Network Working Group
Request for Comments: 3680
Category: Standards Track

J. Rosenberg
dynamicsoft

March 2004

A Session Initiation Protocol (SIP) Event Package for Registrations

Status of this Memo

Copyright Notice

Abstract

This document defines a Session Initiation Protocol (SIP) event package for registrations. Through its REGISTER method, SIP allows a user agent to create, modify, and delete registrations. Registrations can also be altered by administrators in order to enforce policy. As a result, these registrations represent a piece of state in the network that can change dynamically. There are many cases where a user agent would like to be notified of changes in this state. This event package defines a mechanism by which those user agents can request and obtain such notifications.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] provides all of the functions needed for the establishment and maintenance of communications sessions between users. One of the functions it provides is a registration operation. A registration is a binding between a SIP URI, called an address-of-record, and one or more contact URIs. These contact URIs represent additional resources that can be contacted in order to reach the user identified by the address-of-record. When a proxy receives a request within its domain of administration, it uses the Request-URI as an address-of-record, and uses the contacts bound to the address-of-record to forward (or redirect) the request.

The SIP REGISTER method provides a way for a user agent to manipulate registrations. Contacts can be added or removed, and the current set of contacts can be queried. Registrations can also change as a result of administrator policy. For example, if a user is suspected of fraud, their registration can be deleted so that they cannot receive any requests. Registrations also expire after some time if not refreshed.

Registrations represent a dynamic piece of state maintained by the network. There are many cases in which user agents would like to know about changes to the state of registrations. The SIP Events Framework [2] defines a generic framework for subscription to, and notification of, events related to SIP systems. The framework defines the methods SUBSCRIBE and NOTIFY, and introduces the notion of a package. A package is a concrete application of the event framework to a particular class of events. Packages have been defined for user presence [9], for example. This specification defines a package for registration state.

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 BCP 14, RFC 2119 [3] and indicate requirement levels for compliant implementations.

3. Usage Scenarios

There are many applications of this event package. A few are documented here for illustrative purposes.

3.1. Forcing Re-Authentication

It is anticipated that many SIP devices will be wireless devices that will be always-on, and therefore, continually registered to the network. Unfortunately, history has shown that these devices can be compromised. To deal with this, an administrator will want to terminate or shorten a registration, and ask the device to re-register so it can be re-authenticated. To do this, the device subscribes to the registration event package for the address-of-record that it is registering contacts against. When the administrator shortens registration (for example, when fraud is suspected) the registration server sends a notification to the device. It can then re-register and re-authenticate itself. If it cannot re-authenticate, the expiration will terminate shortly thereafter.

3.2. Composing Presence

An important concept to understand is the relationship between this event package and the event package for user presence [9]. User presence represents the willingness and ability of a user to communicate with other users on the network. It is composed of a set of contact addresses that represent the various means for contacting the user. Those contact addresses might represent the contact address for voice, for example. Typically, the contact address

listed for voice will be an address-of-record. The status of that contact (whether its open or closed) may depend on any number of factors, including the state of any registrations against that address-of-record. As a result, registration state can be viewed as an input to the process which determines the presence state of a user. Effectively, registration state is "raw" data, which is combined with other information about a user to generate a document that describes the user's presence.

In fact, this event package allows for a presence server to be separated from a SIP registration server, yet still use registration information to construct a presence document. When a presence server receives a presence subscription for some user, the presence server itself would generate a subscription to the registration server for the registration event package. As a result, the presence server would learn about the registration state for that user, and it could use that information to generate presence documents.

3.3. Welcome Notices

A common service in current mobile networks are "welcome notices". When the user turns on their phone in a foreign country, they receive a message that welcomes them to the country, and provides information on transportation services, for example.

In order to implement this service in a SIP system, an application server can subscribe to the registration state of the user. When the user turns on their phone, the phone will generate a registration. This will result in a notification being sent to the application that the user has registered. The application can then send a SIP MESSAGE request [10] to the device, welcoming the user and providing any necessary information.

4. Package Definition

This section fills in the details needed to specify an event package as defined in Section 4.4 of [2].

4.1. Event Package Name

The SIP Events specification requires package definitions to specify the name of their package or template-package.

The name of this package is "reg". As specified in [2], this value appears in the Event header present in SUBSCRIBE and NOTIFY requests.

Example:

Event: reg

4.2. Event Package Parameters

The SIP Events specification requires package and template-package definitions to specify any package specific parameters of the Event header that are used by it.

No package specific Event header parameters are defined for this event package.

4.3. SUBSCRIBE Bodies

The SIP Events specification requires package or template-package definitions to define the usage, if any, of bodies in SUBSCRIBE requests.

A SUBSCRIBE for registration events MAY contain a body. This body would serve the purpose of filtering the subscription. The definition of such a body is outside the scope of this specification.

A SUBSCRIBE for the registration package MAY be sent without a body. This implies that the default registration filtering policy has been requested. The default policy is:

- Notifications are generated every time there is any change in the state of any of the registered contacts for the resource being subscribed to. Those notifications only contain information on the contacts whose state has changed.

- Notifications triggered from a SUBSCRIBE contain full state (the list of all contacts bound to the address-of-record).

Of course, the server can apply any policy it likes to the subscription.

4.4. Subscription Duration

The SIP Events specification requires package definitions to define a default value for subscription durations, and to discuss reasonable choices for durations when they are explicitly specified.

Registration state changes as contacts are created through REGISTER requests, and then time out due to lack of refresh. Their rate of change is therefore related to the typical registration expiration. Since the default expiration for registrations is 3600 seconds, the

default duration of subscriptions to registration state is slightly longer, 3761 seconds. This helps avoid any potential problems with coupling of subscription and registration refreshes. Of course, clients MAY include an Expires header in the SUBSCRIBE request asking for a different duration.

4.5. NOTIFY Bodies

The SIP Events specification requires package definitions to describe the allowed set of body types in NOTIFY requests, and to specify the default value to be used when there is no Accept header in the SUBSCRIBE request.

The body of a notification of a change in registration state contains a registration information document. This document describes some or all of the contacts associated with a particular address-of-record. All subscribers and notifiers MUST support the "application/reginfo+xml" format described in Section 5. The subscribe request MAY contain an Accept header field. If no such header field is present, it has a default value of "application/reginfo+xml". If the header field is present, it MUST include "application/reginfo+xml", and MAY include any other types capable of representing registration information.

Of course, the notifications generated by the server MUST be in one of the formats specified in the Accept header field in the SUBSCRIBE request.

4.6. Notifier Processing of SUBSCRIBE Requests

The SIP Events framework specifies that packages should define any package-specific processing of SUBSCRIBE requests at a notifier, specifically with regards to authentication and authorization.

Registration state can be sensitive information. Therefore, all subscriptions to it SHOULD be authenticated and authorized before approval. Authentication MAY be performed using any of the techniques available through SIP, including digest, S/MIME, TLS or other transport specific mechanisms [1]. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that a user be allowed to subscribe to their own registration state. Such subscriptions are useful when there are many devices that represent a user, each of which needs to learn the registration state of the other devices. We also anticipate that applications and automata will frequently be subscribers to the

registration state. In those cases, authorization policy will typically be provided ahead of time.

4.7. Notifier Generation of NOTIFY Requests

The SIP Event framework requests that packages specify the conditions under which notifications are sent for that package, and how such notifications are constructed.

To determine when a notifier should send notifications of changes in registration state, we define a finite state machine (FSM) that represents the state of a contact for a particular address-of-record. Transitions in this state machine MAY result in the generation of notifications. These notifications will carry information on the new state and the event which triggered the state change. It is important to note that this FSM is just a model of the registration state machinery maintained by a server. An implementation would map its own state machines to this one in an implementation-specific manner.

4.7.1. The Registration State Machine

The underlying state machine for a registration is shown in Figure 1. The machine is very simple. An instance of this machine is associated with each address-of-record. When there are no contacts registered to the address-of-record, the state machine is in the init state. It is important to note that this state machine exists, and is well-defined, for each address-of-record in the domain, even if there are no contacts registered to it. This allows a user agent to subscribe to an address-of-record, and learn that there are no contacts registered to it. When the first contact is registered to that address-of-record, the state machine moves from init to active.

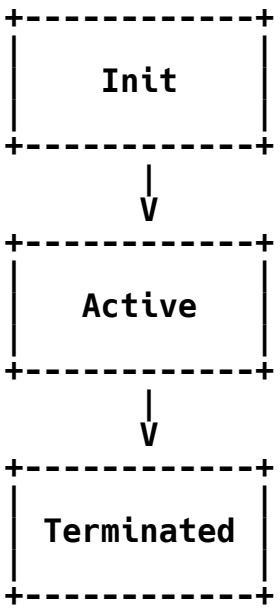

Figure 1: Registration State Machine

As long as there is at least one contact bound to the address-of-record, the state machine remains in the active state. When the last contact expires or is removed, the registration transitions to terminated. From there, it immediately transitions back to the init state. This transition is invisible, in that it MUST NOT ever be reported to a subscriber in a NOTIFY request.

> This allows for an implementation optimization whereby the registrar can destroy the objects associated with the registration state machine once it enters the terminated state and a NOTIFY has been sent. Instead, the registrar can assume that, if the objects for that state machine no longer exist, the state machine is in the init state.

In addition to this state machine, each registration is associated with a set of contacts, each of which is modeled with its own state machine. Unlike the FSM for the address-of-record, which exists even when no contacts are registered, the per-contact FSM is instantiated when the contact is registered, and deleted when it is removed. The diagram for the per-contact state machine is shown in Figure 2. This FSM is identical to the registration state machine in terms of its states, but has many more transition events.

When a new contact is added, the FSM for it is instantiated, and it moves into the active state. Because of that, the init state here is transient. There are two ways in which it can become active. One is

through an actual SIP REGISTER request (corresponding to the registered event), and the other is when the contact is created administratively, or through some non-SIP means (the created event).

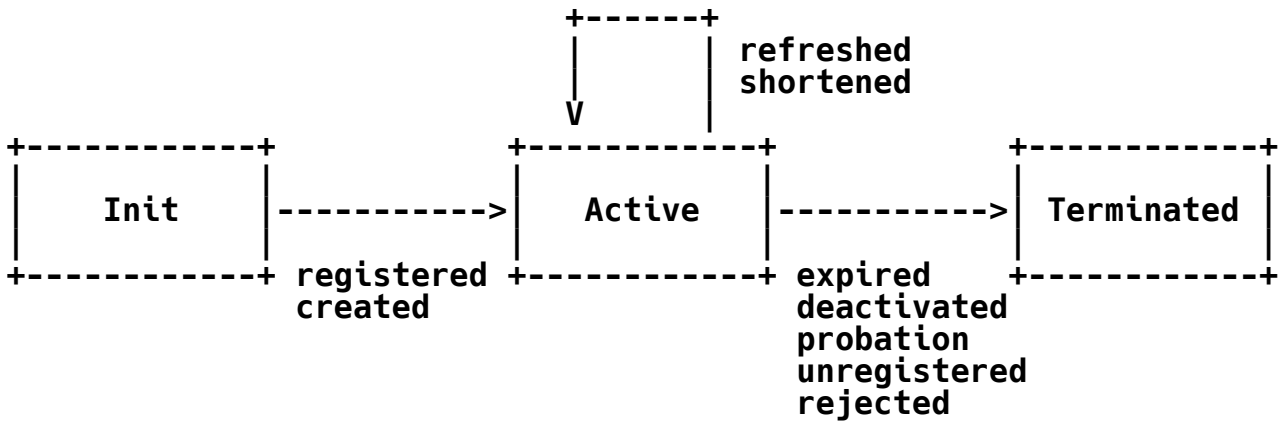

Figure 2: Contact State Machine

The FSM remains in the active state so long as the contact is bound to the address-of-record. When a contact is refreshed through a REGISTER request, the FSM stays in the same state, but a refreshed event is generated. Likewise, when an administrator modifies the expiration time of a binding (without deleting the binding) to trigger the contact to re-register and possibly re-authenticate, the FSM stays in the active state, but a shortened event is generated.

When the contact is no longer bound to the address-of-record, the FSM moves to the terminated state, and once a NOTIFY is sent, the state machine is destroyed. As a result, the terminated state is effectively transient. There are several reasons this can happen. The first is an expiration, which occurs when the contact was not refreshed by a REGISTER request. The second reason is deactivated. This occurs when the administrator has removed the contact as a valid binding, but still wishes the client to attempt to re-register the contact. In contrast, the rejected event occurs when an active contact is removed by the administrator, but re-registrations will not help to re-establish it. This might occur if a user does not pay their bills, for example. The probation event occurs when an active contact is removed by the administrator, and the administrator wants the client to re-register, but to do so at a later time. The unregistered event occurs when a REGISTER request sets the expiration time of that contact to zero.

4.7.2. Applying the state machine

The server MAY generate a notification to subscribers when any event occurs in either the address-of-record or per-contact state machines, except for the transition from terminated to init in the address-of-record state machine. As noted above, a notification MUST NOT be sent in this case. For other transitions, whether the server sends a notification or not is policy dependent. However, several guidelines are defined.

As a general rule, when a subscriber is authorized to receive notifications about a set of registrations, it is RECOMMENDED that notifications contain information about those contacts which have changed state (and thus triggered a notification), instead of delivering the current state of every contact in all registrations. However, notifications triggered as a result of a fetch operation (a SUBSCRIBE with Expires of 0) SHOULD result in the full state of all contacts for all registrations to be present in the NOTIFY.

4.8. Subscriber Processing of NOTIFY Requests

The SIP Events framework expects packages to specify how a subscriber processes NOTIFY requests in any package specific ways, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource. Typically, the NOTIFY will only contain information for contacts whose state has changed. To construct a coherent view of the total state of all registrations, the subscriber will need to combine NOTIFYs received over time. The details of this process depend on the document format used to convey registration state. Section 5 outlines the process for the application/reginfo+xml format.

4.9. Handling of Forked Requests

The SIP Events framework mandates that packages indicate whether or not forked SUBSCRIBE requests can install multiple subscriptions.

Registration state is normally stored in some repository (whether it be co-located with a proxy/registrar or in a separate database). As such, there is usually a single place where the contact information for a particular address-of-record is resident. This implies that a subscription for this information is readily handled by a single element with access to this repository. There is, therefore, no compelling need for a subscription to registration information to fork. As a result, a subscriber MUST NOT create multiple dialogs as a result of a single subscription request. The required processing to guarantee that only a single dialog is established is described in Section 4.4.9 of the SIP Events framework [2].

4.10. Rate of Notifications

The SIP Events framework mandates that packages define a maximum rate of notifications for their package.

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 5 seconds.

4.11. State Agents

The SIP Events framework asks packages to consider the role of state agents in their design.

State agents have no role in the handling of this package.

5. Registration Information

5.1. Structure of Registration Information

Registration information is an XML document [4] that MUST be well-formed and SHOULD be valid. Registration information documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying registration information documents and document fragments. The namespace URI for elements defined by this specification is a URN [5], using the namespace identifier 'ietf' defined by [6] and extended by [7]. This URN is:

urn:ietf:params:xml:ns:reginfo

A registration information document begins with the root element tag "reginfo". It consists of any number of "registration" sub-elements, each of which contains the registration state for a particular address-of-record. The registration information for a particular address-of-record MUST be contained within a single "registration" element; it cannot be spread across multiple "registration" elements within a document. Other elements from different namespaces MAY be present for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored. There are two attributes associated with the "reginfo" element, both of which MUST be present:

version: This attribute allows the recipient of registration information documents to properly order them. Versions start at 0, and increment by one for each new document sent to a subscriber. Versions are scoped within a

subscription. Versions MUST be representable using a 32 bit integer.

state: This attribute indicates whether the document contains the full registration state, or whether it contains only information on those registrations which have changed since the previous document (partial).

Note that the document format explicitly allows for conveying information on multiple addresses-of-record. This enables subscriptions to groups of registrations, where such a group is identified by some kind of URI. For example, a domain might define sip:allusers@example.com as a subscribable resource that generates notifications when the state of any address-of-record in the domain changes.

The "registration" element has a list of any number of "contact" sub-elements, each of which contains information on a single contact. Other elements from different namespaces MAY be present for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored. There are three attributes associated with the "registration" element, all of which MUST be present:

aor: The aor attribute contains a URI which is the address-of-record this registration refers to.

id: The id attribute identifies this registration. It MUST be unique amongst all other id attributes present in other registration elements conveyed to the subscriber within the scope of their subscription. In particular, if two URI identifying an address-of-record differ after their canonicalization according to the procedures in step 5 of Section 10.3 of RFC 3261 [1], the id attributes in the "registration" elements for those addresses-of-record MUST differ. Furthermore, the id attribute for a "registration" element for a particular address-of-record MUST be the same across all notifications sent within the subscription.

state: The state attribute indicates the state of the registration. The valid values are "init", "active" and "terminated".

The "contact" element contains a "uri" element, an optional "display-name" element, and an optional "unknown-param" element. Other elements from different namespaces MAY be present for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored. There are several attributes associated with the "contact" element which MUST be present:

id: The id attribute identifies this contact. It MUST be unique amongst all other id attributes present in other contact elements conveyed to the subscriber within the scope of their subscription. In particular, if the URI for two contacts differ (based on the URI comparison rules in RFC 3261 [1]), the id attributes for those contacts MUST differ. However, unlike the id attribute for an address-of-record, if the URI for two contacts are the same, their id attributes SHOULD be the same across notifications. This requirement is at SHOULD strength, and not MUST strength, since it is difficult to compute such an id as a function of the URI without retaining additional state. No hash function applied to the URI can, in fact, meet a MUST requirement. This is because equality of the SIP URI is not transitive. However, a hash function which includes unknown URI parameters (that is, any not defined in RFC 3261), will always result in a value that is the different if two URI are different, and usually the same if the URI are equal.

state: The state attribute indicates the state of the contact. The valid values are "active" and "terminated".

event: The event attribute indicates the event which caused the contact state machine to go into its current state. Valid values are registered, created, refreshed, shortened, expired, deactivated, probation, unregistered and rejected.

If the event attribute has a value of shortened, the "expires" attribute MUST be present. It contains an unsigned long integer which indicates the number of seconds remaining until the binding is due to expire. This attribute MAY be included with any event attribute value for which the state of the contact is active.

If the event attribute has a value of probation, the "retry-after" attribute MUST be present. It contains an unsigned long integer which indicates the amount of seconds after which the owner of the contact is expected to retry its registration.

The optional "duration-registered" attribute conveys the amount of time that the contact has been bound to the address-of-record, in seconds. The optional "q" attribute conveys the relative priority of this contact compared to other registered contacts. The optional "callid" attribute contains the current Call-ID carried in the REGISTER that was last used to update this contact, and the optional "cseq" attribute contains the last CSeq value present in a REGISTER request that updated this contact value.

The "uri" element contains the URI associated with that contact. The "display-name" element contains the display name for the contact. The "display-name" element MAY contain the xml:lang attribute to indicate the language of the display name.

The "unknown-param" element is used to convey contact header field parameters that are not specified in RFC 3261. One example are the user agent capability parameters specified in [11]. Each "unknown-param" element describes a single contact header field parameter. The name of the parameter is contained in the mandatory name attribute of the "unknown-param" element, and the value of the parameter is the content of the "unknown-param" element. For contact header field parameters that have no value, the content of the "unknown-param" element is empty.

5.2. Computing Registrations from the Document

Typically, the NOTIFY for registration information will only contain information about those contacts whose state has changed. To construct a coherent view of the total state of all registrations, a subscriber will need to combine NOTIFYs received over time. The subscriber maintains a table for each registration it receives information for. Each registration is uniquely identified by the "id" attribute in the "registration" element. Each table contains a row for each contact in that registration. Each row is indexed by the unique ID for that contact. It is conveyed in the "id" attribute of the "contact" element. The contents of each row contain the state of that contact as conveyed in the "contact" element. The tables are also associated with a version number. The version number MUST be initialized with the value of the "version" attribute from the "reginfo" element in the first document received. Each time a new document is received, the value of the local version number, and the "version" attribute in the new document, are compared. If the value in the new document is one higher than the local version number, the local version number is increased by one, and the document is processed. If the value 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, the document is processed, and the subscriber SHOULD generate a refresh request to trigger a full state notification. If the value in the document is less than the local version, the document is discarded without processing.

The processing of the document depends on whether it contains full or partial state. If it contains full state, indicated by the value of the "state" attribute in the "reginfo" element, the contents of all tables associated with this subscription are flushed. They are re-populated from the document. A new table is created for each "registration" element, and a new row in each table is created for

each "contact" element. If the reginfo contains partial state, as indicated by the value of the "state" attribute in the "reginfo" element, the document is used to update the existing tables. For each "registration" element, the subscriber checks to see if a table exists for that registration. This check is done by comparing the value in the "id" attribute of the "registration" element with the ID associated with the table. If a table doesn't exist for that registration, one is created. For each "contact" element in the registration, the subscriber checks to see whether a row exists for that contact. This check is done by comparing the ID in the "id" attribute of the "contact" element with the ID associated with the row. If the contact doesn't exist in the table, a row is added, and its state is set to the information from that "contact" element. If the contact does exist, its state is updated to be the information from that "contact" element. 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.

5.3. Example

The following is an example registration information document:

```
<?xml version="1.0"?>
    <reginfo xmlns="urn:ietf:params:xml:ns:reginfo"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  version="0" state="full">
      <registration aor="sip:user@example.com" id="as9"
                    state="active">
        <contact id="76" state="active" event="registered"
                  duration-registered="7322"
                  q="0.8">
                  <uri>sip:user@pc887.example.com</uri>
        </contact>
        <contact id="77" state="terminated" event="expired"
                  duration-registered="3600"
                  q="0.5">
                  <uri>sip:user@university.edu</uri>
        </contact>
      </registration>
    </reginfo>
```

5.4. XML Schema

The following is the schema definition of the reginfo format:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:reginfo"
xmlns:tns="urn:ietf:params:xml:ns:reginfo"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- This import brings in the XML language attribute xml:lang-->
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
  <xs:element name="reginfo">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="tns:registration" minOccurs="0"
maxOccurs="unbounded"/>
     <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:nonNegativeInteger"
use="required"/>
    <xs:attribute name="state" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="full"/>
       <xs:enumeration value="partial"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:complexType>
  </xs:element>
  <xs:element name="registration">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="tns:contact" minOccurs="0" maxOccurs="unbounded"/>
     <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="aor" type="xs:anyURI" use="required"/>
    <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="init"/>
       <xs:enumeration value="active"/>
       <xs:enumeration value="terminated"/>
      </xs:restriction>
```

```
     </xs:simpleType>
    </xs:attribute>
   </xs:complexType>
  </xs:element>
  <xs:element name="contact">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="uri" type="xs:anyURI"/>
     <xs:element name="display-name" minOccurs="0">
      <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:element name="unknown-param" minOccurs="0"
maxOccurs="unbounded">
      <xs:complexType>
       <xs:simpleContent>
        <xs:extension base="xs:string">
         <xs:attribute name="name" type="xs:string" use="required"/>
        </xs:extension>
       </xs:simpleContent>
      </xs:complexType>
     </xs:element>
     <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="state" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="active"/>
       <xs:enumeration value="terminated"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="event" use="required">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="registered"/>
       <xs:enumeration value="created"/>
       <xs:enumeration value="refreshed"/>
       <xs:enumeration value="shortened"/>
       <xs:enumeration value="expired"/>
       <xs:enumeration value="deactivated"/>
       <xs:enumeration value="probation"/>
```

```
       <xs:enumeration value="unregistered"/>
       <xs:enumeration value="rejected"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="duration-registered" type="xs:unsignedLong"/>
    <xs:attribute name="expires" type="xs:unsignedLong"/>
    <xs:attribute name="retry-after" type="xs:unsignedLong"/>
    <xs:attribute name="id" type="xs:string" use="required"/>
    <xs:attribute name="q" type="xs:string"/>
    <xs:attribute name="callid" type="xs:string"/>
    <xs:attribute name="cseq" type="xs:unsignedLong"/>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

6. Example Call Flow

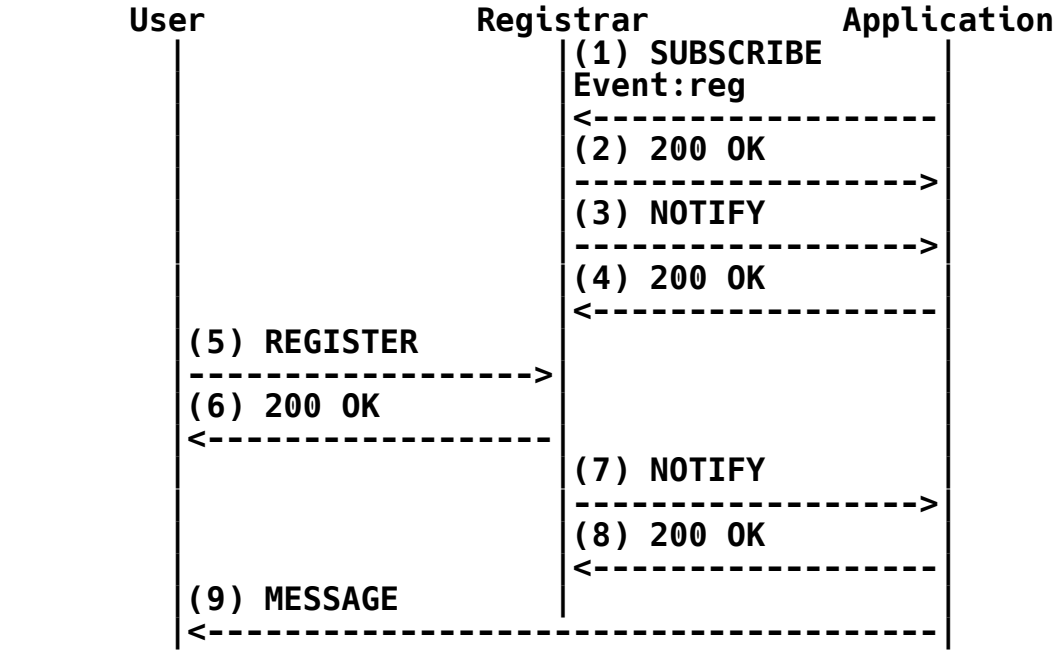

Figure 3: Example Call Flow

This section provides an example call flow, shown in Figure 3. It shows an implementation of the welcome notice application described in Section 3.3. First, the application SUBSCRIBEs to the registration event package for the desired user (1):

```
SUBSCRIBE sip:joe@example.com SIP/2.0
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bKnashds7
From: sip:app.example.com;tag=123aa9
To: sip:joe@example.com
Call-ID: 9987@app.example.com
CSeq: 9887 SUBSCRIBE
Contact: sip:app.example.com
Event: reg
Max-Forwards: 70
Accept: application/reginfo+xml
```

The registrar (which is acting as the notifier for the registration event package) generates a 200 OK to the SUBSCRIBE:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bKnashds7
  ;received=192.0.2.1
From: sip:app.example.com;tag=123aa9
To: sip:joe@example.com;tag=xyzygg
Call-ID: 9987@app.example.com
CSeq: 9987 SUBSCRIBE
Contact: sip:server19.example.com
Expires: 3600
```

The registrar then generates a notification (3) with the current state. Since there is no active registration, the state of the registration is "init":

```
NOTIFY sip:app.example.com SIP/2.0
Via: SIP/2.0/UDP server19.example.com;branch=z9hG4bKnasaii
From: sip:joe@example.com;tag=xyzygg
To: sip:app.example.com;tag=123aa9
Call-ID: 9987@app.example.com
CSeq: 1288 NOTIFY
Contact: sip:server19.example.com
Event: reg
Max-Forwards: 70
Content-Type: application/reginfo+xml
Content-Length: ...
```

```
<?xml version="1.0"?>
<reginfo xmlns="urn:ietf:params:xml:ns:reginfo"
             version="0" state="full">
  <registration aor="sip:joe@example.com" id="a7" state="init" />
</reginfo>
```

Later on, the user registers (5):

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pc34.example.com;branch=z9hG4bKnaaff
From: sip:joe@example.com;tag=99a8s
To: sip:joe@example.com
Call-ID: 88askjda9@pc34.example.com
CSeq: 9976 REGISTER
Contact: sip:joe@pc34.example.com
```

This results in a NOTIFY being generated to the application (7):

```
NOTIFY sip:app.example.com SIP/2.0
Via: SIP/2.0/UDP server19.example.com;branch=z9hG4bKnasaij
From: sip:joe@example.com;tag=xyzygg
To: sip:app.example.com;tag=123aa9
Call-ID: 9987@app.example.com
CSeq: 1289 NOTIFY
Contact: sip:server19.example.com
Event: reg
Max-Forwards: 70
Content-Type: application/reginfo+xml
Content-Length: ...

<?xml version="1.0"?>
<reginfo xmlns="urn:ietf:params:xml:ns:reginfo"
          version="1" state="partial">
  <registration aor="sip:joe@example.com" id="a7" state="active">
    <contact id="76" state="active" event="registered"
          duration-registered="0">
       <uri>sip:joe@pc34.example.com</uri>
    </contact>
  </registration>
</reginfo>
```

The application can then send its instant message to the device (9):

```
MESSAGE sip:joe@pc34.example.com SIP/2.0
Via: SIP/2.0/UDP app.example.com;branch=z9hG4bKnashds8
From: sip:app.example.com;tag=123aa10
To: sip:joe@example.com
Call-ID: 9988@app.example.com
CSeq: 82779 MESSAGE
Max-Forwards: 70
Content-Type: text/plain
Content-Length: ...

Welcome to the example.com service!
```

7. Security Considerations

Security considerations for SIP event packages are discussed in RFC 3265 [2], and those considerations apply here.

Registration information is sensitive, potentially private, information. Subscriptions to this event package SHOULD be authenticated and authorized according to local policy. Some policy guidelines are suggested in Section 4.6. In addition, notifications SHOULD be sent in such a way to ensure confidentiality, message integrity and verification of subscriber identity, such as sending subscriptions and notifications using a SIPS URL or protecting the notification bodies with S/MIME.

8. IANA Considerations

This document registers a new SIP Event Package, a new MIME type (application/reginfo+xml), and a new XML namespace.

8.1. SIP Event Package Registration

Package name: reg

Type: package

Contact: Jonathan Rosenberg, <jdrosen@jdrosen.net>

Published Specification: RFC 3680.

8.2. application/reginfo+xml MIME Registration

MIME media type name: application

MIME subtype name: reginfo+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [8].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [8].

Security considerations: See Section 10 of RFC 3023 [8] and Section 7 of this specification.

Interoperability considerations: none.

Published specification: This document.

Applications which use this media type: This document type is being used in notifications to alert SIP user agents that their registrations have expired and must be redone.

Additional Information:

Magic Number: None

File Extension: .rif or .xml

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 Registration for urn:ietf:params:xml:ns:reginfo

This section registers a new XML namespace, as per the guidelines in [7].

URI: The URI for this namespace is urn:ietf:params:xml:ns:reginfo.

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>Registration Information Namespace</title>
</head>
<body>
   <h1>Namespace for Registration Information</h1>
   <h2>urn:ietf:params:xml:ns:reginfo</h2>
   <p>See <a href="ftp://ftp.rfc-editor.org/in-notes/rfc3680.txt">
          RFC3680</a>.</p>
 </body>
</html>
END
```

9. References

9.1. 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., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[3] Bradner, S., "Key words for use in RFCs to indicate requirement levels", BCP 14, RFC 2119, March 1997.

[4] W. W. W. C. (W3C), "Extensible markup language (xml) 1.0." The XML 1.0 spec can be found at http://www.w3.org/TR/1998/REC-xml-19980210.

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

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

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

[8] Murata, M., St. Laurent, S. and D. Kohn, "XML media types", RFC 3023, January 2001.

9.2. Informative References

[9] Rosenberg, J., "Session initiation protocol (SIP) extensions for presence", Work In Progress.

[10] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[11] Schulzrinne, H. and J. Rosenberg, "Session initiation protocol (SIP) caller preferences and callee capabilities", Work In Progress.

[12] Mayer, G. and M. Beckmann, "Registration event package", Work In Progress.

10. Contributors

This document is based heavily on the registration event package originally proposed by Beckmann and Mayer in [12]. They can be contacted at:

Georg Mayer
Siemens AG
Hoffmannstr. 51
Munich 81359
Germany

EMail: Georg.Mayer@icn.siemens.de

Mark Beckmann
Siemens AG
P.O. Box 100702
Salzgitter 38207
Germany

EMail: Mark.Beckmann@siemens.com

Rohan Mahy provided editorial work in order to progress this specification. His contact address is:

Rohan Mahy
Cisco Systems
170 West Tasman Dr, MS: SJC-21/3/3

Phone: +1 408 526 8570
EMail: rohan@cisco.com

11. Acknowledgements

We would like to thank Dean Willis for his support.

12. Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054

EMail: jdrosen@dynamicsoft.com

13. Full Copyright Statement

Intellectual Property

Acknowledgement

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