

## SIP-Specific *Event* Notifications and *Event* Packages

**Abstract:** This is an essay discussing the extension to Session Initiation Protocol so as to introduce a capability in the SIP nodes to request for SIP notification on occurrence of a specific *Events*. This extension is Request for Comments [6] in the IETF's Network Working group. The RFC was posted on the IETF's list of RFCs on June 2002.

**Date** 12 May 2003  
**Submitted by** Suresh Chande  
**Email** [suresh.chande@cs.helsinki.fi](mailto:suresh.chande@cs.helsinki.fi)  
**Contact**

### Introduction

SIP is signalling protocol and provides control functions for the purpose of setting up, modifying and tearing down the multi-media sessions over the Internet. SIP is used mainly to identify the Location of an endpoint, check the willingness of the end point in setting up a session, exchange media information in order to set-up a media session and finally, modifying and tearing down an existing session between endpoints. SIP is a text-based protocol, similar to HTTP and SMTP, for initiating interactive communication sessions between users. Sessions include voice, video, chat, interactive games, and virtual reality [4]

SIP development originally started in the IETF's MMUSIC (Multi-Media Session Control) Working group and ended up as an Internet draft v1.0 in 1997, this later evolved and became the RFC 2543 (in March 1999). The RFC 2543 is now replaced by the currently published RFC 3625 (in June 2002).

In this following sections we will first look into the overview of the SIP and with the help of an example understand the SIP infrastructure and then proceed into the understanding the topic of this essay "SIP-Specific *Event* Notification and *Event* Packages"

### Session Initiation Protocol Overview

SIP is an application layer control protocol targeted towards end to end client and a server (proxies, gateways, routers, etc) signalling protocol. This protocol is used to establish a session with one or more participants; the sessions established can be modified and terminated. SIP can for example be utilised to set-up sessions such as telephone calls, multimedia conferencing, etc. It is typically useful for applications with a notion of sessions (networking games, videoconferences, etc).

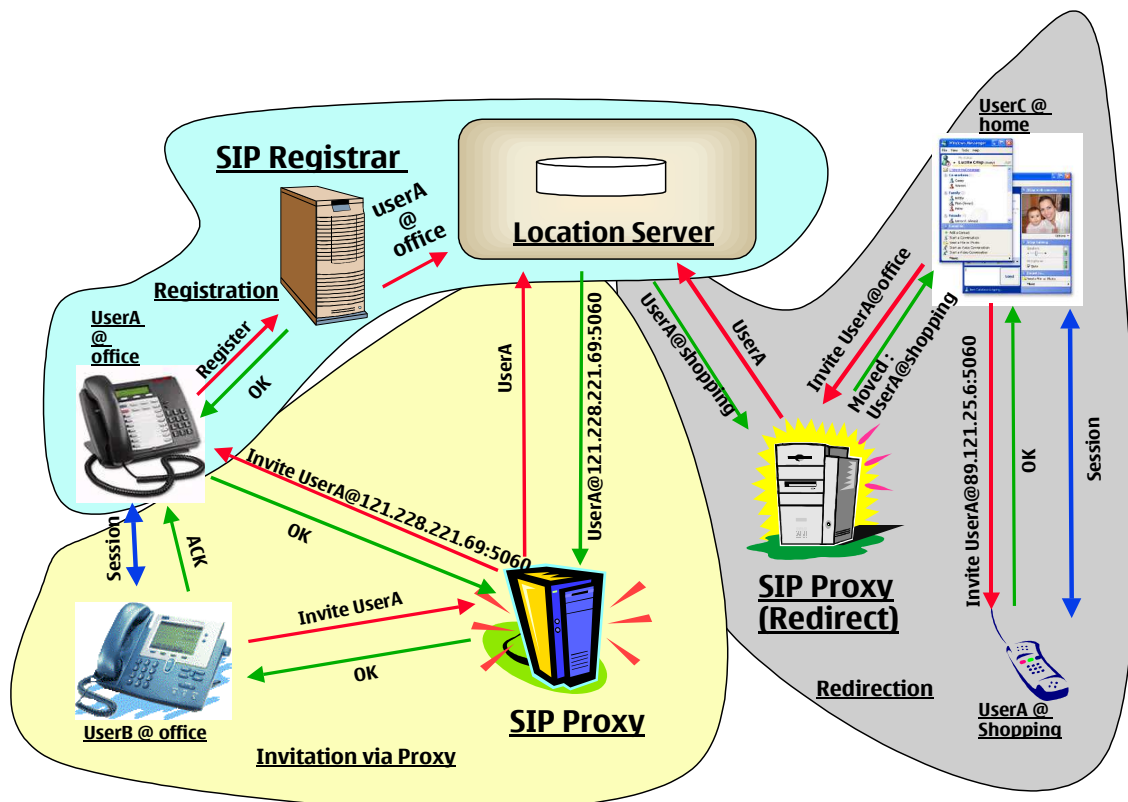
The process of setting up a session involves discovery of the user so as to deliver the description of the session that needs to be set-up between the user's useragent. (Software client utilised by the end user on his/her device or terminal to access and allow the execution of an application)

The applications on the Internet typically require setting up sessions to exchange data between two or multiple counter participants. The issues get sophisticated when we consider the various factors that can influence the setting up such session:

- User is addressable through multiple means
- User can utilise multiple user-agents
- User is moving between multiple internet nodes
- Useragent varies by functionality and the device supported features
- User can be reachable by multiple names
- User can sometimes like to communicate with multiple media simultaneously
- There are numerous protocols available to carry multiple-media elements

SIP works very well with a variety of internet protocols (TCP, UDP, IP) and provides means for multiple useragents to discover each other over the internet and establish a session between them taking into consideration various factors and characteristics of the user's user-agent and the preferred session characteristics.

SIP emphasises a rather stateless infrastructure making the overall network infrastructure lightweight and allowing it to be highly scalable system. SIP is a text based lightweight protocol. To understand the overall SIP system let us discuss The Figure 1 which depicts the overall SIP infrastructure and depicts the three scenarios of Registration, Redirection and Invitation to start-up a sessions between the two useragents in the question.



**Figure 1: Session Initiation Protocol - The overall Infrastructure**

SIP consists of the following basic components:

**SIP User Agent Server/Client:** These are the SIP client systems, they have the role of both a client and server. As a client they are capable of generating a SIP message or a SIP request and as server when another client requests for a specific service they generate a SIP message as a response to the request, hence acting as a SIP server as well.

**SIP Server:** SIP servers are entities available over the network providing services to the SIP Useragents to facilitate registration, finding and routing SIP messages to the right useragents. The servers support

TCP and UDP sort of connections to the useragents. The SIP servers are utilised for different purposes such as: Registrations, Gateway, Proxies, Redirecting, Subscription-Notifications, etc. Below are a few of them discussed:

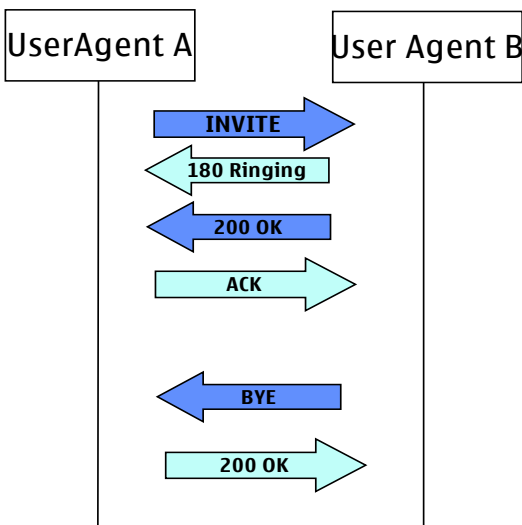
**SIP Registrar:** The SIP useragent client registers with the SIP Registrar so as to enable other SIP Useragents to find it. The SIP registrar provides the functionality to register and update information regarding the useragent the user is using. As depicted in the Figure 1 the Registrar will utilise the Location Server to persist such information about the useragent.

**SIP Proxy:** The SIP proxy server is utilised for forwarding the requests from the originating useragent to the target useragent. It utilises the location database to identify the right address of the target useragent. The SIP proxy by itself does not generate any SIP request but only facilitates the forwarding the request and delivering the responses generated by the SIP useragents.

The SIP Proxy can be of either stateful or stateless. The stateful SIP proxy can be utilised to enable reliable delivery of messages by re-transmitting the messages, broadcasting the request to multiple target recipient useragents.

**SIP Redirect Server:** This server is utilised to redirect the requesting useragent to the rightful target recipient address. This server is used to identify right useragent that a specific is currently using. This is the case where users are moving and switching between different useragents (multiple device ownership)

SIP useragents utilise SIP requests or methods to communicate with each other in-order to exchange messages and in-turn set-up a session between the two. The SIP consists of the following Request Methods **INVITE, ACK, BYE, CANCEL, OPTIONS and REGISTER**; and the following SIP extension methods: **INFO, COMET, PRACK, SUBSCRIBE, NOTIFY, MESSAGE**. The focus of this essay is on the extension methods *Subscribe* and *Notify*. Here below in the Figure 2 and Figure 3 we see a SIP invitation request method and the corresponding Message exchange sequence. The SIP Message consists of the Request Method, Headers and the payload message as shown in the Figure 3. The Request method in this case is INVITE, the request method has an URI (sip:pcsupport@helsinki.fi) to which this message has to be delivered. The headers are meant for the SIP servers and the target useragent to process. The Payload is meant for the target useragent, which contains information to set-up a session (RTP, Video call, etc).



**Figure 2: SIP Invitation Message sequence**



**Figure 3: SIP Invitation Message Structure**

This gives an overview of the SIP its various infrastructure elements and the functionality of them at a very brief level. The introduced elements above help set-up a session between two or more than two useragents, taking into consideration finding the rightful useragent provided that end-users can switch and move to more than one terminal.

The next section discusses about the extension of the SIP to enable subscription for the *Event* notification to be delivered to the SIP nodes.

## Notifications and *Event*

This extension to SIP enables the SIP nodes to request for the asynchronous notification of *Events* that might occur over the SIP systems. This benefits the SIP clients/useragents to receive regular notifications from the SIP services on any changes in state that may occur over the *Events* the user is interested to be notified about. Few of the examples could be: In the case of Instant messaging, change in presence information could be requested to notified (user is : free/ off the desk/ online / in a meeting, etc)

The notification will require the useragent to request for such notification or rather termed as *Subscribe* to specific *Events* to be notified. Once the changes occur the information about such *Events* can be notified by the corresponding useragent to the requested useragent.

The Figure 4 depicts the sequence of messages that are exchanged between the **Subscriber** and the **Notifier**. The *Subscriber* is a SIP node who is interested to receive notification of *Events*. The notification is dispatched to the *Subscriber* through the means of **Event Package**, which contains the information about the change in state. The *Subscribe* request message receives and immediate 200-class response code and then immediately followed by a *Notify* message. The *Notify* messages follow thereafter until the valid subscription period.

### Subscription

The *Subscribe* request message contains mandatory elements and optional elements. The mandatory elements are :

**Expires:** The value of which indicates the duration of the subscription. If it contains a "0" value this means the *Subscriber* has requested to be unSubscribed.

**Event Identification:** *Request URI, Event Type and Message body*, these three together can contain enough information to uniquely identify the resource for which the *Event* notification is desired.

**Event:** There can be only one *Event* header in the request which indicates which *Event* or class of *Events* one would like to *Subscribe* to.

The message can also define additional parameters for the *Event* header along with the semantics for such parameters (for example "id").

The optional element is **Accept** header element, which specifies the accepted body format that will arrive as a Notification message.

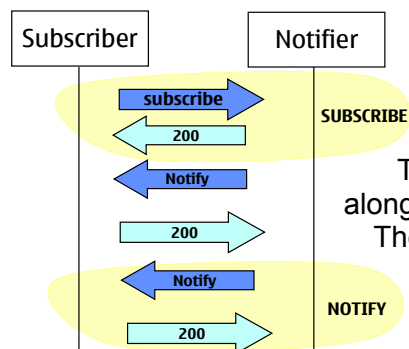


Figure 4: SIP Subscribe Request Method Example

### Request Processing at the *Notifier* :

The *Notifier* on receiving the *Subscribe* Message will process the message in the following manner:

- Check the *Event* header and if any error respond back with a error code "489", indicate that the specified *Event* or *Event* class is not understood.
- Will not allocate the subscription period longer then the requested period of time and in some cases even for a shorter period.
- Perform authentication and authorisation
- Stores the *Event* package and the *Event* header id as part of the subscription information

After completion of the request processing the *Notifier* will send a confirmation using a 200-class response indicating that the subscription request has been accepted and that the *Notify* will be sent immediately. The response codes generated and their meaning is as provided below:

200 - Request accepted and user is authorised to receive notifications

202 - Request has been understood and the authorisation may or may not be grated.

403 - Forbidden to request such subscription

603 - Decline to *Subscribe* such *Event* notification

423 - *Subscription Too Brief*, if the expires value is too small

The response message from the *Notifier* contains the **Expires** header entry, This is the duration for which the *Notifier* has actually accepted the subscription request. The duration can either be the same as in *Subscribe* request or then lesser than that.

### **Notification**

The *Notifier* sends a *Notify* message as and when there is a change in the state of the *Events* based on the previously accepted *Subscribe* request from the *Subscriber*. The *Notify* message will contain the following entries:

**Event Header:** This is similar the *Event* header in the *SUBSCRIBE* request. The *Event* header will have the same "id" parameter if it did exists in the original *Subscribe* request.

**Event Packages:** These could define the semantics associated with the body of the *Notify* message

**Body:** The body will contain the additional details about the occurred *Event*. The body will be complaint to one of the formats expressed in the Accept header in the *Subscribe* header, if it did exists in the request. It could either directly contain the state of the Subscribed resource or then contain an URI, which will contain the state.

**Subscription-State:** This can have a value of *Active*, *Pending* or *Terminated*. *Active* means that the subscription has been accepted and authorised. The *Pending* means that the subscription has been received but the policy information available is insufficient to accept or deny the subscription. The *Terminated* means that the subscription is no longer valid.

The *Notifier* sends an immediate *Notify* message to the *Subscriber* on accepting a *Subscribe* request. If there is a failure in the delivery of a *Notify* message due to time-out or if *Notifier* received a non-200 class response code or then if there was no "*Retry-After*" then the corresponding subscriptions will be removed.

The *Subscriber* on receiving a Notification, must check if the *Notify* was related to an existing *Subscribe* request by mapping it with the corresponding *Event* parameters submitted in the *Subscribe* request. The *Notify* is mapped to a specific *Subscribe* request if they contain the same "Call-Id", "From" and "To" that the *Subscriber* received in the 200-class response to a *Subscribe* request. If the *Notify* is not related to any previous *Subscribe* request then the *Subscriber* must send a "*481 Subscription does not exist*" response to the *Notifier*. If the *Notify* request contains a reason code then the *Subscriber* should follow specific steps to recover from the current status. The reason codes that can be returned and the corresponding actions that a *Subscriber* should take-up are listed below:

**Deactivated:** Subscription is terminated, the *Subscriber* should retry subscribing immediately or after the "retry-after" period if existed in the request

**Probation:** Subscription is terminated, *Subscriber* should retry subscribing later sometime or after the "retry-after" period, if existed in the request

**Rejected:** Subscription is terminated due to change in authorisation policy. Clients should not retry subscribing.

**Timeout:** Subscription is terminated due to timeout because subscription was not refreshed before it was expired, *Subscriber* should retry subscribing immediately or after the "retry-after" period if existed in the request

**Giveup:** Subscription is terminated due to not being able to authorise in a timely manner. *Subscriber* may retry subscribing immediately or after the "retry-after" period, if it existed in the request.

**Noresource:** Subscription is terminated due to the reason that the resource does not exist anymore. Clients should not retry subscribing.

### **Event Packages**

An *Event Package* is a set of additional specification, which contains the state information of specific resources to which a *Subscriber* has subscribed. The *Notifier* delivers an *Event* package in the *Notify* request to the *Subscriber*. The *Event* package in addition also specifies the syntax and semantics in order to convey such information. *Event* template-packages are composite kind of *Events* packages,

which define a set of state applied to other packages. They could also be applied to other template-packages.

The SIP nodes willing to *Subscribe* or *Notify* about specific set of *Events* will require to advertise the set of *Event* packages in message header, namely "*Allow Events*". This header contains a list of tokens, which indicates the *Event* packages supported by it. This will enable the useragents to render the corresponding interface to handle that specific *Event* package accordingly.

The utilisation of SIP *Event* notifications should be considered very moderately and in situations, which are not very frequent, as this could easily clog the SIP network. Like for example, **Location** of specific car on the dynamic geographical map, which is based on a GPS system, should not be considered to published a SIP *Event*, Whereas the presence information of a specific user could be considered as a right candidate for SIP *Events*.

**Event Packaging:** The *Event* information is to be packaged with careful consideration so as to keep the protocol light weights but yet conveys the relevant information with minimal content. The *Events* should normally contain the information to identify a specific *Event* and as well as the state of the *Event*. The information contained within the *Event* should be in the order of 1Kbytes. If the *Event* information is bigger, then the *Event* notification (*Notify*) request sent immediately after the *Subscribe* could contain the complete information. All other subsequent *Notify* request could only contain the delta *Event* information, which has actually changed, while the useragent client will merge the changes within to get the current state of the information.

The basic properties of the *Event* packages:

- **Name:** This is a token name to be used to identify the *Event* package and must contain information as registered at IANA
- **Parameters:** This contains the syntax and semantics of the *Event* header
- **Subscribe Bodies:** This is a mandatory section which defines the type of *Event* bodies that are expected in the *Subscribe* requests and the corresponding syntax and semantics.
- **Subscription Duration:** This defines the duration of the subscription requested.
- **Notify Bodies:** This defines the expected types of the *Event* bodies that could be expected in the *Notify* request. This also contains the syntax and semantics of such bodies.
- **Subscribe Request processing:** This defines the processing of the *Subscribe* request at the *Notifier*, specifically the authentication to be used and the authorisation to be applied to a specific *Subscribe* request.
- **Notify Request creation:** This defines the process by which a *Notifier* generates a *Notify* request. The definition contains the process of identifying which *Events* cause the *Notifier* to *Notify* how to compute the state information that can be notified.
- **Rate of Notification:** This defines the rate of notification by which a *Notifier* can send *Notify* requests.
- **URI based access to state Information:** Some *Event* packages can contain state information that are too huge to be conveyed over a SIP and under this case the SIP *Event* package can contain a URI using which the state information can be fetched.

### Security Considerations

The Subscription and Notification can reveal valuable and important information to SIP useragents and intermediaries. This information could reveal very sensitive information and hence must be protected from misuse of such information and allows the useragents and the complete SIP infrastructure to be secure. The several security aspects are necessary to be addressed; few of them are as discussed below, the ones being the most important:

- **Privacy:** Protection of (personal) information
- **Confidentiality:** Ensure that the message is only accessible to the intended target and that the message is indecipherable to other parties. The message could be encrypted
- **Access Control:** Allow only a set of authorised users to *Subscribe* to only a specific *Events* to which that specific user is allowed to do so.

- Denial of Service: Ensure against the intentional and unnecessary attacks of consuming resources of the *Subscribers* or *Notifiers* in such a manner that they are rendered unusable.
- Replay Attacks: Replaying of the *Notify* messages could be utilised to deliver wrong state information by nodes, which are a means of attacking the *Subscribers*. This can be addressed by enforcing authentication
- Man in the middle attacks: There could be attacks on the system even with including measures such as authentication. Taking into consideration message integrity can reduce this, by protecting the various fields of the *Subscribe* and *Notify* messages.

Concluding remarks:

SIP *Event* Notifications and packaging will enable end users to *Subscribe* to the services *Events* which are interesting to his/her. Since SIP being a lightweight protocol the utilisation of this protocol is very critical contain the level of state information in a balanced manner so as to serve the purpose of the SIP applications and as well as to respect the principles of the SIP network.

In this essay we have discussed about the general SIP overview, looked into the various SIP Request Methods. We have specifically elaborated and discussed about the SIP *Event Subscribe*, Notification and *Event* packaging mechanisms.

## 1. References

1. Understanding SIP, Dorgam Sisalem, Jiri Kuthan, Mobile Integrated Services,GMD Fokus, URL: <http://iptel.org/sip/siptutorial.pdf>
2. SIP: Understanding the Session Initiation Protocol, Alan B. Johnson, ISBN 1-58053-168-7
3. Internet Protocols for Mobile Computing, Internet Sessions:  
<http://www.cs.helsinki.fi/u/kraatika/Courses/MobInt/InternetSessions.pdf>
4. SIP Charter: <http://www.ietf.org/html.charters/sip-charter.html>
5. Session Initiation Protocol (SIP) - IETF's RFC 3261, June 2002, URL:  
<http://www.ietf.org/rfc/rfc3261.txt>
6. Session Initiation Protocol (SIP) - Specific *Event* Notification - IETF's RFC 3265, June 2002, URL:  
<http://www.ietf.org/rfc/rfc3265.txt>
7. The Session Initiation Protocol: Internet-Centric Signalling , October 2000 , IEEE Communications, Henning Schulzrinne,Columbia University  
Jonathan Rosenberg, dynamicsoft <http://www.comsoc.org/ci/private/2000/oct/schulzrinne.html>
8. Good resource for SIP related information , URL: <http://www1.cs.columbia.edu/sip/>
9. SIP Forum, a non profit association whose mission is to promote awareness and provide information about the benefits and capabilities that are enabled by SIP, URL:  
<http://www.sipforum.org/>