

Network Working Group  
Request for Comments: 4662  
Category: Standards Track

A. B. Roach  
B. Campbell  
Estacado Systems  
J. Rosenberg  
Cisco Systems  
August 2006

## A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists

### Status of This Memo

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

### Copyright Notice

Copyright (C) The Internet Society (2006).

### Abstract

This document presents an extension to the Session Initiation Protocol (SIP)-Specific Event Notification mechanism for subscribing to a homogeneous list of resources. Instead of sending a SUBSCRIBE for each resource individually, the subscriber can subscribe to an entire list and then receive notifications when the state of any of the resources in the list changes.

## Table of Contents

1. Introduction .....	3
2. Terminology .....	4
3. Overview of Operation .....	4
4. Operation of List Subscriptions .....	5
4.1. Negotiation of Support for Resource Lists .....	6
4.2. Subscription Duration .....	7
4.3. NOTIFY Bodies .....	7
4.4. RLS Processing of SUBSCRIBE Requests .....	7
4.5. RLS Generation of NOTIFY Requests .....	7
4.6. Subscriber Processing of NOTIFY Requests .....	9
4.7. Handling of Forked Requests .....	10
4.8. Rate of Notifications .....	10
5. Using multipart/related to Convey Aggregate State .....	10
5.1. XML Syntax .....	11
5.2. List Attributes .....	13
5.3. Resource Attributes .....	14
5.4. Name Attributes .....	14
5.5. Instance Attributes .....	14
5.6. Constructing Coherent Resource State .....	16
5.6.1. Processing Full State Notifications .....	17
5.6.2. Processing Partial State Notifications .....	17
6. Example .....	18
7. Security Considerations .....	31
7.1. Authentication .....	31
7.1.1. RLS and Subscriber in the Same Domain .....	31
7.1.2. RLS and Subscriber in Different Domains .....	32
7.2. Risks of Improper Aggregation .....	33
7.3. Signing and Sealing .....	33
7.4. Infinite Loops .....	34
8. IANA Considerations .....	34
8.1. New SIP Option Tag: eventlist .....	34
8.2. New MIME type for Resource List Meta-Information .....	34
8.3. URN Sub-Namespace .....	35
9. Acknowledgements .....	36
10. References .....	36
10.1. Normative References .....	36
10.2. Informative References .....	37

## 1. Introduction

The SIP-specific event notification mechanism [2] allows a user (the subscriber) to request to be notified of changes in the state of a particular resource. This is accomplished by the subscriber generating a SUBSCRIBE request for the resource, which is processed by a notifier that represents the resource.

In many cases, a subscriber has a list of resources they are interested in. Without some aggregating mechanism, this will require the subscriber to generate a SUBSCRIBE request for each resource about which they want information. For environments in which bandwidth is limited, such as wireless networks, subscribing to each resource individually is problematic. Some specific problems are:

- o Doing so generates substantial message traffic, in the form of the initial SUBSCRIBE requests for each resource and the refreshes of each individual subscription.
- o The notifier may insist on low refresh intervals, in order to avoid a long-lived subscription state. This means that the subscriber may need to generate SUBSCRIBE refreshes faster than it would like to or has the capacity to.
- o The notifier may generate NOTIFY requests more rapidly than the subscriber desires, causing NOTIFY traffic at a greater volume than is desired by the subscriber.

To solve these problems, this specification defines an extension to RFC 3265 [2] that allows for requesting and conveying notifications for lists of resources. A resource list is identified by a URI, and it represents a list of zero or more URIs. Each of these URIs is an identifier for an individual resource for which the subscriber wants to receive information. In many cases, the URI used to identify the resource list will be a SIP URI [1]; however, the use of other schemes (such as pres: [10]) is also foreseen.

The notifier for the list is called a "resource list server", or RLS. In order to determine the state of the entire list, the RLS will act as if it has generated a subscription to each resource in the list.

The resource list is not restricted to be inside the domain of the subscriber. Similarly, the resources in the list are not constrained to be in the domain of the resource list server.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [5].

The following terms are used throughout the remainder of this document.

**Back-End Subscription:** Any subscription (SIP or otherwise) that an RLS creates to learn of the state of a resource. An RLS will create back-end subscriptions to learn of the state of a resource about which the RLS is not an authority. For back-end subscriptions, RLSes act as a subscriber.

**List Subscription:** A subscription to a resource list. In list subscriptions, RLSes act as the notifier.

**Resource:** A resource is any logical entity that has a state or states that can be subscribed to. Resources are identified by URIs.

**Resource List:** A list of zero or more resources that can have their individual states subscribed to with a single subscription.

**RLMI:** Resource List Meta-Information. RLMI is a document that describes the state of the virtual subscriptions associated with a list subscription.

**RLS:** Resource List Server. RLSes accept subscriptions to resource lists and send notifications to update subscribers of the state of the resources in a resource list.

**Virtual Subscription:** A Virtual Subscription is a logical construct within an RLS that represents subscriptions to the resources in a resource list. For each list subscription it services, an RLS creates at least one virtual subscription for every resource in the resource list being subscribed to. In some cases, such as when the RLS is not the authority for the state of the resource, this virtual subscription will be associated with a back-end subscription. In other cases, such as when the RLS is the authority for the state of the resource, the virtual subscription will not have a corresponding back-end subscription.

## 3. Overview of Operation

This section provides an overview of the typical mode of operation of this extension. It is not normative.

When users wish to subscribe to the resource of a list of resources, they can use the mechanisms described in this specification. The first step is the creation of a resource list. This resource list is represented by a SIP URI. The list contains a set of URIs, each of which represents a resource for which the subscriber wants to receive information. The resource list can exist in any domain. The list could be manipulated through a web page, through a voice response system, or through some other protocol. The specific means by which the list is created and maintained is outside the scope of this specification.

To learn the resource state of the set of elements on the list, the user sends a single SUBSCRIBE request targeted to the URI of the list. This will be routed to an RLS for that URI. The RLS acts as a notifier, authenticates the subscriber, and accepts the subscription.

The RLS may have direct information about some or all of the resources specified by the list. If it does not, it could subscribe to any non-local resources specified by the list resource.

Note that subscriptions to non-local resources may or may not be SIP subscriptions; any mechanism for determining such information may be employed. This document uses the term "back-end subscription" to refer to such a subscription, regardless of whether SIP is used to establish and service it.

As the state of resources in the list change, the RLS generates notifications to the list subscribers. The RLS can, at its discretion, buffer notifications of resource changes and send the resource information to the subscriber in batches, rather than individually. This allows the RLS to provide rate limiting for the subscriber.

The list notifications contain a body of type multipart/related. The root section of the multipart/related content is an XML document that provides meta-information about each resource present in the list. The remaining sections contain the actual state information for each resource.

#### 4. Operation of List Subscriptions

The event list extension acts, in many ways, like an event template package. In particular, any single list subscription must be homogeneous with respect to the underlying event package. In other words, a single list subscription can apply only one event package to all the resources in the resource list.

Note that it is perfectly valid for an RLS to allow multiple subscriptions to the same list to use differing event packages.

The key difference between a list subscription and templates in general is that support for list subscriptions indicates support for arbitrary nesting of list subscriptions. In other words, elements within the list may be atomic elements, or they may be lists themselves.

The consequence of this is that subscription to a URI that represents a list actually results in several virtual subscriptions to a tree of resources. The leaf nodes of this tree are virtual subscriptions of the event type given in the "Event" header field; all other nodes in the tree are list subscriptions that are serviced as described in this section and its subsections.

Keep in mind that these virtual subscriptions are not literal SIP subscriptions (although they may result in SIP subscriptions, depending on the RLS implementation).

#### 4.1. Negotiation of Support for Resource Lists

This specification uses the SIP option tag mechanism for negotiating support for the extension defined herein. Refer to RFC 3261 [1] for the normative description of processing of the "Supported" and "Require" header fields and the 421 (Extension Required) response code.

A non-normative description of the implications of the use of option tags follows.

Any client that supports the event list extension will include an option tag of "eventlist" in a "Supported" header field of every SUBSCRIBE message for a subscription for which it is willing to process a list. If the subscription is made to a URI that represents a list, the RLS will include "eventlist" in a "Require" header field of the response to the SUBSCRIBE, and in all NOTIFY messages within that subscription.

Use of "Require: eventlist" in NOTIFY messages is applied by the notifier to satisfy the RFC 3261 requirement that a UAC MUST insert a Require header field into a request if the UAC wishes to insist that a UAS understand an extension in order to process the request. Because the NOTIFY would not be usable without applying the eventlist option, the notifier is obligated to include it.

Including "eventlist" in a "Require" header field in a SUBSCRIBE request serves no purpose except to break interoperability in certain cases, and is consequently NOT RECOMMENDED.

Sending of "Supported: eventlist" in a NOTIFY message is meaningless and silly. Implementations SHOULD NOT include "Supported: eventlist" in any requests except for SUBSCRIBE.

There is nothing in a SIP URI that indicates whether it represents a list of resources or a single resource. Therefore, if a subscriber sends a request to a URI that represents a list resource but does not include a Supported header field listing the "eventlist" token, the notifier will typically return a 421 (Extension Required) response code. RFC 3261 [1] advises that servers avoid returning a 421 and instead attempt to process the request without the extension. However, in this case, the URI fundamentally represents a list resource, and therefore the subscription cannot proceed without this extension.

#### 4.2. Subscription Duration

Since the primary benefit of the resource list server is to reduce the overall messaging volume to a subscriber, it is RECOMMENDED that the subscription duration to a list be reasonably long. The default, when no duration is specified, is taken from the underlying event package. Of course, the standard techniques [2] can be used to increase or reduce this amount.

#### 4.3. NOTIFY Bodies

An implementation compliant to this specification MUST support the multipart/related and application/rlmi+xml MIME types. These types MUST be included in an Accept header sent in a SUBSCRIBE message, in addition to any other types supported by the client (including any types required by the event package being used).

#### 4.4. RLS Processing of SUBSCRIBE Requests

Once the subscriber is authenticated, the RLS performs authorization per its local policy. In many cases, each resource list is associated with a particular user (the one who created it and manages the set of elements in it), and only that user will be allowed to subscribe. Of course, this mode of operation is not inherent in the use of resource lists, and an RLS can use any authorization policy it chooses.

#### 4.5. RLS Generation of NOTIFY Requests

This specification leaves the choice about how and when to generate NOTIFY requests at the discretion of the implementor. One of the differentiators between various RLS implementations is the means by which they aggregate, rate-limit, or optimize the way in which

notifications are generated. As a baseline behavior, the RLS MAY generate a NOTIFY to the RLS subscriber whenever the state of any resource on the list changes.

It is important to understand that any given subscription is a subscription either to a single resource or to a list of resources. This nature (single resource versus list of resources) cannot change during the duration of a single subscription. In particular, this means that RLSes MUST NOT send NOTIFY messages that do not contain RLMI for a subscription if they have previously sent NOTIFY messages in that subscription containing RLMI. Similarly, RLSes MUST NOT send NOTIFY messages that do contain RLMI for a subscription if they have previously sent NOTIFY messages in that subscription which do not.

List representations necessarily contain RLMI documents for two reasons. Importantly, they identify the resource to which the event state corresponds. Many state syntaxes do not fully identify the resource to which the state applies, or they may identify the resource in a different way than it is represented in the list; for example, PIDF documents may contain resource URIs that are not identical to the URI used to retrieve them. Further, RLMI documents serve to disambiguate multiple instances of a single resource.

See Section 5 for a detailed definition of the syntax used to convey the state of resource lists. For the purposes of the following discussion, it is important to know that the overall list contains zero or more resources, and that the resources contain zero or more instances. Each instance has a state associated with it (pending, active, or terminating) representing the state of the virtual subscription.

Notifications contain a multipart document, the first part of which always contains meta-information about the list (e.g., membership, state of the virtual subscription to the resource). Remaining parts are used to convey the actual state of the resources listed in the meta-information.

The "state" attribute of each instance of a resource in the meta-information is set according to the state of the virtual subscription. The meanings of the "state" attribute are described in RFC 3265 [2].

If an instance of a resource was previously reported to the subscriber but is no longer available (i.e., the virtual subscription to that instance has been terminated), the resource list server SHOULD include that resource instance in the meta-information in the first NOTIFY message sent to the subscriber following the instance's



unavailability. The RLS MAY continue to do so for future notifications.

When sending information for a terminated resource instance, the RLS indicates a state of "terminated" and an appropriate reason value. Valid reason values and their meanings are described in RFC 3265 [2]. If the RLS will attempt to recover the resource state again at some point in the future (e.g., when the reason in the meta-information is "probation"), then the instance of the resource SHOULD remain in the meta-information until the instance state is available, or until the RLS gives up on making such state available.

When the first SUBSCRIBE message for a particular subscription is received by an RLS, the RLS will often not know state information for all the resources specified by the resource list. For any resource for which state information is not known, the corresponding "uri" attribute will be set appropriately, and no <instance> elements will be present for the resource.

For an initial notification, sections corresponding to resources for which the RLS does have state will be populated with appropriate data (subject, of course, to local policy decisions). This will often occur if the resource list server is co-located with the server for one or more of the resources specified on the list.

Immediate notifications triggered as a result of subsequent SUBSCRIBE messages SHOULD include an RLMI document in which the full state is indicated. The RLS SHOULD also include state information for all resources in the list for which the RLS has state, subject to policy restrictions. This allows the subscriber to refresh their state, and to recover from lost notifications.

#### 4.6. Subscriber Processing of NOTIFY Requests

Notifications for a resource list can convey information about a subset of the list elements. This means that an explicit algorithm needs to be defined in order to construct coherent and consistent state.

The XML document present in the root of the multipart/related document contains a <resource> element for some or all of the resources in the list. Each <resource> element contains a URI that uniquely identifies the resource to which that section corresponds. When a NOTIFY arrives, it can contain full or partial state (as indicated by the "fullState" attribute of the top-level <list> element). If full state is indicated, then the recipient replaces all state associated with the list with the entities in the NOTIFY body. If full state is not indicated, the recipient of the NOTIFY

updates information for each identified resource. Information for any resources that are not identified in the NOTIFY is not changed, even if they were indicated in previous NOTIFY messages. See Section 5.6 for more information.

When full state is indicated, note that it applies only to the RLMI document in which it occurs. In particular, one of the <resource> elements in the document may in turn refer to another list of resources. Any such sub-lists will be detailed in their own RLMI documents, which may or may not have full state indicated.

Further note that the underlying event package may have its own rules for compositing partial state notification. When processing data related to those packages, their rules apply (i.e., the fact that they were reported as part of a list does not change their partial notification semantics).

Finally, note that as a consequence of the way in which resource list subscriptions work, polling of resource state may not be particularly useful. While such polls will retrieve the resource list, they will not necessarily contain state for some or all of the resources on the list.

#### 4.7. Handling of Forked Requests

Forking makes little sense with subscriptions to event lists, since the whole idea is a centralization of the source of notifications. Therefore, a subscriber to a list MUST NOT install multiple subscriptions when the initial request is forked. If multiple responses are received, they are handled using the techniques described in Section 4.4.9 of RFC 3265 [2].

#### 4.8. Rate of Notifications

One potential role of the RLS is to perform rate limitations on behalf of the subscriber. As such, this specification does not mandate any particular rate limitation, and rather leaves that to the discretion of the implementation.

#### 5. Using multipart/related to Convey Aggregate State

In order to convey the state of multiple resources, the list extension uses the "multipart/related" mime type. The syntax for multipart/related is defined in "The MIME Multipart/Related Content-type" [4].

### 5.1. XML Syntax

The root document of the multipart/related body MUST be a Resource List Meta-Information (RLMI) document. It is of the type "application/rlmi+xml". This document contains the meta-information for the resources contained in the notification. The schema for this XML document is given below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:rlmi"
  elementFormDefault="qualified"
  xmlns="urn:ietf:params:xml:ns:rlmi"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:element name="list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name" minOccurs="0"
          maxOccurs="unbounded" />
        <xs:element ref="resource" minOccurs="0"
          maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="uri" type="xs:anyURI" use="required" />
      <xs:attribute name="version" type="xs:unsignedInt"
        use="required" />
      <xs:attribute name="fullState" type="xs:boolean"
        use="required" />
      <xs:attribute name="cid" type="xs:string" use="optional" />
      <xs:anyAttribute processContents="lax" />
    </xs:complexType>
  </xs:element>
  <xs:element name="resource">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name" minOccurs="0"
          maxOccurs="unbounded" />
        <xs:element ref="instance" minOccurs="0"
          maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="uri" type="xs:anyURI" use="required" />
      <xs:anyAttribute processContents="lax" />
    </xs:complexType>
  </xs:element>
  <xs:element name="instance">
    <xs:complexType>
      <xs:sequence>
        <xs:any minOccurs="0" maxOccurs="unbounded"
```

```

        processContents="lax" />
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required" />
<xs:attribute name="state" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="active" />
      <xs:enumeration value="pending" />
      <xs:enumeration value="terminated" />
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="reason" type="xs:string"
  use="optional" />
<xs:attribute name="cid" type="xs:string" use="optional" />
<xs:anyAttribute processContents="lax" />
</xs:complexType>
</xs:element>
<xs:element name="name">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="xml:lang" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:schema>

```

An example of a document formatted using this schema follows.

```

<?xml version="1.0"?>
<list xmlns="urn:ietf:params:xml:ns:rlmi"
  uri="sip:adam-friends@lists.vancouver.example.com"
  version="7" fullState="true">
  <name xml:lang="en">Buddy List</name>
  <name xml:lang="fr">Liste d'amis</name>
  <resource uri="sip:bob@vancouver.example.com">
    <name>Bob Smith</name>
    <instance id="juwigmtboe" state="active"
      cid="12345.aaa@vancouver.example.com"/>
  </resource>
  <resource uri="sip:dave@vancouver.example.com">
    <name>Dave Jones</name>
    <instance id="hqzsuxtfyq" state="active"
      cid="12345.aab@vancouver.example.com"/>
  </resource>
  <resource uri="sip:jim@vancouver.example.com">

```

```
<name>Jim</name>
<instance id="oflzxqzuvvg" state="terminated"
          reason="rejected" />
</resource>
<resource uri="sip:ed@vancouver.example.com">
  <name>Ed</name>
  <instance id="grqhzsppxb" state="pending"/>
</resource>
</list>
```

## 5.2. List Attributes

The <list> element present in a list notification MUST contain three attributes.

The first mandatory <list> attribute is "uri", which contains the uri that corresponds to the list. Typically, this is the URI to which the SUBSCRIBE request was sent.

The second mandatory <list> attribute is "version", which contains a number from 0 to  $2^{32}-1$ . This version number MUST be 0 for the first NOTIFY message sent within a subscription, and MUST increase by exactly one for each subsequent NOTIFY sent within a subscription.

The third mandatory attribute is "fullState". The "fullState" attribute indicates whether the NOTIFY message contains information for every resource in the list. If it does, the value of the attribute is "true" (or "1"); otherwise, it is "false" (or "0"). The first NOTIFY sent in a subscription MUST contain full state, as must the first NOTIFY sent after receipt of a SUBSCRIBE request for the subscription.

Finally, <list> elements MAY contain a "cid" attribute. If present, the "cid" attribute identifies a section within the multipart/related body that contains aggregate state information for the resources contained in the list. The definition of such aggregate information is outside the scope of this document and will be defined on a per-package basis, as needed. The cid attribute is the Content-ID for the corresponding section in the multipart body.

The cid attribute MUST refer only to top-level parts of the multipart/related document for which the RLMI document in which it appears is the root. See Section 5.5 for an example.

### 5.3. Resource Attributes

The resource list contains one <resource> element for each resource being reported in the notification. These resource elements contain attributes that identify meta-data associated with each resource.

The "uri" attribute identifies the resource to which the <resource> element corresponds. Typically, this will be a SIP URI that, if subscribed to, would return the state of the resource. This attribute **MUST** be present.

### 5.4. Name Attributes

Each list and resource element contains zero or more name elements. These name elements contain human-readable descriptions or names for the resource list or resource. The contents of these elements are somewhat analogous to the "Display Name" present in the SIP name-addr element.

Name elements optionally contain the standard XML "xml:lang" attribute. The "xml:lang" attribute, if present, specifies the language of the human-readable name. If this attribute is present, it **MUST** contain a valid language tag. Language tags are defined in RFC 3066 [6]. The language tag assists applications in determining which of potentially several name elements should be rendered to the user.

### 5.5. Instance Attributes

Each resource element contains zero or more instance elements. These instance elements are used to represent a single notifier for the resource. For event packages that allow forking, multiple virtual subscriptions may exist for a given resource. Multiple virtual subscriptions are represented as multiple instance elements in the corresponding resource element. For subscriptions in which forking does not occur, at most one instance will be present for a given resource.

The "id" attribute contains an opaque string used to uniquely identify the instance of the resource. The "id" attribute is unique only within the context of a resource. Construction of this string is an implementation decision. Any mechanism for generating this string is valid, as long as uniqueness within the resource is assured.

The "state" attribute contains the subscription state for the identified instance of the resource. This attribute contains one of the values "active", "pending", or "terminated". The meanings for

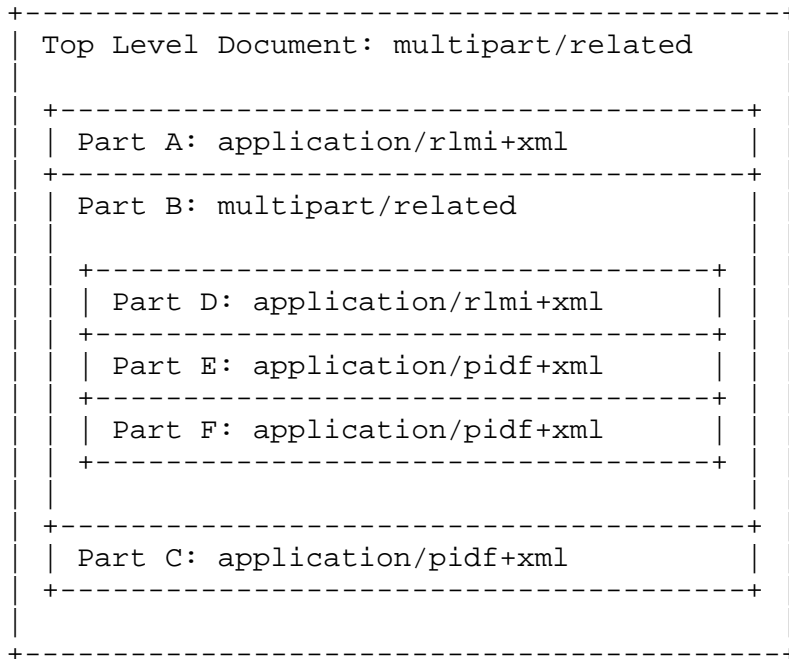
these values are as defined for the "Subscription-State" header field in RFC 3265 [2].

If the "state" attribute indicates "terminated", then a "reason" attribute MUST also be present. This "reason" attribute has the same values and meanings as those given for the "reason" parameter on the "Subscription-State" header field in RFC 3265 [2]. Note that the "reason" attribute is included for informational purposes; the list subscriber is not expected to take any automated actions based on the reason value.

Finally, the "cid" attribute, which MUST be present if the "state" attribute is "active", identifies the section within the multipart/related body that contains the actual resource state. This state is expressed in the content type defined by the event package for conveying state. The cid attribute is the Content-ID for the corresponding section in the multipart body.

The cid attribute MUST refer only to top-level parts of the multipart/related document for which the RLMI document in which it appears is the root.

For example, consider a multipart/related document containing three parts; we'll label these parts A, B, and C. Part A is type application/rlmi+xml, part B is type multipart/related, and part C is type application/pidf+xml. Part B is in turn a document containing three parts: D, E, and F. Part D is of type application/rlmi+xml, and parts E and F are of type application/pidf+xml.



Any "cid" attributes in document A must refer only to parts B or C. Referring to parts D, E, or F would be illegal. Similarly, any "cid" attributes in document D must refer only to parts E or F. Referring to any other parts would be illegal.

Also note that the subscription durations of any back-end subscriptions are not propagated into the meta-information state in any way.

## 5.6. Constructing Coherent Resource State

The resource list subscriber maintains a table for each resource list. The table contains a row for each resource in the resource list. Each row is indexed by the URI for that resource. That URI is obtained from the "uri" attribute on each <resource> element. The contents of each row contain the state of that resource as conveyed in the resource document.



For resources that provide versioning information (which is mandated by [2] for any formats that allow partial notification), each row also contains a resource state version number. The version number of the row is initialized with the version specified in the first document received, as defined by the corresponding event package. This value is used when comparing versions of partial notifications for a resource.

The processing of the resource list notification depends on whether it contains full or partial state.

#### 5.6.1. Processing Full State Notifications

If a notification contains full state, indicated by the <list> attribute "fullState" set to "true", the notification is used to update the table. A check is first made to ensure that the "version" attribute of the <list> attribute in the received message is greater than the local version number. If not, the received document is discarded without any further processing. Otherwise, the contents of the resource-list table are flushed and repopulated from the contents of the document. A new row in the table is created for each "resource" element.

#### 5.6.2. Processing Partial State Notifications

If a notification contains partial state, indicated by the <list> attribute "fullState" set to "false", a check is made to ensure that no list notifications have been lost. The value of the local version number (the "version" attribute of the <list> element) is compared to the version number of the new document.

- o If the value in the new document is exactly one higher than the local version number, the local version number is increased by one, and the document is processed as described below.
- o If the version in the document is more than one higher than the local version number, the local version number is set to the value in the new document, and the document is processed as described below. The list subscriber SHOULD also generate a refresh request to trigger a full state notification.
- o If the version in the document is less than or equal to the local version, the document is discarded without any further processing.

For each resource listed in the document, the subscriber checks to see whether a row exists for that resource. This check is done by comparing the Resource-URI value with the URI associated with the row. If the resource doesn't exist in the table, a row is added, and

its state is set to the information from that "resource" element. If the resource does exist, its state is updated to be the information from that "resource" element, as described in the definition of the event package. If a row is updated or created such that its state is now "terminated," that entry MAY be removed from the table at any time.

## 6. Example

This section gives an example call flow. It is not normative. If a conflict arises between this call flow and the normative behavior described in this or any other document, the normative descriptions are to be followed.

In this particular example, we request a subscription to a nested presence list. The subscriber's address-of-record is "sip:adam@vancouver.example.com", and the name of the nested list resource that we are subscribing to is called "sip:adam-buddies@pres.vancouver.example.com". The underlying event package is "presence", described by [8].

In this example, the RLS has information to service some of the resources on the list, but must consult other servers to retrieve information for others. The implementation of the RLS in this example uses the SIP SUBSCRIBE/NOTIFY mechanism to retrieve such information.

Terminal	pres.vancouver.example.com	pres.stockholm.example.org	pres.dallas.example.net
1	---SUBSCRIBE--->		
2	<-----200----->		
3	<-----NOTIFY----->		
4	-----200----->		
5		---SUBSCRIBE--->	
6		<-----200----->	
7		<-----NOTIFY----->	
8		-----200----->	
9		-----SUBSCRIBE----->	
10		<-----200----->	
11		<-----NOTIFY----->	
12		-----200----->	
13	<-----NOTIFY----->		
14	-----200----->		

1. We initiate the subscription by sending a SUBSCRIBE message to our local RLS. (There is no reason that the RLS we contact has to be in our domain, of course). Note that we must advertise support for application/rlmi+xml and multipart/related because we support the eventlist extension, and that we must advertise application/pidf+xml because we are requesting a subscription to presence.

Terminal -> Local RLS

```

SUBSCRIBE sip:adam-buddies@pres.vancouver.example.com SIP/2.0
Via: SIP/2.0/TCP terminal.vancouver.example.com;
    branch=z9hG4bKwYb6QREiCL
Max-Forwards: 70
To: <sip:adam-buddies@pres.vancouver.example.com>
From: <sip:adam@vancouver.example.com>;tag=ie4hbb8t
Call-ID: cdB34qLToC@terminal.vancouver.example.com
CSeq: 322723822 SUBSCRIBE
Contact: <sip:terminal.vancouver.example.com>
Event: presence
Expires: 7200
Supported: eventlist
Accept: application/pidf+xml
Accept: application/rlmi+xml
Accept: multipart/related
Accept: multipart/signed
Accept: application/pkcs7-mime
Content-Length: 0

```

2. The Local RLS completes the SUBSCRIBE transaction. Note that authentication and authorization would normally take place at this point in the call flow. Those steps are omitted for brevity.

Local RLS -> Terminal

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP terminal.vancouver.example.com;
    branch=z9hG4bKwYb6QREiCL
To: <sip:adam-buddies@pres.vancouver.example.com>;tag=zpNctbZq
From: <sip:adam@vancouver.example.com>;tag=ie4hbb8t
Call-ID: cdB34qLTtoC@terminal.vancouver.example.com
CSeq: 322723822 SUBSCRIBE
Contact: <sip:pres.vancouver.example.com>
Expires: 7200
Require: eventlist
Content-Length: 0
```

3. As is required by RFC 3265 [2], the RLS sends a NOTIFY immediately upon accepting the subscription. In this example, we are assuming that the local RLS is also an authority for presence information for the users in the "vancouver.example.com" domain. The NOTIFY contains an RLMI document describing the entire buddy list (initial notifies require full state), as well as presence information for the users about which it already knows. Note that, since the RLS has not yet retrieved information for some of the entries on the list, those <resource> elements contain no <instance> elements.

Local RLS -> Terminal

```
NOTIFY sip:terminal.vancouver.example.com SIP/2.0
Via: SIP/2.0/TCP pres.vancouver.example.com;
    branch=z9hG4bKMgRentTETmm
Max-Forwards: 70
From: <sip:adam-buddies@pres.vancouver.example.com>;tag=zpNctbZq
To: <sip:adam@vancouver.example.com>;tag=ie4hbb8t
Call-ID: cdB34qLTtoC@terminal.vancouver.example.com
CSeq: 997935768 NOTIFY
Contact: <sip:pres.vancouver.example.com>
Event: presence
Subscription-State: active;expires=7200
Require: eventlist
Content-Type: multipart/related;type="application/rlmi+xml";
    start="<nXYxAE@pres.vancouver.example.com>";
    boundary="50UBfW7LSCVltggUPe5z"
Content-Length: 1560
```

```
--50UBfW7LSCVltggUPe5z
Content-Transfer-Encoding: binary
Content-ID: <nXYxAE@pres.vancouver.example.com>
Content-Type: application/rlmi+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<list xmlns="urn:ietf:params:xml:ns:rlmi"
      uri="sip:adam-friends@pres.vancouver.example.com"
      version="1" fullState="true">
  <name xml:lang="en">Buddy List at COM</name>
  <name xml:lang="de">Liste der Freunde an COM</name>
  <resource uri="sip:bob@vancouver.example.com">
    <name>Bob Smith</name>
    <instance id="juwigmtboe" state="active"
              cid="bUZBsM@pres.vancouver.example.com"/>
  </resource>
  <resource uri="sip:dave@vancouver.example.com">
    <name>Dave Jones</name>
    <instance id="hqzsuxtfyq" state="active"
              cid="ZvSvkz@pres.vancouver.example.com"/>
  </resource>
  <resource uri="sip:ed@dallas.example.net">
    <name>Ed at NET</name>
  </resource>
  <resource uri="sip:adam-friends@stockholm.example.org">
    <name xml:lang="en">My Friends at ORG</name>
    <name xml:lang="de">Meine Freunde an ORG</name>
  </resource>
</list>
```

```
--50UBfW7LSCVltggUPe5z
Content-Transfer-Encoding: binary
Content-ID: <bUZBsM@pres.vancouver.example.com>
Content-Type: application/pidf+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          entity="sip:bob@vancouver.example.com">
  <tuple id="sg89ae">
    <status>
      <basic>open</basic>
    </status>
    <contact priority="1.0">sip:bob@vancouver.example.com</contact>
  </tuple>
</presence>
```

```
--50UBfW7LSCVltggUPe5z
Content-Transfer-Encoding: binary
```

Content-ID: <ZvSvkz@pres.vancouver.example.com>  
 Content-Type: application/pidf+xml; charset="UTF-8"

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
  entity="sip:dave@vancouver.example.com">
  <tuple id="slie74">
    <status>
      <basic>closed</basic>
    </status>
  </tuple>
</presence>
```

--50UBfW7LSCVltggUPe5z--

4. The terminal completes the transaction.

Terminal -> Local RLS

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP pres.vancouver.example.com;
  branch=z9hG4bKMgRentETmm
From: <sip:adam-buddies@pres.vancouver.example.com>;tag=zpNctbZq
To: <sip:adam@vancouver.example.com>;tag=ie4hbb8t
Call-ID: cdB34qLTtoC@terminal.vancouver.example.com
CSeq: 997935768 NOTIFY
Contact: <sip:terminal.vancouver.example.com>
Content-Length: 0
```

5. In order to service the subscription, the local RLS subscribes to the state of the resources. In this step, the RLS attempts to subscribe to the presence state of the resource "sip:ed@dallas.example.net". Since the local RLS knows how to receive notifications for list subscriptions, it includes the "Supported: eventlist" header field in its request. Although the linkage between this subscription and the one sent by the terminal is left up to the application, this message demonstrates some reasonable behavior by including "Accept" header fields for all the body types it knows the subscriber (Terminal) supports. This is safe to do, since the local RLS will only pass these formats through to the subscriber and does not need to actually understand them.

Local RLS -> Presence Server in dallas.example.net

```
SUBSCRIBE sip:ed@dallas.example.net SIP/2.0
Via: SIP/2.0/TCP pres.vancouver.example.com;
  branch=z9hG4bKMEyGjdG1LH
```

Max-Forwards: 70  
To: <sip:ed@dallas.example.net>  
From: <sip:adam@vancouver.example.com>;tag=aM5icQu9  
Call-ID: Ugwz5ARxNw@pres.vancouver.example.com  
CSeq: 870936068 SUBSCRIBE  
Contact: <sip:pres.vancouver.example.com>  
Identity: Tm8sIHRoaXMgaXNuJ3QgYSByZWFrS1GNlcnQuIFlvdSBvn  
Zpb3VzbHkgaGF2ZSB0aW1lIHRvIGtpbGwuIEkKc3VnZ2V  
zdCBodHRwOi8vd3d3LmhvbWVzdGFycnVubmVyLmNvbS8K  
Identity-Info: https://vancouver.example.com/cert  
Event: presence  
Expires: 3600  
Supported: eventlist  
Accept: application/pidf+xml  
Accept: application/rlmi+xml  
Accept: multipart/related  
Accept: multipart/signed  
Accept: application/pkcs7-mime  
Content-Length: 0

6. The Presence Server in dallas.example.net completes the SUBSCRIBE transaction. Note that authentication would normally take place at this point in the call flow. This step is omitted for brevity.

Presence Server in dallas.example.net -> Local RLS

SIP/2.0 200 OK  
Via: SIP/2.0/TCP pres.vancouver.example.com;  
branch=z9hG4bKMEyGjdG1LH  
To: <sip:ed@dallas.example.net>;tag=e45TmHTh  
From: <sip:adam@vancouver.example.com>;tag=aM5icQu9  
Call-ID: Ugwz5ARxNw@pres.vancouver.example.com  
CSeq: 870936068 SUBSCRIBE  
Contact: <sip:dallas.example.net>  
Expires: 3600  
Content-Length: 0

7. In this example, we assume that the server at dallas.example.net doesn't have enough authorization information to reject or accept our subscription. The initial notify, therefore, contains a "Subscription-State" of "pending". Presumably, the party responsible for accepting or denying authorization for the resource is notified of this change; however, those steps are not included in this call flow for brevity.

Presence Server in dallas.example.net -> Local RLS

```
NOTIFY sip:pres.vancouver.example.com SIP/2.0
Via: SIP/2.0/TCP pres.dallas.example.net;
    branch=z9hG4bKfwpklPxmrW
Max-Forwards: 70
From: <sip:ed@dallas.example.net>;tag=e45TmHTh
To: <sip:adam@vancouver.example.com>;tag=aM5icQu9
Call-ID: Ugwz5ARxNw@pres.vancouver.example.com
CSeq: 1002640632 NOTIFY
Contact: <sip:dallas.example.net>
Subscription-State: pending;expires=3600
Event: presence
Require: eventlist
Content-Length: 0
```

8. The local RLS completes the NOTIFY transaction. Note that, at this point, the Local RLS has new information to report to the subscriber. Whether it chooses to report the information immediately or spool it up for later delivery is completely up to the application. For this example, we assume that the RLS will wait for a short period of time before doing so, in order to allow the subscriptions it sent out sufficient time to provide useful data.

Local RLS -> Presence Server in dallas.example.net

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP pres.dallas.example.net;
    branch=z9hG4bKfwpklPxmrW
From: <sip:ed@dallas.example.net>;tag=e45TmHTh
To: <sip:adam@vancouver.example.com>;tag=aM5icQu9
Call-ID: Ugwz5ARxNw@pres.vancouver.example.com
CSeq: 1002640632 NOTIFY
Contact: <sip:pres.vancouver.example.com>
Content-Length: 0
```

9. The Local RLS subscribes to the state of the other non-local resource.

Local RLS -> RLS in stockholm.example.org

```
SUBSCRIBE sip:adam-friends@stockholm.example.org SIP/2.0
Via: SIP/2.0/TCP pres.vancouver.example.com;
    branch=z9hG4bKFSrAF8CZFL
Max-Forwards: 70
To: <sip:adam-friends@stockholm.example.org>
From: <sip:adam@vancouver.example.com>;tag=a12eztNf
```



Call-ID: kBq5XhtZLN@pres.vancouver.example.com  
CSeq: 980774491 SUBSCRIBE  
Contact: <sip:pres.vancouver.example.com>  
Identity: Tm90IGEgcmVhbCBzaWduYXR1cmUsIGVpdGhlci4gQ2VydGFp  
bmX5IHlvdSBoYXZlIGJldHRlcgp0aGluZ3MgdG8gYmUgZG9p  
bmcuIEhhdmUgeW91IGZpbmlzaGVkIHlvdXlUkxTIHl1dD8K  
Identity-Info: https://vancouver.example.com/cert  
Event: presence  
Expires: 3600  
Supported: eventlist  
Accept: application/pidf+xml  
Accept: application/rlmi+xml  
Accept: multipart/related  
Accept: multipart/signed  
Accept: application/pkcs7-mime  
Content-Length: 0

10. The RLS in stockholm.example.org completes the SUBSCRIBE transaction. Note that authentication would normally take place at this point in the call flow. This step is omitted for brevity.

RLS in stockholm.example.org -> Local RLS

SIP/2.0 200 OK  
Via: SIP/2.0/TCP pres.vancouver.example.com;  
branch=z9hG4bKFSrAF8CZFL  
To: <sip:adam-friends@stockholm.example.org>;tag=JenZ40P3  
From: <sip:adam@vancouver.example.com>;tag=a12eztNf  
Call-ID: kBq5XhtZLN@pres.vancouver.example.com  
CSeq: 980774491 SUBSCRIBE  
Contact: <sip:stockholm.example.org>  
Expires: 3600  
Content-Length: 0

11. In this example, we assume that the RLS in stockholm.example.org is also an authority for presence information for the users in the "stockholm.example.org" domain. The NOTIFY contains an RLMI document describing the contained buddy list, as well as presence information for those users. In this particular case, the RLS in stockholm.example.org has chosen to sign [14] the body of the NOTIFY message. As described in RFC 3851, signing is performed by creating a multipart/signed document that has two parts. The first part is the document to be signed (in this example, the multipart/related document that describes the list resource states), while the second part is the actual signature.

RLS in stockholm.example.org -> Local RLS

NOTIFY sip:pres.vancouver.example.com SIP/2.0  
Via: SIP/2.0/TCP pres.stockholm.example.org;  
branch=z9hG4bKmGLlNyZfQI  
Max-Forwards: 70  
From: <sip:adam-friends@stockholm.example.org>;tag=JenZ40P3  
To: <sip:adam@vancouver.example.com>;tag=a12eztNf  
Call-ID: kBq5XhtZLN@pres.vancouver.example.com  
CSeq: 294444656 NOTIFY  
Contact: <sip:stockholm.example.org>  
Event: presence  
Subscription-State: active;expires=3600  
Require: eventlist  
Content-Type: multipart/signed;  
protocol="application/pkcs7-signature";  
micalg=sha1;boundary="l3WMZaaL8NpQWGnQ4mlU"  
Content-Length: 2038

--l3WMZaaL8NpQWGnQ4mlU  
Content-Transfer-Encoding: binary  
Content-ID: <ZPvJHL@stockholm.example.org>  
Content-Type: multipart/related;type="application/rlmi+xml";  
start="<Cvjpeo@stockholm.example.org>";  
boundary="tuLLl3lDyPZX0GMr2YOo"

--tuLLl3lDyPZX0GMr2YOo  
Content-Transfer-Encoding: binary  
Content-ID: <Cvjpeo@stockholm.example.org>  
Content-Type: application/rlmi+xml;charset="UTF-8"

```
<?xml version="1.0" encoding="UTF-8"?>
<list xmlns="urn:ietf:params:xml:ns:rlmi"
      uri="sip:adam-friends@stockholm.example.org" version="1"
      fullState="true">
  <name xml:lang="en">Buddy List at COM</name>
  <name xml:lang="de">Liste der Freunde an COM</name>
  <resource uri="sip:joe@stockholm.example.org">
    <name>Joe Thomas</name>
    <instance id="1" state="active"
              cid="mrEakg@stockholm.example.org"/>
  </resource>
  <resource uri="sip:mark@stockholm.example.org">
    <name>Mark Edwards</name>
    <instance id="1" state="active"
              cid="KKMDmv@stockholm.example.org"/>
  </resource>
</list>
```

```
--tuLLl3lDyPZX0GMr2Y0o
Content-Transfer-Encoding: binary
Content-ID: <mrEakg@stockholm.example.org>
Content-Type: application/pidf+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
  entity="sip:joe@stockholm.example.org">
  <tuple id="x823a4">
    <status>
      <basic>open</basic>
    </status>
    <contact priority="1.0">sip:joe@stockholm.example.org</contact>
  </tuple>
</presence>

--tuLLl3lDyPZX0GMr2Y0o
Content-Transfer-Encoding: binary
Content-ID: <KKMDmv@stockholm.example.org>
Content-Type: application/pidf+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
  entity="sip:mark@stockholm.example.org">
  <tuple id="z98075">
    <status>
      <basic>closed</basic>
    </status>
  </tuple>
</presence>

--tuLLl3lDyPZX0GMr2Y0o--

--l3WMZaaL8NpQWGnQ4mlU
Content-Transfer-Encoding: binary
Content-ID: <K9LB7k@stockholm.example.org>
Content-Type: application/pkcs7-signature

[PKCS #7 signature here]

--l3WMZaaL8NpQWGnQ4mlU--
```

12. The Local RLS completes the NOTIFY transaction.

Local RLS -> RLS in stockholm.example.org

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP pres.stockholm.example.org;
    branch=z9hG4bKmGLlNyZfQI
From: <sip:adam-friends@stockholm.example.org>;tag=JenZ40P3
To: <sip:adam@vancouver.example.com>;tag=a12eztNf
Call-ID: kBq5XhtZLN@pres.vancouver.example.com
CSeq: 294444656 NOTIFY
Contact: <sip:pres.vancouver.example.com>
Content-Length: 0
```

13. At this point, the Local RLS decides it has collected enough additional information to warrant sending a new notification to the user. Although sending a full notification would be perfectly acceptable, the RLS decides to send a partial notification instead. The RLMI document contains only information for the updated resources, as indicated by setting the "fullState" parameter to "false". To avoid corrupting the S/MIME signature on the data received from the RLS in stockholm.example.org, the local RLS copies the entire multipart/signed body as-is into the notification that it sends.

Local RLS -> Terminal

```
NOTIFY sip:terminal.vancouver.example.com SIP/2.0
Via: SIP/2.0/TCP pres.vancouver.example.com;
    branch=z9hG4bK4EPlfSFQK1
Max-Forwards: 70
From: <sip:adam-buddies@pres.vancouver.example.com>;tag=zpNctbZq
To: <sip:adam@vancouver.example.com>;tag=ie4hbb8t
Call-ID: cdB34qLToC@terminal.vancouver.example.com
CSeq: 997935769 NOTIFY
Contact: <sip:pres.vancouver.example.com>
Event: presence
Subscription-State: active;expires=7200
Require: eventlist
Content-Type: multipart/related;type="application/rlmi+xml";
    start="<2BEI83@pres.vancouver.example.com>";
    boundary="TfZxoxgAvLqgj4wRWPDL"
Content-Length: 2862

--TfZxoxgAvLqgj4wRWPDL
Content-Transfer-Encoding: binary
Content-ID: <2BEI83@pres.vancouver.example.com>
Content-Type: application/rlmi+xml;charset="UTF-8"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<list xmlns="urn:ietf:params:xml:ns:rlmi"
      uri="sip:adam-friends@pres.vancouver.example.com" version="2"
      fullState="false">
  <name xml:lang="en">Buddy List at COM</name>
  <name xml:lang="de">Liste der Freunde an COM</name>
  <resource uri="sip:ed@dallas.example.net">
    <name>Ed at NET</name>
    <instance id="sdlkmeopdf" state="pending"/>
  </resource>
  <resource uri="sip:adam-friends@stockholm.example.org">
    <name xml:lang="en">My Friends at ORG</name>
    <name xml:lang="de">Meine Freunde an ORG</name>
    <instance id="cmpqweitlp" state="active"
      cid="1KQhyE@pres.vancouver.example.com"/>
  </resource>
</list>
```

```
--TfZxoxgAvLqgj4wRWPDL
Content-Transfer-Encoding: binary
Content-ID: <1KQhyE@pres.vancouver.example.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1;boundary="l3WMZaaL8NpQWGnQ4mlU"
```

```
--l3WMZaaL8NpQWGnQ4mlU
Content-Transfer-Encoding: binary
Content-ID: <ZPvJHL@stockholm.example.org>
Content-Type: multipart/related;type="application/rlmi+xml";
  start="<Cvjpeo@stockholm.example.org>";
  boundary="tuLLl3lDyPZX0GMr2YOo"
```

```
--tuLLl3lDyPZX0GMr2YOo
Content-Transfer-Encoding: binary
Content-ID: <Cvjpeo@stockholm.example.org>
Content-Type: application/rlmi+xml;charset="UTF-8"
<?xml version="1.0" encoding="UTF-8"?>
<list xmlns="urn:ietf:params:xml:ns:rlmi"
      uri="sip:adam-friends@stockholm.example.org" version="1"
      fullState="true">
  <name xml:lang="en">Buddy List at ORG</name>
  <name xml:lang="de">Liste der Freunde an ORG</name>
  <resource uri="sip:joe@stockholm.example.org">
    <name>Joe Thomas</name>
    <instance id="1" state="active"
      cid="mrEakg@stockholm.example.org"/>
  </resource>
  <resource uri="sip:mark@stockholm.example.org">
```

```

        <name>Mark Edwards</name>
        <instance id="1" state="active"
            cid="KKMDmv@stockholm.example.org"/>
    </resource>
</list>

--tuLLl3lDyPZX0GMr2Y0o
Content-Transfer-Encoding: binary
Content-ID: <mrEakg@stockholm.example.org>
Content-Type: application/pidf+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
    entity="sip:joe@stockholm.example.org">
    <tuple id="x823a4">
        <status>
            <basic>open</basic>
        </status>
        <contact priority="1.0">sip:joe@stockholm.example.org</contact>
    </tuple>
</presence>

--tuLLl3lDyPZX0GMr2Y0o
Content-Transfer-Encoding: binary
Content-ID: <KKMDmv@stockholm.example.org>
Content-Type: application/pidf+xml;charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
    entity="sip:mark@stockholm.example.org">
    <tuple id="z98075">
        <status>
            <basic>closed</basic>
        </status>
    </tuple>
</presence>
--tuLLl3lDyPZX0GMr2Y0o--

--l3WMZaaL8NpQWGnQ4mlU
Content-Transfer-Encoding: binary
Content-ID: <K9LB7k@stockholm.example.org>
Content-Type: application/pkcs7-signature

[PKCS #7 signature here]

--l3WMZaaL8NpQWGnQ4mlU--
--TfZxoxgAvLqgj4wRWPDL--

```

14. The terminal completes the NOTIFY transaction.

Terminal -> Local RLS

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP pres.vancouver.example.com;
    branch=z9hG4bK4EPlfSFQK1
From: <sip:adam-buddies@pres.vancouver.example.com>;tag=zpNctbZq
To: <sip:adam@vancouver.example.com>;tag=ie4hbb8t
Call-ID: cdB34qLTtoC@terminal.vancouver.example.com
CSeq: 997935769 NOTIFY
Contact: <sip:terminal.vancouver.example.com>
Content-Length: 0
```

## 7. Security Considerations

Note that the mechanisms for obtaining state information for resources in a list are generally left to the RLS implementor. Some of the security issues below are specific to the circumstance in which a SIP back-end subscription is used for such a purpose. Non-SIP mechanisms for obtaining state information of resources in a list will typically have their own security issues associated with doing so; however, exhaustively enumerating such access methods is not possible in this document. Implementors using such mechanisms must analyze their chosen access methods for relevant security issues.

### 7.1. Authentication

If back-end subscriptions are required to retrieve resource state information, the end user is no longer the direct subscriber to the state of the resource. This means that direct authentication of the user is no longer possible.

#### 7.1.1. RLS and Subscriber in the Same Domain

It is expected that the most common deployment of RLSes entails that the subscribers to the RLS will be in the same domain as the RLS. When this is the case, the RLS then has the ability to act as an authentication service. The role of authentication service is defined in "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)" [7].

At a high level, under this system, the RLS authenticates the subscriber and then includes an "Identity" header field in all of the back-end subscriptions performed on behalf of that authenticated user. This "Identity" header field cryptographically asserts that the request has been authorized to be made on behalf of the user indicated in the "From" header field.

Because the ability to authenticate requests is central to the proper functioning of the network, any RLS that uses SIP back-end subscriptions to acquire information about the resources in a resource list MUST be able to act as an authentication service as defined in [7], provided that local administrative policy allows it to do so.

In other words, all RLS implementations that support back-end SIP subscriptions also must include the ability to be configured to act as an authentication service. Whether any given administrator chooses to activate such a feature is completely up to them. Of course, lacking the ability to act as an identity server, any RLS so configured will behave as described in the following section, since it is effectively acting as if it were in a different domain than the user.

#### 7.1.2. RLS and Subscriber in Different Domains

In the general case, the SIP Authenticated Identity extensions do not provide a means for the RLS to securely assert that subscriptions are being performed on the end user's behalf. Specifically, when the subscriber and the RLS are in different domains, the RLS will have no means by which it can vouch for the user's identity. Mechanisms by which back-end subscriptions in such circumstances can be authenticated are left for future study.

Until such general solutions are developed, RLSes that are in a different domain than the subscriber on whose behalf they are creating back-end subscriptions SHOULD subscribe to the resources using their own identity. By doing so, the RLS will generally obtain only the resource information that is made publicly available.

Absent such general solutions, implementations of subscriber user agents MAY attempt direct subscriptions to resources in the resource list when subscribing to an RLS outside of their domain (either directly or by way of another resource list subscription). The resources to be subscribed to will be those indicated in the "uri" attribute of the <resource> elements present in the RLMI document returned by the RLS. Directly subscribing to the resources allows proper authentication of the user to take place, which will generally authorize them to receive more complete state information. Implementations that choose to perform such direct subscriptions SHOULD use the data retrieved instead of any information about the resource obtained via the list subscription.



## 7.2. Risks of Improper Aggregation

A resource list server typically serves information to multiple subscribers at once. In many cases, resources may be present in several lists; additionally, it is quite possible that resource list servers will have two users subscribe to the same list.

In these cases, misguided RLS implementations may attempt to minimize network load by maintaining only one back-end subscription to a resource in a list and presenting the result of such a subscription to more than one user. Of course, doing so circumvents any authorization policy that the notifier for the resource maintains. Keep in mind that authorization is often much more than a simple binary "allowed/not allowed" decision; resources may render very different -- and even conflicting -- resource states, depending on the identity of the subscribing user.

To prevent the transmission of event information to anyone other than the intended recipient, implementations **MUST NOT** present the result of one back-end subscription to more than one user, unless:

- a. The RLS has adequate access to the complete authorization policy associated with the resource to which the back-end subscription has been made, **AND**
- b. The RLS can and has determined that presenting the information to more than one user does not violate such policy.

Note that this is a very difficult problem to solve correctly. Even in the cases where such access is believed possible, this mode of operation is **NOT RECOMMENDED**.

## 7.3. Signing and Sealing

Implementors should keep in mind that any section of the MIME body may be signed and/or encrypted as necessary. Resource List Servers should take care not to modify any MIME bodies they receive from any back-end subscriptions, and should not generally rely on being able to read them.

In order to facilitate security, resource list servers **SHOULD** pass along indication for support of "multipart/signed" and "application/pkcs7-mime" content types to any SIP back-end subscriptions, if the subscriber includes them in the initial SUBSCRIBE message. Not doing so may actually result in resources refusing to divulge state (if notifier policy requires encryption, but the RLS fails to convey support), or subscribers discarding valid state (if subscriber policy requires a signature, but the RLS fails to convey support).

Note that actual implementation of encryption and signing by the RLS is not necessary to be able to pass through signed and/or encrypted bodies.

#### 7.4. Infinite Loops

One risk introduced by the ability to nest resource lists is the possibility of creating lists that ultimately contain themselves as a sub-list. Detection and handling of such a case is trivial when the RLS services all the virtual subscriptions internally. When back-end subscriptions are created to service virtual subscriptions, however, detection of such situations becomes a more difficult problem.

Implementors of RLSes that create back-end subscriptions MUST implement safeguards to prevent such nestings from creating an infinite loop of subscriptions. Typically, such mechanisms will require support in the back-end subscription protocol. In particular, applying filters to the back-end subscriptions can be an effective way to preclude such problems.

### 8. IANA Considerations

#### 8.1. New SIP Option Tag: eventlist

This section defines a new option tag for the registry established by Section 27.1 of RFC 3261[1].

Option Tag Name: eventlist

Description: Extension to allow subscriptions to lists of resources.

Published specification: RFC 4662

#### 8.2. New MIME type for Resource List Meta-Information

MIME Media Type Name: application

MIME subtype name: rlmi+xml

Required parameters: None

Optional parameters: charset

See RFC 3023 [12] for a discussion of the charset parameter on XML-derived MIME types. Since this MIME type is used exclusively in SIP, the use of UTF-8 encoding is strongly encouraged.

Encoding considerations: 8-bit text

Security considerations: Security considerations specific to uses of this MIME type are discussed in RFC 4662. RFC 1874 [11] and RFC 3023 [12] discuss security issues common to all uses of XML.

Interoperability considerations: The use of this MIME body is intended to be generally interoperable. No unique considerations have been identified.

Published specification: RFC 4662

Applications that use this media type: This media type is used to convey meta-information for the state of lists of resources within a Session Initiation Protocol (SIP) subscription.

Additional information:

Magic Number(s): None.

File Extension(s): None.

Macintosh File Type Code(s): None.

Object Identifier(s) or OID(s): None.

Intended usage: Limited Use

Other Information/General Comment: None.

Person to contact for further information:

Name: Adam Roach

E-Mail: adam@estacado.net

Author/Change Controller: The specification of this MIME type is a work product of the SIMPLE working group and was authored by Adam Roach, Jonathan Rosenberg, and Ben Campbell. The IETF has change control over its specification.

### 8.3. URN Sub-Namespace

URI: urn:ietf:params:xml:ns:rlmi

Description: This is the XML namespace URI for XML elements defined by RFC 4662 to describe information about subscriptions when such subscriptions are aggregated within a single SIP subscription. It is used in the application/rlmi+xml body type.

Registrant Contact:

Name: Adam Roach

E-Mail: adam@estacado.net

Author/Change Controller: The specification of this MIME type is a work product of the SIMPLE working group and was authored by Adam Roach, Jonathan Rosenberg, and Ben Campbell. The IETF has change control over its specification.

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=utf-8"/>
  <title>Namespace for SIP Event Resource List
    Meta-Information</title>
</head>
<body>
  <h1>Namespace for SIP Event Resource List
    Meta-Information</h1>
  <h2>application/rlmi+xml</h2>
  <p>See <a href="[http://www.rfc-editor.org/rfc/rfc4662.txt]">
    RFC4662</a>.</p>
</body>
</html>
END
```

## 9. Acknowledgements

Thanks to Sean Olson for a review of and corrections to the usage of XML in this protocol.

Thanks also to Hisham Khartabil, Paul Kyzivat, Keith Drage, and Robert Sparks for their careful reviews of and comments on this document.

## 10. References

### 10.1. Normative References

- [1] 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.
- [2] Roach, A. B., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.
- [3] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

- [4] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, August 1998.
- [5] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [6] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.
- [7] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.

#### 10.2. Informative References

- [8] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.
- [9] Burger, E., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", RFC 4483, May 2006.
- [10] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.
- [11] Levinson, E., "SGML Media Types", RFC 1874, December 1995.
- [12] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [13] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.
- [14] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.
- [15] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

## Authors' Addresses

Adam Roach  
Estacado Systems  
US

EMail: adam@estacado.net

Ben Campbell  
Estacado Systems  
US

EMail: ben@estacado.net

Jonathan Rosenberg  
Cisco Systems  
600 Lanidex Plaza  
Parsippany, NJ 07054-2711  
US

EMail: jdrosen@cisco.com

## Full Copyright Statement

Copyright (C) The Internet Society (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).

## Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

