Network Working Group   J. Rosenberg
Request for Comments: 4826   Cisco
Category: Standards Track   May 2007

                   Extensible Markup Language (XML)
                 Formats for Representing Resource Lists

Status of This Memo

Copyright Notice

Abstract

   In multimedia communications, presence, and instant messaging
   systems, there is a need to define Uniform Resource Identifiers
   (URIs) that represent services that are associated with a group of
   users. One example is a resource list service. If a user sends a
   Session Initiation Protocol (SIP) SUBSCRIBE message to the URI
   representing the resource list service, the server will obtain the
   state of the users in the associated group, and provide it to the
   sender. To facilitate definition of these services, this
   specification defines two Extensible Markup Language (XML) documents.
   One document contains service URIs, along with their service
   definition and a reference to the associated group of users. The
   second document contains the user lists that are referenced from the
   first. This list of users can be utilized by other applications and
   services. Both documents can be created and managed with the XML
   Configuration Access Protocol (XCAP).

Table of Contents

1. Introduction

   The Session Initiation Protocol (SIP) [4] defines the SIP Uniform
   Resource Identifier (URI) as any resource to which a SIP request can
   be generated for the purposes of establishing some form of
   communications operation. These URIs can represent users (for
   example, sip:joe@example.com). The SIP URI can also represent a
   service, such as voicemail, conferencing, or a presence list. A
   common pattern across such SIP services is that the service is
   defined, and associated with a URI. In order to operate, that
   service needs to make use of a list of users (or, more generally, a
   list of resources). When a SIP request is sent to the service URI,
   the server providing the service reads that list, and then performs
   some kind of operation against each resource on the list. This is
   shown in Figure 1.

```
                                /---\
                                |   |
                                \---/ Resource
                          +----|   |   List
                          |    |   |
                          |    \---/
                          |
                          |
                          |
                          |
                          V
             +------------+
             |            | -------->
             |    SIP     |
   ---------------> |  Service   | -------->
         service |            |
         URI     |            | -------->
             +------------+
```
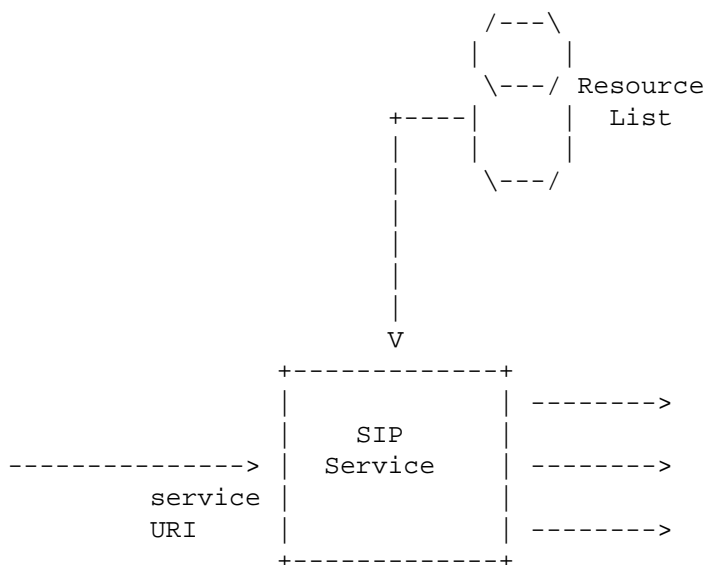
                              Figure 1

   One important example of such a service is a presence [11] list
   service. A presence list service allows a client to generate a SIP
   SUBSCRIBE request to ask for presence information for a list of
   users. The presence list server obtains the presence for the users
   on the list and provides them back to the client. A presence list

server is a specific case of a resource list server (RLS) [14], which
allows a client to generate a SIP SUBSCRIBE request to ask for
notifications of SIP events for a list of resources.

Another example of such a service is an instant conference service.
If a client sends a SIP INVITE request to the URI representing the
instance conference service, the conference server will create a
conference call containing the client and the associated group of
users.

It is very useful for a user of these systems to define the groups of
users or resources (generally called a resource list) separately from
the services that access those resource lists. Indeed, there are
usages for resource lists even in the absence of any associated
network-based service. As an example, rather than use a presence
list service, a client might generate individual SUBSCRIBE requests
to obtain the presence of each user in a locally stored presence
list. In such a case, there is a need for a format for storing the
list locally on disk. Furthermore, the user might wish to share the
list with friends, and desire to email it to those friends. This
also requires a standardized format for the resource list.

As such, this document defines two Extensible Markup Language (XML)
document formats. The first is used to represent resource lists,
independent of any particular service. The second is used to define
service URIs for an RLS, and to associate a resource list with the
service URI. This document also defines an XML Configuration Access
Protocol (XCAP) [10] application usage for managing each of these two
documents.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED",
"SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY",
and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and
indicate requirement levels for compliant implementations.

3. Resource Lists Documents

3.1. Structure

A resource lists document is an XML [2] document that MUST be well-
formed and MUST be valid according to schemas, including extension
schemas, available to the validater and applicable to the XML
document. Resource lists documents MUST be based on XML 1.0 and MUST
be encoded using UTF-8. This specification makes use of XML
namespaces for identifying resource lists documents and document
fragments. The namespace URI for elements defined by this

specification is a URN [3] that uses the namespace identifier 'ietf'
defined by RFC 2648 [6] and extended by RFC 3688 [8]. This URN is:

    urn:ietf:params:xml:ns:resource-lists

A resource lists document has the <resource-lists> element as the
root element of the document. This element has no attributes. Its
content is a sequence of zero or more <list> elements, each of which
defines a single resource list.

Each <list> element can contain an optional "name" attribute. This
attribute is a handle for the list. When present, it MUST be unique
amongst all other <list> elements within the same parent element.
The <list> element may also contain attributes from other namespaces,
for the purposes of extensibility.

Each <list> element is composed of an optional display name, a
sequence of zero or more elements, each of which may be an <entry>
element, a <list> element, an <entry-ref> element, or an <external>
element, followed by any number of elements from other namespaces,
for the purposes of extensibility. The ability of a <list> element
to contain other <list> elements means that a resource list can be
hierarchically structured. The <display-name> then allows for a
human-friendly name to be associated with each level in the
hierarchy. An <entry> element describes a single resource, defined
by a URI, that is part of the list. An <entry-ref> element allows an
entry in a document within the same XCAP root to be included by
reference, rather than by value. An <external> element contains a
reference to a list stored on this or another server.

The <entry> element describes a single resource. The <entry> element
has a single mandatory attribute, "uri". This attribute is equal to
the URI that is used to access the resource. The resource list
format itself does not constrain the type of URI that can be used.
However, the service making use of the resource list may require
specific URI schemes. For example, RLS services will require URIs
that represent subscribeable resources. This includes the SIP and
pres [15] URIs. The "uri" attribute MUST be unique amongst all other
"uri" attributes in <entry> elements within the same parent.
Uniqueness is determined by case-sensitive string comparisons. As
such, it is possible that two "uri" attributes will have the same URI
when compared using the functional equality rules defined for that
URI scheme, but different ones when compared using case sensitive
string comparison. The <entry> element can also contain attributes
from other namespaces for the purposes of extensibility.

The <entry> element contains a sequence of elements that provide
information about the entry. Only one such element is defined at

this time, which is <display-name>. This element provides a UTF-8-
encoded string, meant for consumption by a human user, that describes
the resource. Unlike the "name" attribute of the <entry> element,
the <display-name> has no uniqueness requirements. The <display-
name> element can contain the "xml:lang" attribute, which provides
the language of the display name. The <entry> element can contain
other elements from other namespaces. This is meant to support the
inclusion of other information about the entry, such as a phone
number or postal address.

The <entry-ref> element allows an entry to be included in the list by
reference, rather than by value. This element is only meaningful
when the document was obtained through XCAP. In such a case, the
referenced entry has to exist within the same XCAP root. The <entry>
element has a single mandatory attribute, "ref". The "ref" attribute
MUST be unique amongst all other "ref" attributes in <entry-ref>
elements within the same parent. Uniqueness is determined by case
sensitive string comparisons. The <entry-ref> element also allows
attributes from other namespaces, for the purposes of extensibility.
The content of an <entry-ref> element is an optional display name,
followed by any number of elements from other namespaces, for the
purposes of extensibility. The display name is useful for providing
a localized nickname as an alternative to the name defined in the
<entry> to which the <entry-ref> refers.

The content of the "ref" attribute is a relative HTTP URI [7].
Specifically, it MUST be a relative path reference, where the base
URI is equal to the XCAP root URI of the document in which the
<entry-ref> appears. This relative URI, if resolved into an absolute
URI according to the procedures in RFC 3986, MUST resolve to an
<entry> element within a resource-lists document. For example,
suppose that an <entry> element within a specific XCAP root was
identified by the following HTTP URI:

http://xcap.example.com/resource-lists/users/sip:bill@example.com/
index/~~/resource-lists/list%5b@name=%22list1%22%5d/
entry%5b@uri=%22sip:petri@example.com%22%5d

If http://xcap.example.com is the XCAP root URI, then an <entry-ref>
element pointing to this entry would have the following form:

<entry-ref ref="resource-lists/users/sip:bill@example.com/
index/~~/resource-lists/list%5b@name=%22list1%22%5d/
entry%5b@uri=%22sip:petri@example.com%22%5d"/>

Note that line folding within the HTTP URI and XML attribute above
are for the purposes of readability only. Also note that, as
described in RFC 3986, the relative path URI does not begin with the

"/". Since the relative URI used within the "ref" attribute must be
a relative path URI, the "/" will never be present as the first
character within the content of a "ref" attribute. Since the content
of the "ref" attribute is a valid HTTP URI, it must be percent-
encoded within the XML document.

The <external> element is similar to the <entry-ref> element. Like
<entry-ref>, it is only meaningful in documents obtained from an XCAP
server. It too is a reference to content stored elsewhere. However,
it refers to an entire list, and furthermore, it allows that list to
be present on another server. The <external> element has a single
mandatory attribute, "anchor", which specifies the external list by
means of an absolute HTTP URI. The "anchor" attribute MUST be unique
amongst all other "anchor" attributes in <external> elements within
the same parent. Uniqueness is determined by case-sensitive string
comparisons. The <external> element can also contain attributes from
other namespaces, for the purposes of extensibility. The content of
an <external> element is an optional <display-name> followed by any
number of elements from another namespace, for the purposes of
extensibility. The value of the "anchor" attribute MUST be an
absolute HTTP URI. This URI MUST identify an XCAP resource, and in
particular, it MUST represent a <list> element within a resource
lists document. The URI MUST be percent-encoded.

For both the <entry-ref> and <external> elements, the responsibility
of resolving their references falls upon the entity that is making
use of the document. When the document is used in conjunction with
XCAP, this means that the burden falls on the XCAP client. If the
XCAP client is a PC-based application using the resource-lists
document as a presence list, the references would likely be resolved
upon explicit request by the user. They can, of course, be resolved
at any time. If the XCAP client is an RLS itself, the references
would be resolved when the RLS receives a SUBSCRIBE request for an
RLS service associated with a resource list that contains one of
these references (see below). An XCAP server defined by this
specification will not attempt to resolve the references before
returning the document to the client. Similarly, if, due to network
errors or some other problem, the references cannot be resolved, the
handling is specific to the usage of the document. For resource
lists being used by RLS services, the handling is discussed below.

3.2. Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:resource-lists"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="urn:ietf:params:xml:ns:resource-lists"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
 schemaLocation="http://www.w3.org/2001/xml.xsd"/>
 <xs:complexType name="listType">
  <xs:sequence>
   <xs:element name="display-name" type="display-nameType"
    minOccurs="0"/>
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
     <xs:element name="list">
      <xs:complexType>
       <xs:complexContent>
        <xs:extension base="listType"/>
       </xs:complexContent>
      </xs:complexType>
     </xs:element>
     <xs:element name="external" type="externalType"/>
     <xs:element name="entry" type="entryType"/>
     <xs:element name="entry-ref" type="entry-refType"/>
    </xs:choice>
   </xs:sequence>
   <xs:any namespace="##other" processContents="lax" minOccurs="0"
    maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="optional"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
 </xs:complexType>
 <xs:complexType name="entryType">
  <xs:sequence>
   <xs:element name="display-name" minOccurs="0">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="display-nameType"/>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:any namespace="##other" processContents="lax" minOccurs="0"
    maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="uri" type="xs:anyURI" use="required"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
 </xs:complexType>
```

```
       <xs:complexType name="entry-refType">
        <xs:sequence>
         <xs:element name="display-name" type="display-nameType"
          minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ref" type="xs:anyURI" use="required"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
       </xs:complexType>
       <xs:complexType name="externalType">
        <xs:sequence>
         <xs:element name="display-name" type="display-nameType"
          minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="anchor" type="xs:anyURI"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
       </xs:complexType>
       <xs:element name="resource-lists">
        <xs:complexType>
         <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:element name="list" type="listType"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:complexType name="display-nameType">
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute ref="xml:lang"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:schema>
```

3.3. Example Document

   The following is an example of a document compliant to the schema.
   All line feeds within element content are for display purposes only.

```
   <?xml version="1.0" encoding="UTF-8"?>
   <resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <list name="friends">
     <entry uri="sip:bill@example.com">
      <display-name>Bill Doe</display-name>
     </entry>
```

```
      <entry-ref ref="resource-lists/users/sip:bill@example.com/index/~~/
       resource-lists/list%5b@name=%22list1%22%5d/entry%5b@uri=%22sip:pet
       ri@example.com%22%5d"/>
      <list name="close-friends">
       <display-name>Close Friends</display-name>
       <entry uri="sip:joe@example.com">
        <display-name>Joe Smith</display-name>
       </entry>
       <entry uri="sip:nancy@example.com">
        <display-name>Nancy Gross</display-name>
       </entry>
       <external anchor="http://xcap.example.org/resource-lists/users/
        sip:a@example.org/index/~~/resource-lists/list%5b@name=%22mkti
        ng%22%5d">
         <display-name>Marketing</display-name>
        </external>
      </list>
     </list>
    </resource-lists>
```

3.4. Usage with XCAP

   Resource lists documents can be manipulated with XCAP. This section
   provides the details necessary for such a usage.

3.4.1. Application Unique ID

   XCAP requires application usages to define an application unique ID
   (AUID) in either the IETF tree or a vendor tree. This specification
   defines the "resource-lists" AUID within the IETF tree, via the IANA
   registration in Section 8.

3.4.2. MIME Type

   The MIME type for this document is "application/resource-lists+xml".

3.4.3. XML Schema

   The XML Schema for this document is defined as the sole content of
   Section 3.2.

3.4.4. Default Namespace

   The default namespace used in expanding URIs is
   urn:ietf:params:xml:ns:resource-lists.

3.4.5. Additional Constraints

   In addition to the schema, there are constraints on the values
   present in the "name" attribute of the <list> element, the "uri"
   attribute of the <external> element, the "ref" attribute of the
   <entry-ref> element, and the "anchor" attribute of the <external>
   element. These constraints are defined in Section 3.1. Some of
   these constraints are enforced by the XCAP server. Those constraints
   are:

   o The "name" attribute in a <list> element MUST be unique amongst
      all other "name" attributes of <list> elements within the same
      parent element. Uniqueness is determined by case-sensitive string
      comparison.

   o The "uri" attribute in a <entry> element MUST be unique amongst
      all other "uri" attributes of <entry> elements within the same
      parent element. Uniqueness is determined by case-sensitive string
      comparison.

   o The URI in the "ref" attribute of the <entry-ref> element MUST be
      unique amongst all other "ref" attributes of <entry-ref> elements
      within the same parent element. Uniqueness is determined by case-
      sensitive string comparison. The value of the attribute MUST be a
      relative path reference. Note that the server is not responsible
      for verifying that the reference resolves to an <entry> element in
      a document within the same XCAP root.

   o The URI in the "anchor" attribute of the <external> element MUST
      be unique amongst all other "anchor" attributes of <external>
      elements within the same parent element. Uniqueness is determined
      by case-sensitive string comparison. The value of the attribute
      MUST be an absolute HTTP URI. Note that the server is not
      responsible for verifying that the URI resolves to a <list>
      element in a document. Indeed, since the URI may reference a
      server in another domain, referential integrity cannot be
      guaranteed without adding substantial complexity to the system.

3.4.6. Data Semantics

   Semantics for the document content are provided in Section 3.1.

3.4.7. Naming Conventions

   Resource lists documents are usually identified as references from
   other application usages. For example, an RLS services document
   contains a reference to the resource list it uses.

Frequently, an XCAP client will wish to insert or remove an <entry>,
<entry-ref>, or <external> element from a document without having a
cached copy of that document. In such a case, the "uri" attribute of
the <entry> element, the "ref" attribute of the <entry-ref> element,
or the "anchor" attribute of the <external> element is used as an
index to select the element to operate upon. The XCAP server will
determine uniqueness by case-sensitive string comparison. However,
each of these attributes contain URIs, and the URI equality rules for
their schemes may allow two URIs to be the same, even if they are
different by case sensitive string comparison. As such, it is
possible that a client will attempt a PUT or DELETE in an attempt to
modify or remove an existing element. Instead, the PUT ends up
inserting a new element, or the DELETE ends up returning an error
response.

If the XCAP client cannot determine whether the user intent is to
create or replace, the client SHOULD canonicalize the URI before
performing the operation. For a SIP URI (often present in the "uri"
attribute of the <entry> element), this canonicalization procedure is
defined in Section 5. We expect that the SIP URIs that will be
placed into resource lists documents will usually be of the form
sip:user@domain, and possibly include a user parameter. The
canonicalization rules work perfectly for these URIs.

For HTTP URIs, a basic canonicalization algorithm is as follows. If
the port in the URI is equal to the default port (80 for http URIs),
then the port is removed. The hostname is converted to all
lowercase. Any percent-encoding in the URI for characters which do
not need to be percent-encoded is removed. A character needs to be
percent-encoded when it is not permitted in that part of the URI
based on the grammar for that part of the URI.

3.4.8. Resource Interdependencies

   There are no resource interdependencies identified by this
   application usage.

3.4.9. Authorization Policies

   This application usage does not modify the default XCAP authorization
   policy, which is that only a user can read, write, or modify their
   own documents. A server can allow privileged users to modify
   documents that they don't own, but the establishment and indication
   of such policies is outside the scope of this document. It is
   anticipated that a future application usage will define which users
   are allowed to modify a list resource.

4. RLS Services Documents

4.1. Structure

   An RLS services document is used to define URIs that represent
   services provided by a Resource List Server (RLS) as defined in [14].
   An RLS services document is an XML [2] document that MUST be well-
   formed and MUST be valid according to schemas, including extension
   schemas, available to the validater and applicable to the XML
   document. RLS services documents MUST be based on XML 1.0 and MUST
   be encoded using UTF-8. This specification makes use of XML
   namespaces for identifying RLS services documents and document
   fragments. The namespace URI for elements defined by this
   specification is a URN [3] that uses the namespace identifier 'ietf'
   defined by RFC 2648 [6] and extended by RFC 3688 [8]. This URN is:

      urn:ietf:params:xml:ns:rls-services

   The root element of an rls-services document is <rls-services>. It
   contains a sequence of <service> elements, each of which defines a
   service available at an RLS.

   Each <service> element has a single mandatory attribute, "uri". This
   URI defines the resource associated with the service. That is, if a
   client subscribes to that URI, they will obtain the service defined
   by the corresponding <service> element. The <service> element can
   also contain attributes from other namespaces, for the purposes of
   extensibility. The <service> element contains child elements that
   define the service. For an RLS service, very little service
   definition is needed: just the resource list to which the server will
   perform virtual subscriptions [14] and the set of event packages that
   the service supports. The former can be conveyed in one of two ways.
   There can be a <resource-list> element, which points to a <list>
   element in a resource-lists document, or there can be a <list>
   element, which includes the resource list directly. The supported
   packages are contained in the <packages> element. The <service>
   element can also contain elements from other namespaces, for the
   purposes of extensibility.

   By including the contents of the resource list directly, a user can
   create lists and add members to them with a single XCAP operation.
   However, the resulting list becomes "hidden" within the RLS service
   definition, and is not usable by other application usages. For this
   reason, the <resource-list> element exists as an alternative. It can
   reference a <list> element in a resource-lists document. Since the
   list is separated from the service definition, it can be easily
   reused by other application usages.

   The <list> element is of the list type defined by the schema for
   resource lists. It is discussed in Section 3.1.

   The <resource-list> element contains a URI. This element is only
   meaningful when the document was obtained through XCAP. The URI MUST
   be an absolute HTTP URI representing an XCAP element resource. Its
   XCAP root MUST be the same as the XCAP root of the RLS services
   document. When the RLS services document is present in a user's home
   directory, the HTTP URI MUST exist underneath that user's home
   directory in the resource-lists application usage. When the RLS
   services document is in the global directory, the HTTP URI MUST exist
   underneath any user's home directory in the resource-lists
   application usage. In either case, the element referenced by the URI
   MUST be a <list> element within a resource-lists document. All of
   these constraints except for the latter one (which is a referential
   integrity constraint) will be enforced by the XCAP server.

   The <packages> element contains a sequence of <package> elements.
   The content of each <package> element is the name of a SIP event
   package [13]. The <packages> element may also contain elements from
   additional namespaces, for the purposes of extensibility. The
   <packages> element is optional. When it is not present, it means
   that the RLS service will accept subscriptions for any event package.

4.2. Schema

   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema targetNamespace="urn:ietf:params:xml:ns:rls-services"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:ietf:params:xml:ns:rls-services"
    xmlns:rl="urn:ietf:params:xml:ns:resource-lists"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="rls-services">
     <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:element name="service" type="serviceType"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:complexType name="serviceType">
     <xs:sequence>
      <xs:choice>
       <xs:element name="resource-list" type="xs:anyURI"/>
       <xs:element name="list" type="rl:listType"/>
      </xs:choice>
      <xs:element name="packages" type="packagesType" minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
       maxOccurs="unbounded"/>

```
      </xs:sequence>
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
     </xs:complexType>
     <xs:complexType name="packagesType">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:element name="package" type="packageType"/>
       <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
     <xs:simpleType name="packageType">
      <xs:restriction base="xs:string"/>
     </xs:simpleType>
    </xs:schema>
```

4.3. Example Document

   This document shows two services. One is sip:mybuddies@example.com,
   and the other is sip:marketing@example.com. The former service
   references a resource list in a resource-lists document, and the
   latter one includes a list locally. Both services are for the
   presence event package only.

```
   <?xml version="1.0" encoding="UTF-8"?>
   <rls-services xmlns="urn:ietf:params:xml:ns:rls-services"
       xmlns:rl="urn:ietf:params:xml:ns:resource-lists"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <service uri="sip:mybuddies@example.com">
     <resource-list>http://xcap.example.com/resource-lists/user
      s/sip:joe@example.com/index/~~/resource-lists/list%5b@nam
      e=%22l1%22%5d</resource-list>
     <packages>
      <package>presence</package>
     </packages>
    </service>
    <service uri="sip:marketing@example.com">
      <list name="marketing">
        <rl:entry uri="sip:joe@example.com"/>
        <rl:entry uri="sip:sudhir@example.com"/>
      </list>
      <packages>
        <package>presence</package>
      </packages>
    </service>
   </rls-services>
```

4.4. Usage with XCAP

   RLS services documents can be manipulated with XCAP. This section
   provides the details necessary for such a usage.

4.4.1. Application Unique ID

   XCAP requires application usages to define an application unique ID
   ID (AUID) in either the IETF tree or a vendor tree. This
   specification defines the "rls-services" AUID within the IETF tree,
   via the IANA registration in Section 8.

4.4.2. MIME Type

   The MIME type for this document is "application/rls-services+xml".

4.4.3. XML Schema

   The XML Schema for this document is defined as the sole content of
   Section 4.2.

4.4.4. Default Namespace

   The default namespace used in expanding URIs is
   urn:ietf:params:xml:ns:rls-services.

4.4.5. Additional Constraints

   In addition to the schema, there are constraints on the URIs present
   in the <service> and <resource-list> elements. These constraints are
   defined in Section 3.1. Some of these constraints are enforced by
   the XCAP server. Those constraints are:

   o The URI in the "uri" attribute of the <service> element MUST be
      unique amongst all other URIs in "uri" elements in any <service>
      element in any document on a particular server. This uniqueness
      constraint spans across XCAP roots. Furthermore, the URI MUST NOT
      correspond to an existing resource within the domain of the URI.
      If a server is asked to set the URI to something that already
      exists, the server MUST reject the request with a 409, and use the
      mechanisms defined in [10] to suggest alternate URIs that have not
      yet been allocated.

   o The URI in a <resource-list> element MUST be an absolute URI. The
      server MUST verify that the URI path contains "resource-lists" in
      the path segment corresponding to the AUID. If the RLS services
      document is within the XCAP user tree (as opposed to the global
      tree), the server MUST verify that the XUI in the path is the same

      as the XUI in the URI of to the RLS services document. These
      checks are made by examining the URI value, as opposed to
      dereferencing the URI. The server is not responsible for
      verifying that the URI actually points to a <list> element within
      a valid resource lists document.

   o In addition, an RLS services document can contain a <list>
      element, which in turn can contain <entry>, <entry-ref>, <list>,
      and <external> elements. The constraints defined for these
      elements in Section 3.4.7 MUST be enforced.

   o In some cases, an XCAP client will wish to create a new RLS
      service, and wish to assign it a "vanity URI", such as
      sip:friends@example.com. However, the client does not know
      whether this URI meets the uniqueness constraints defined above.
      In that case, it can simply attempt the creation operation, and if
      the result is a 409 that contains a detailed conflict report with
      the <uniqueness-failure> element, the client knows that the URI
      could not be assigned. It can then retry with a different vanity
      URI, or use one of the suggestions in the detailed conflict
      report.

   o If the client wishes to create a new RLS service, and it doesn't
      care what the URI is, the client creates a random one, and
      attempts the creation operation. As discussed in [10], if this
      should fail with a uniqueness conflict, the client can retry with
      different URIs with increasing randomness.

4.4.6. Data Semantics

   Semantics for the document content are provided in Section 4.1.

4.4.7. Naming Conventions

   Typically, there are two distinct XCAP clients that access RLS
   services documents. The first is a client acting on behalf of the
   end user in the system. This client edits and writes both resource
   lists and RLS services documents as they are created or modified by
   the end user. The other XCAP client is the RLS itself, which reads
   the RLS services documents in order to process SUBSCRIBE requests.

   To make it easier for an RLS to find the <service> element for a
   particular URI, the XCAP server maintains, within the global tree, a
   single RLS services document representing the union of all the
   <service> elements across all documents created by all users within
   the same XCAP root. There is a single instance of this document, and
   its name is "index". Thus, if the root services URI is

   http://xcap.example.com, the following is the URI that an RLS would
   use to fetch this index:

   http://xcap.example.com/rls-services/global/index

   As discussed below, this index is created from all the documents in
   the user tree that have the name "index" as well. An implication of
   this is that a client operating on behalf of a user SHOULD define its
   RLS services within the document named "index". If the root services
   URI is http://xcap.example.com, for user "sip:joe@example.com" the
   URI for this document would be:

   http://xcap.example.com/rls-services/users/sip:joe@example.com/index

   If a client elects to define RLS services in a different document,
   this document will not be "picked up" in the global index, and
   therefore, will not be used as an RLS service.

4.4.8. Resource Interdependencies

   As with other application usages, the XML schema and the XCAP
   resource naming conventions describe most of the resource
   interdependencies applicable to this application usage.

   This application usage defines an additional resource interdependence
   between a single document in the global tree and all documents in the
   user tree with the name "index". This global document is formed as
   the union of all of the index documents for all users within the same
   XCAP root. In this case, the union operation implies that each
   <service> element in a user document will also be present as a
   <service> element in the global document. The inverse is true as
   well. Every <service> element in the global document exists within a
   user document within the same XCAP root.

   As an example, consider the RLS services document for user
   sip:joe@example.com:

   <?xml version="1.0" encoding="UTF-8"?>
   <rls-services>
    <service uri="sip:mybuddies@example.com">
     <resource-list>http://xcap.example.com/resource-lists/users/si
      p:joe@example.com/index/~~/resource-lists/list%5b@name=%22l1%
      22%5d</resource-list>
     <packages>
      <package>presence</package>
     </packages>
    </service>
   </rls-services>

   And consider the RLS services document for user bob:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <rls-services>
    <service uri="sip:marketing@example.com">
      <list name="marketing">
        <rl:entry uri="sip:joe@example.com"/>
        <rl:entry uri="sip:sudhir@example.com"/>
      </list>
      <packages>
        <package>presence</package>
      </packages>
    </service>
   </rls-services>
   ```

   The global document at
   http://xcap.example.com/rls-services/global/index would look like
   this:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <rls-services xmlns="urn:ietf:params:xml:ns:rls-services"
      xmlns:rl="urn:ietf:params:xml:ns:resource-lists"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <service uri="sip:mybuddies@example.com">
     <resource-list>http://xcap.example.com/resource-lists/user
     s/sip:joe@example.com/index/~~/resource-lists/list%5b@nam
     e=%22l1%22%5d</resource-list>
     <packages>
     <package>presence</package>
     </packages>
    </service>
    <service uri="sip:marketing@example.com">
      <list name="marketing">
        <rl:entry uri="sip:joe@example.com"/>
        <rl:entry uri="sip:sudhir@example.com"/>
      </list>
      <packages>
        <package>presence</package>
      </packages>
    </service>
   </rls-services>
   ```

   Requests made against the global document MUST generate responses
   that reflect the most recent state of all the relevant user
   documents. This requirement does not imply that the server must
   actually store this global document. It is anticipated that most
   systems will dynamically construct the responses to any particular
   request against the document resource.

The uniqueness constraint on the "uri" attribute of <service> will
ensure that no two <service> elements in the global document have the
same value of that attribute.

4.4.9. Authorization Policies

This application usage does not modify the default XCAP authorization
policy, which is that only a user can read, write, or modify their
own documents. A server can allow privileged users to modify
documents that they don't own, but the establishment and indication
of such policies are outside the scope of this document. It is
anticipated that a future application usage will define which users
are allowed to modify an RLS services document.

The index document maintained in the global tree represents sensitive
information, as it contains the union of all the information for all
users on the server. As such, its access MUST be restricted to
trusted elements within domain of the server. Typically, this would
be limited to the RLSs that need access to this document.

4.5. Usage of an RLS Services Document by an RLS

This section discusses how an RLS, on receipt of a SUBSCRIBE request,
uses XCAP and the RLS services document to guide its operation.

When an RLS receives a SUBSCRIBE request for a URI (present in the
Request URI), it obtains the <service> element whose uri attribute
matches (based on URI equality) the URI in the SUBSCRIBE request.
This document makes no normative statements on how this might be
accomplished. The following paragraph provides one possible
approach.

The RLS canonicalizes the Request URI as described in Section 5. It
then performs an XCAP GET operation against the URI formed by
combining the XCAP root with the document selector of the global
index with a node selector of the form "rls-services/
service[@uri=<canonical-uri>]", where <canonical-uri> is the
canonicalized version of the Request URI. If the response is a 200
OK, it will contain the service definition for that URI.

Once the <service> element has been obtained, it is examined. If the
<packages> element is present, and the event package in the SUBSCRIBE
request is not amongst those listed in the <package> elements within
<packages>, the request MUST be rejected with a 489 (Bad Event)
response code, as described in [13]. Otherwise, it SHOULD be
processed. The next step is to authorize that the client is allowed
to subscribe to the resource. This can be done using the data
defined in [12], for example. Assuming the subscriber is authorized

to subscribe to that resource, the subscription is processed
according to the procedures defined in [14]. This processing
requires the RLS to compute a flat list of URIs that are to be
subscribed to. If the <service> element had a <list> element, it is
extracted. If the <service> element had a <resource-list> element,
its URI content is dereferenced. The result should be a <list>
element. If it is not, the request SHOULD be rejected with a 502
(Bad Gateway). Otherwise, that <list> element is extracted.

At this point, the RLS has a <list> element in its possession. The
next step is to obtain a flat list of URIs from this element. To do
that, it traverses the tree of elements rooted in the <list> element.
Before traversal begins, the RLS initializes two lists: the "flat
list", which will contain the flat list of the URI after traversal,
and the "traversed list", which contains a list of HTTP URIs in
<external> elements that have already been visited. Both lists are
initially empty. Next, tree traversal begins. A server can use any
tree-traversal ordering it likes, such as depth-first search or
breadth-first search. The processing at each element in the tree
depends on the name of the element:

o If the element is <entry>, the URI in the "uri" attribute of the
   element is added to the flat list if it is not already present
   (based on case-sensitive string equality) in that list, and the
   URI scheme represents one that can be used to service
   subscriptions, such as SIP [4] and pres [15].

o If the element is an <entry-ref>, the relative path reference
   making up the value of the "ref" attribute is resolved into an
   absolute URI. This is done using the procedures defined in
   Section 5.2 of RFC 3986 [7], using the XCAP root of the RLS
   services document as the base URI. This absolute URI is resolved.
   If the result is not a 200 OK containing a <entry> element, the
   SUBSCRIBE request SHOULD be rejected with a 502 (Bad Gateway).
   Otherwise, the <entry> element returned is processed as described
   in the previous step.

o If the element is an <external> element, the absolute URI making
   up the value of the "anchor" attribute of the element is examined.
   If the URI is on the traversed list, the server MUST cease
   traversing the tree, and SHOULD reject the SUBSCRIBE request with
   a 502 (Bad Gateway). If the URI is not on the traversed list, the
   server adds the URI to the traversed list, and dereferences the
   URI. If the result is not a 200 OK containing a <list> element,
   the SUBSCRIBE request SHOULD be rejected with a 502 (Bad Gateway).
   Otherwise, the RLS replaces the <external> element in its local
   copy of the tree with the <list> element that was returned, and
   tree traversal continues.

Because the <external> element is used to dynamically construct the
tree, there is a possibility of recursive evaluation of references.
The traversed list is used to prevent this from happening.

Once the tree has been traversed, the RLS can create virtual
subscriptions to each URI in the flat list, as defined in [14]. In
the processing steps outlined above, when an <entry-ref> or
<external> element contains a reference that cannot be resolved,
failing the request is at SHOULD strength. In some cases, an RLS may
provide better service by creating virtual subscriptions to the URIs
in the flat list that could be obtained, omitting those that could
not. Only in those cases should the SHOULD recommendation be
ignored.

5. SIP URI Canonicalization

   This section provides a technique for URI canonicalization. This
   canonicalization produces a URI that, in most cases, is equal to the
   original URI (where equality is based on the URI comparison rules in
   RFC 3261). Furthermore, the canonicalized URI will usually be
   lexically equivalent to the canonicalized version of any other URI
   equal to the original.

   To canonicalize the URI, the following steps are followed:

   1. First, the domain part of the URI is converted into all
       lowercase, and any tokens (such as "user" or "transport" or
       "udp") are converted to all lowercase.

   2. Secondly, any percent-encoding in the URI for characters which do
       not need to be percent-encoded is removed. A character needs to
       be percent-encoded when it is not permitted in that part of the
       URI based on the grammar for that part of the URI. For example,
       if a SIP URI is sip:%6aoe%20smith@example.com, it is changed to
       sip:joe%20smith@example.com. In the original URI, the character
       'j' was percent-encoded. This is allowed, but not required,
       since the grammar allows a 'j' to appear in the user part. As a
       result, it appears as 'j' after this step of canonicalization.

   3. Thirdly, any URI parameters are reordered so that they appear in
       lexical order based on parameter name. The ordering of a
       character is determined by the US-ASCII numerical value of that
       character, with smaller numbers coming first. Parameters are
       ordered with the leftmost character as most significant. For
       parameters that contain only letters, this is equivalent to an
       alphabetical ordering.

   4. Finally, any header parameters are discarded. This canonicalized
       URI is used instead of the original URI.

   If two URIs, A and B, are functionally equal (meaning that they are
   equal according to the URI comparison rules in RFC 3261), their
   canonicalized URIs are equal under case-sensitive string comparison
   if the following are true:

   o Neither URI contains header parameters.

   o If one of the URI contains a URI parameter not defined in RFC
      3261, the other does as well.

6. Extensibility

   Resource-lists and RLS services documents are meant to be extended.
   An extension takes place by defining a new set of elements in a new
   namespace, governed by a new schema. Every extension MUST have an
   appropriate XML namespace assigned to it. The XML namespace of the
   extension MUST be different from the namespaces defined in this
   specification. The extension MUST NOT change the syntax or semantics
   of the schemas defined in this document. All XML tags and attributes
   that are part of the extension MUST be appropriately qualified so as
   to place them within that namespace.

   This specification defines explicit places where new elements or
   attributes from an extension can be placed. These are explicitly
   indicated in the schemas by the <any> and <anyAttribute> elements.
   Extensions to this specification MUST specify where their elements
   can be placed within the document.

   As a result, a document that contains extensions will require
   multiple schemas in order to determine its validity: a schema defined
   in this document, along with those defined by extensions present in
   the document. Because extensions occur by adding new elements and
   attributes governed by new schemas, the schemas defined in this
   document are fixed and would only be changed by a revision to this
   specification. Such a revision, should it take place, would endeavor
   to allow documents compliant to the previous schema to remain
   compliant to the new one. As a result, the schemas defined here
   don't provide explicit schema versions, as this is not expected to be
   needed.

7. Security Considerations

   The information contained in rls-services and resource-lists
   documents are particularly sensitive. It represents the principle
   set of people with whom a user would like to communicate. As a
   result, clients SHOULD use TLS when contacting servers in order to
   fetch this information. Note that this does not represent a change
   in requirement strength from XCAP.

8. IANA Considerations

   There are several IANA considerations associated with this
   specification.

8.1. XCAP Application Unique IDs

   This section registers two new XCAP Application Unique IDs (AUIDs)
   according to the IANA procedures defined in [10].

8.1.1. resource-lists

   Name of the AUID: resource-lists

   Description: A resource lists application is any application that
      needs access to a list of resources, identified by a URI, to which
      operations, such as subscriptions, can be applied.

8.1.2. rls-services

   Name of the AUID: rls-services

   Description: A Resource List Server (RLS) services application is a
      Session Initiation Protocol (SIP) application whereby a server
      receives SIP SUBSCRIBE requests for resource, and generates
      subscriptions towards a resource list.

8.2. MIME Type Registrations

   This specification requests the registration of two new MIME types
   according to the procedures of RFC 4288 [9] and guidelines in RFC
   3023 [5].

8.2.1. application/resource-lists+xml

   MIME media type name: application

   MIME subtype name: resource-lists+xml

   Mandatory parameters: none

   Optional parameters: Same as charset parameter application/xml as
      specified in RFC 3023 [5].

   Encoding considerations: Same as encoding considerations of
      application/xml as specified in RFC 3023 [5].

   Security considerations: See Section 10 of RFC 3023 [5] and
      Section 7 of RFC 4826.

   Interoperability considerations: none

   Published specification: RFC 4826

   Applications that use this media type: This document type has been
      used to support subscriptions to lists of users [14] for SIP-based
      presence [11].

   Additional Information:

        Magic Number: none

        File Extension: .rl

        Macintosh file type code: "TEXT"

   Personal and email address for further information:
      Jonathan Rosenberg, jdrosen@jdrosen.net

   Intended usage: COMMON

   Author/Change controller: The IETF.

8.2.2. application/rls-services+xml

   MIME media type name: application

   MIME subtype name: rls-services+xml

   Mandatory parameters: none

   Optional parameters: Same as charset parameter application/xml as
      specified in RFC 3023 [5].

   Encoding considerations: Same as encoding considerations of
      application/xml as specified in RFC 3023 [5].

   Security considerations: See Section 10 of RFC 3023 [5] and
      Section 7 of RFC 4826.

   Interoperability considerations: none

   Published specification: RFC 4826

   Applications that use this media type: This document type has been
      used to support subscriptions to lists of users [14] for SIP-based
      presence [11].

   Additional Information:

        Magic Number: none

        File Extension: .rs

        Macintosh file type code: "TEXT"

   Personal and email address for further information:
      Jonathan Rosenberg, jdrosen@jdrosen.net

   Intended usage: COMMON

   Author/Change controller: The IETF.

8.3. URN Sub-Namespace Registrations

   This section registers two new XML namespaces, as per the guidelines
   in RFC 3688 [8].

8.3.1. urn:ietf:params:xml:ns:resource-lists

   URI: The URI for this namespace is
      urn:ietf:params:xml:ns:resource-lists.

   Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org),
      Jonathan Rosenberg (jdrosen@jdrosen.net).

    XML:
          BEGIN
          <?xml version="1.0"?>
          <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
             "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
          <html xmlns="http://www.w3.org/1999/xhtml">
          <head>
            <meta http-equiv="content-type"
              content="text/html;charset=iso-8859-1"/>
            <title>Resource Lists Namespace</title>
          </head>
          <body>
            <h1>Namespace for Resource Lists</h1>
            <h2>urn:ietf:params:xml:ns:resource-lists</h2>
            <p>See
               RFC4826.</p>
          </body>
          </html>
          END

8.3.2. urn:ietf:params:xml:ns:rls-services

   URI: The URI for this namespace is
      urn:ietf:params:xml:ns:rls-services.

   Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org),
      Jonathan Rosenberg (jdrosen@jdrosen.net).

   XML:
         BEGIN
         <?xml version="1.0"?>
         <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
            "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
         <html xmlns="http://www.w3.org/1999/xhtml">
         <head>
           <meta http-equiv="content-type"
             content="text/html;charset=iso-8859-1"/>
           <title>Resource List Server (RLS) Services Namespace</title>
         </head>
         <body>
           <h1>Namespace for Resource List Server (RLS) Services</h1>
           <h2>urn:ietf:params:xml:ns:rls-services</h2>
           <p>See
             RFC4826.</p>
         </body>
         </html>
         END

8.4. Schema Registrations

   This section registers two XML schemas per the procedures in [8].

8.4.1. urn:ietf:params:xml:schema:resource-lists

   URI: urn:ietf:params:xml:schema:resource-lists

   Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org),
      Jonathan Rosenberg (jdrosen@jdrosen.net).

   The XML for this schema can be found as the sole content of
   Section 3.2.

8.4.2. urn:ietf:params:xml:schema:rls-services

   URI: urn:ietf:params:xml:schema:rls-services

   Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org),
      Jonathan Rosenberg (jdrosen@jdrosen.net).

   The XML for this schema can be found as the sole content of
   Section 4.2.

9. Acknowledgements

   The authors would like to thank Hisham Khartabil, Jari Urpalainen,
   and Spencer Dawkins for their comments and input. Thanks to Ted
   Hardie for his encouragement and support of this work.

10. References

10.1. Normative References

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

   [2]   Paoli, J., Maler, E., Bray, T., and C. Sperberg-McQueen,
          "Extensible Markup Language (XML) 1.0 (Second Edition)", World
          Wide Web Consortium FirstEdition REC-xml-20001006,
          October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>.

   [3]   Moats, R., "URN Syntax", RFC 2141, May 1997.

   [4]   Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.,
          Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP:
          Session Initiation Protocol", RFC 3261, June 2002.

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

   [6]   Moats, R., "A URN Namespace for IETF Documents", RFC 2648,
          August 1999.

   [7]   Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
          Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986,
          January 2005.

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

   [9]   Freed, N. and J. Klensin, "Media Type Specifications and
         Registration Procedures", BCP 13, RFC 4288, December 2005.

   [10] Rosenberg, J., "The Extensible Markup Language (XML)
         Configuration Access Protocol (XCAP)", RFC 4825, May 2007.

10.2. Informative References

   [11] Rosenberg, J., "A Presence Event Package for the Session
         Initiation Protocol (SIP)", RFC 3856, August 2004.

   [12] Rosenberg, J., "Presence Authorization Rules", Work
         in Progress, October 2006.

   [13] Roach, A., "Session Initiation Protocol (SIP)-Specific Event
         Notification", RFC 3265, June 2002.

   [14] Roach, A., Rosenberg, J., and B. Campbell, "A Session
         Initiation Protocol (SIP) Event Notification Extension for
         Resource Lists", RFC 4662, January 2005.

   [15] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859,
         August 2004.

Author's Address

   Jonathan Rosenberg
   Cisco
   Edison, NJ
   US

   EMail: jdrosen@cisco.com
   URI: http://www.jdrosen.net

Full Copyright Statement

Intellectual Property

Acknowledgement