which access permissions are queried;
- the "actions" attribute specifies one or more actions for which permission is queried; and,
- the "transID" attribute specifies the transaction-identifier associated with this operation.

When the service receives a "query" element, we refer to the "owner" attribute as the "subject". The service performs these steps:

1. If the subject is outside this administrative domain, a "reply" element having code 553 is sent to the originator.
2. If the subject does not refer to a valid address, a "reply" element having code 550 is sent to the originator.
3. If the subject's access entry matching the originator does not contain an "access:query" token, a "reply" element having code 537 is sent to the originator.

4. The subject's access entry matching the actor attribute of the query element is selected (cf., Section 3.1).

5. If all of the permissions in the "actions" attribute of the query element are contained in the selected access entry, then an "allow" element is sent to the originator.

6. Otherwise, a "deny" element is sent to the originator.

Regardless of whether an "allow", "deny", or "reply" element is sent to the originator, the "transID" attribute is identical to the value found in the "query" element sent by the originator.

4.3 The Get Operation

Prior to creating or updating an access entry for some owner/actor combination, an application will usually need to retrieve any existing access entry. It does so by sending a "get" element to the service. In particular, a successful response returns a "lastUpdate" value that is necessary when sending a subsequent "set" element.

The "get" element has an "owner" attribute, an "actor" attribute, a "transID" attribute, and no content:

- the "owner" attribute specifies the address associated with the access entry;

- the "actor" attribute specifies the address (with possible wildcarding) for which access permissions are retrieved; and,

- the "transID" attribute specifies the transaction-identifier associated with this operation.

When the service receives a "get" element, we refer to the "owner" attribute as the "subject". The service performs these steps:

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

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

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

4. The subject's access entry whose "actor" attribute identically matches the "actor" attribute of the "get" element is selected.

5. If no such entry exists, a "reply" element having code 551 is sent to the originator.

6. Otherwise, a "set" element corresponding to the selected access entry is sent to the originator.

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

4.4 The Set Operation

When an application wants to modify (i.e., create, replace, or delete) the access entry associated with an owner/actor combination, it sends a "set" element to the service.

The "set" element has a "transID" attribute, and contains an "access" element:

- the "transID" attribute specifies the transaction-identifier associated with this operation; and,

- the "access" element contains the access entry to be created, replaced, or deleted.

The "access" element has an "owner" attribute, an "actor" attribute, an optional "actions" attribute, an optional "lastUpdate" attribute, and no content:

- the "owner" attribute specifies the address associated with the access entry;

- the "actor" attribute specifies the address (with possible wildcarding) for which access permissions are specified;

- the "actions" attribute (present only to add or replace an entry) specifies one or more actions for which permission is to be determined; and,

- the "lastUpdate" attribute (present only to replace or delete an entry) specifies the current timestamp of the access entry that is to be replaced.

When the service receives a "set" element, we refer to the "owner" attribute of the access element as the "subject". The service performs these steps:

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

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

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

4. The subject's access entry whose "actor" attribute identically matches the "actor" attribute of the "set" element is selected.

5. If no such entry exists and the "lastUpdate" attribute is present in the supplied "set" element, a "reply" element having code 555 is sent to the originator.

6. If no such entry exists and the "lastUpdate" attribute is absent in the supplied "set" element, then:

   1. The access entry for the owner/actor combination is created from the supplied "access" element.

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

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

   4. A "set" element corresponding to the newly-created access entry is sent to the subject's address.

7. If the selected entry exists, but its "lastUpdate" attribute is not semantically identical to the "lastUpdate" attribute of the supplied "access" element, a "reply" element having code 555 is sent to the originator.

8. If "actions" attribute of the supplied "access" element is not present, then:

   1. The selected entry is deleted.

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

3. A "set" element corresponding to the owner/actor combination, but lacking an "actions" attribute is sent to the subject's address.

9. Otherwise:

1. The access entry for the owner/actor combination is updated from the supplied "access" element.

2. The "lastUpdate" attribute of the updated access entry is set to the service's notion of the current date and time (which should be different from the "lastUpdate" value associated with any replaced entry).

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

4. A "set" element corresponding to the newly-updated access entry is sent to the subject's address.

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

4.5 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 definition and an exposition of the syntax of the reply element.

5. Registration: The Access Service

Well-Known Endpoint: apex=access

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

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

Access Control Tokens: access:query, access:get, access:set

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

6. The Access Service DTD

```
<!--
  DTD for the APEX access service, as of 2001-06-19

  Refer to this DTD as:

    <!ENTITY % APEXACCESS PUBLIC "-//IETF//DTD APEX ACCESS//EN" "">
    %APEXACCESS;
  -->

<!ENTITY % APEXCORE PUBLIC "-//IETF//DTD APEX CORE//EN" "">
%APEXCORE;

<!--
  DTD data types:

      entity        syntax/reference     example
      ======        ================     =======
    access actor
      ACTOR         an ENDPOINT or a     *@example.com
                    wildcard

    permitted actions
      ACTIONS       a list of access     "core:any access:query"
                    tokens
  -->

<!ENTITY  % ACTOR   "CDATA">
<!ENTITY  % ACTIONS "NMTOKENS">

<!--
  Synopsis of the APEX access service

    service WKE: apex=access

    message exchanges:

        consumer initiates    service replies
        ==================    ================
        query                 allow, deny, or reply
        get                   set or reply
        set                   reply

        service initiates     consumer replies
        =================     ================
        set                   (nothing)
```

```
      access control:

          token                target
          ==========           ======
          access:query         for "owner" of "access" element
          access:get           for "owner" of "access" element
          access:set           for "owner" of "access" element
    -->

  <!ELEMENT query       EMPTY>
  <!ATTLIST query
            owner       %ENDPOINT;        #REQUIRED
            actor       %ACTOR;           #REQUIRED
            actions     %ACTIONS;         #REQUIRED
            transID     %UNIQID;          #REQUIRED>

  <!ELEMENT get         EMPTY>
  <!ATTLIST get
            owner       %ENDPOINT;        #REQUIRED
            actor       %ACTOR;           #REQUIRED
            transID     %UNIQID;          #REQUIRED>

  <!ELEMENT set         (access)>
  <!ATTLIST set
            transID     %UNIQID;          #REQUIRED>

  <!ELEMENT allow       EMPTY>
  <!ATTLIST allow
            transID     %UNIQID;          #REQUIRED>

  <!ELEMENT deny        EMPTY>
  <!ATTLIST deny
            transID     %UNIQID;          #REQUIRED>

  <!--
    access entries
    -->

  <!ELEMENT access      EMPTY>
  <!ATTLIST access
            owner       %ENDPOINT;        #REQUIRED
            actor       %ACTOR;           #REQUIRED
            actions     %ACTIONS;         #IMPLIED
            lastUpdate  %TIMESTAMP;       #IMPLIED>
```

7. Security Considerations

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

In addition, timestamps issued by the the access service may disclose location information. If this information is considered sensitive, the special timezone value "-00:00" may be used (after converting the local time accordingly).

References

[1] Rose, M., Klyne, G. and D. Crocker, "The Application Exchange Core", RFC 3340, July 2002.

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

Authors' Addresses

Marshall T. Rose
Dover Beach Consulting, Inc.
POB 255268
Sacramento, CA 95865-5268
US

Phone: +1 916 483 8878
EMail: mrose@dbc.mtview.ca.us

Graham Klyne
Clearswift Corporation
1310 Waterside
Arlington Business Park
Theale, Reading RG7 4SA
UK

Phone: +44 11 8903 8903
EMail: Graham.Klyne@MIMEsweeper.com

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, Darren New, Chris Newman, Scott Pead, and Bob Wyman.

Full Copyright Statement

Acknowledgement

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