A Session Initiation Protocol (SIP) Event Package for Debugging
draft-dawes-sipping-debug-event-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 4, 2008.

Abstract

This document defines a Session Initiation Protocol (SIP) event package for debugging. SIP requests and responses for session setup can traverse a number of network entities, which might be geographically spread over a large area. This document provides a means by which logging of requests and responses can be configured on a per-entity basis.
# Table of Contents

1. Introduction ................................................. 3  
   1.1. Requirements Language ................................. 3  
2. Usage Scenarios ................................. 3  
   2.1. Debugging a reported fault ......................... 3  
   2.2. Regression testing ............................... 4  
3. Principle of Operation ................................ 4  
   3.1. Minimum debugging configuration ................. 5  
4. Package Definition ..................................... 5  
   4.1. Event Package Name ................................ 5  
   4.2. Event Package Parameters ....................... 5  
   4.3. SUBSCRIBE Bodies ................................ 6  
   4.4. Subscription Duration ............................ 6  
   4.5. NOTIFY Bodies .................................. 6  
   4.6. Notifier Processing of SUBSCRIBE Requests ....... 7  
   4.7. Notifier Generation of NOTIFY Requests ............ 7  
      4.7.1. The Debug Configuration State Machine ...... 8  
      4.7.2. Applying the State Machine ................. 9  
   4.8. Subscriber Processing of NOTIFY Requests ...... 9  
   4.9. Handling of Forked Requests .................... 9  
   4.10. Rate of Notifications ............................ 10  
   4.11. State Agents ................................. 10  
5. Debug Configuration Information ....................... 10  
   5.1. Structure of Debug Configuration .................. 10  
   5.2. Computing Debug Configuration from the Document ... 12  
   5.3. Example ........................................ 13  
   5.4. XML Schema ...................................... 13  
6. Example Call Flows ..................................... 16  
   6.1. Subscription to debug-event ....................... 17  
   6.2. Incoming session ................................ 21  
   6.3. Outgoing session ................................ 23  
   6.4. Prompting a user agent to subscribe to debug-event 27  
7. Acknowledgements ...................................... 27  
8. IANA Considerations .................................. 28  
   8.1. SIP Event Package Registration ................... 28  
   8.2. application/debuinfo+xml MIME Registration ...... 28  
9. Security Considerations ................................ 29  
10. References .......................................... 29  
   10.1. Normative References ............................. 29  
   10.2. Informative References ........................... 30  
Appendix A. Additional Stuff ............................. 30  
Author’s Address ........................................ 30  
Intellectual Property and Copyright Statements .......... 31
1. Introduction

The Session Initiation Protocol (SIP) [RFC 3261] provides all of the functions needed for the establishment and maintenance of communications sessions between users. Registration establishes an address at which a user can be contacted to set up communication sessions.

It is possible for that session setup might fail, for example due to a network fault or mis-configuration, or that poor network performance causes long setup delays. In such circumstances, it is useful to be able to analyse SIP requests and responses end-to-end from the UAC to the UAS (prd check terminology), which entails logging of requests and responses by entities along the signalling route.

This document describes an event package that enables SIP UAs, proxies and B2BUAs to be dynamically configured to log SIP requests and responses.

Logged signalling is identified across different entities with the help of a private header defined in [draft-dawes-sipping-debug-id]

1.1. Requirements Language

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].

2. Usage Scenarios

This event package supplies configuration for two broad applications, debugging user reported faults and regression testing of a SIP network.

2.1. Debugging a reported fault

If a user is unable to set up a session using SIP, a number of causes are possible. For example, the user is unable to perform registration, or the user terminal is forming requests incorrectly or a SIP proxy in the network is preventing session setup. In such cases, it is useful to be able to construct a complete picture of SIP requests and responses related to a session, in time order.
2.2. Regression testing

Networks and user terminals are subject to continuous upgrading and modification as the software that they run is developed. However, such software upgrades can have the unwanted side effect of not interworking with existing SIP proxies or user agents. For software upgrades, either of user agents or of core network entities, it is useful to perform a set of regression tests after an upgrade and compare signalling logged before and after the upgrade.

3. Principle of Operation

Debugging can be understood by the simple state machine below, which applies to any SIP UA or Proxy.

![Debugging State Machine Diagram]

The debugging configuration is an XML document described by the. 

Dawes
Expires November 4, 2008
[Page 4]
schema in this document. Supply of configuration causes debugging state to change from Inactive to Active. Configuration defines one or more start trigger events, which cause debugging state to change from Active to Logging when they are detected. A start trigger event is typically a user identity, in the SIP From: or To: header field, plus a 6-digit hexadecimal reference number. Similarly, configuration contains one or more stop triggering events, which cause debugging state to change from Logging to Active when they are detected. A stop trigger event is typically the expiry of a timer or the end of a SIP session.

3.1. Minimum debugging configuration

In order to uniquely identify signalling logged at different entities as belonging to the same session, the minimum set of debugging configuration is a <user-id> element and a <debug-id> element.

4. Package Definition

This section fills in the details needed to specify an event package as defined in Section 4.4 of RFC 3265 [RFC3265].

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 "debug". As specified in RFC 3265 [RFC3265], this value appears in the Event header present in SUBSCRIBE and NOTIFY requests.

Example:

Event: debug

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 debug 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 debug 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 part of the debug configuration for the resource being subscribed to. Those notifications only contain information on the configuration elements whose state has changed.

- Notifications triggered from a SUBSCRIBE contain full state (all debug configuration 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.

Debug configuration state changes as the need for debugging arises, either to investigate a user-reported fault or perform regression testing. The debug configuration for a user or users is updated by administrative means.

Since configuration of debugging will be followed quickly by the debugging itself, the default duration of subscriptions to debug configuration is 43200 seconds. 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 debug configuration state
contains a debug configuration information document. This document describes some or all of the debugging configuration associated with a particular address-of-record. All subscribers and notifiers MUST support the "application/debuginfo+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/debuginfo+xml". If the header field is present, it MUST include "application/debuginfo+xml", and MAY include any other types capable of representing debugging 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.

Debug configuration 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 RFC3261 [RFC3261]. 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 debug configuration state. Debugging relies upon a user agent including a network-provided debug-id, and this is most easily provided by allowing the user to subscribe to debug-event. We also anticipate that applications and automata will frequently be subscribers to the debug configuration 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 debug configuration state, we define a finite state machine (FSM) that represents the state of debug configuration 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 debug configuration 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 Debug Configuration State Machine

The underlying state machine for debug configuration is shown in the figure below. The machine is very simple. An instance of this machine is associated with each address-of-record. When there is no debugging configuration defined for an 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 is no debug configuration defined for it. This allows a user agent to subscribe to an address-of-record, and learn that there is no debug configuration for it. When debug configuration is added for that address-of-record, the state machine moves from init to active.

```
+------------+
|            |
|    Init    |
|            |
+------------+
|            |
V
+------------+
|            |
|   Active   |
|            |
+------------+
|            |
V
+------------+
|            |
| Terminated |
|            |
+------------+
```

Figure 2: Debug Configuration State Machine

As long as there is debugging configuration for the address-of-record, the state machine remains in the active state. When the last debug configuration expires or is removed, the debug configuration 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 debug configuration 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.

4.7.2. Applying the State Machine

The server MAY generate a notification to subscribers when any event occurs in the debug configuration state machine, except for the transition from terminated to init. 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.

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. For debug configuration, the NOTIFY will contain all information for the address of record whose configuration has changed.

4.9. Handling of Forked Requests

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

Debug configuration 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 debug configuration 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 debug configuration 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 RFC3265 [RFC3265].
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. A change of debug configuration is usually followed by a session to be debugged, which takes significant time. Even during regression testing, in which a number of consecutive sessions might be debugged, notifications are punctuated by the sessions or standalone transactions to be debugged. It is therefore RECOMMENDED that the server not generate notifications for a single subscriber at a rate faster than once every 60 seconds.

4.11. State Agents

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

Debug configuration information is passed from a network management server to the SIP registrar, which hosts the debug configuration event package. The details of how debug configuration information is passed to the SIP registrar is outside the scope of this document.

5. Debug Configuration Information

5.1. Structure of Debug Configuration

Debug configuration is an XML document Canonical XML Version 1.0 [W3C.xml-c14n] that MUST be well-formed and SHOULD be valid. Debug configuration documents MUST be based on XML 1.0 and MUST be encoded using UTF-8. This specification makes use of XML namespaces for identifying debug configuration documents and document fragments. The namespace URI for elements defined by this specification is a URN RFC 2141 [RFC2141], using the namespace identifier 'ietf' defined by RFC 2648 [RFC2648] and extended by RFC 3688 [RFC3688]; This URN is:

urn:ietf:params:xml:ns:debuginfo

A debug configuration document begins with the root element tag "debuginfo". It consists of any number of "debugconfig" sub-elements, each of which contains descriptions of sessions or standalone transactions that are to be debugged for a particular address-of-record. The debug configuration information for a particular address-of-record MUST be contained within a single "debuginfo" element; it cannot be spread across multiple "debuginfo"
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 "debuginfo" element, both of which MUST be present:

version: This attribute allows the recipient of debug configuration 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 debug configurations 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 debug configurations, where such a group is identified by some kind of URI. For example, a domain might define sip:allusers@example.com as a resource that can be subscribed to and generates notifications when the state of any address-of-record in the domain changes.

The "debugconfig" element has a list of any number of "session" sub-elements, each of which contains information on a single session or standalone transaction. 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 "debug configuration" 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 debug configuration. It MUST be unique amongst all other id attributes present in other debug configuration 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 [RFC3261], the id attributes in the "debug configuration" elements for those addresses-of-record MUST differ. Furthermore, the id attribute for a "debug configuration" 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 debug configuration. The valid values are "init", "active" and "terminated".

The "session" element contains a "debug-id" element and a "start trigger" element, "stop trigger" element, and "control" element. Other elements from different namespaces MAY be present for the purposes of extensibility; elements or attributes from unknown namespaces MUST be ignored. There is one attribute associated with the "session" element which MUST be present:

id: The "id" attribute identifies this session. It MUST be unique amongst all other id attributes present in other debug configuration elements conveyed to the subscriber within the scope of their subscription.

5.2. Computing Debug Configuration from the Document

Typically, the NOTIFY for debug configuration information will only contain information about those sessions 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 debug configuration it receives information for. Each debug configuration is uniquely identified by the "id" attribute in the "debug configuration" element. Each table contains a row for each session in that debug configuration. Each row is indexed by the unique debug-id for that session. It is conveyed in the "id" attribute of the "session" 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 "debuginfo" 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 "debuginfo" 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 
"debugconfig" element, and a new row in each table is created for 
each "session" element. If the debuginfo contains partial state, as 
indicated by the value of the "state" attribute in the "debuginfo" 
element, the document is used to update the existing tables. For 
each "debugconfig" element, the subscriber checks to see if a table 
exists for that debug configuration. This check is done by comparing 
the value in the "aor" attribute of the "debugconfig" element with 
the aor associated with the table. If a table doesn’t exist for that 
registration, one is created. For each "session" element in the 
debug configuration, the subscriber checks to see whether a row 
exists for that session. This check is done by comparing the ID in 
the "id" attribute of the "session" element with the ID associated 
with the row. If the session doesn’t exist in the table, a row is 
added, and its state is set to the information from that "session" 
element. If the session does exist, its state is updated to be the 
information from that "session" 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 debug configuration information document:

```xml
<?xml version="1.0"?><debuginfo xmlns="urn:ietf:params:xml:ns:debuginfo" version="0" state="full">
  <debugconfig aor="alice@atlanta.com" id="r01" state="active" expires="10000">
    <session id="r03">
      <start-trigger>
        <debug-id>1A346D</debug-id>
        <user-id>alice@atlanta.com</user-id>
      </start-trigger>
      <stop-trigger>
        <reason>dialog-established</reason>
      </stop-trigger>
      <control>
        <trace-depth>minimum</trace-depth>
      </control>
    </session>
  </debugconfig>
</debuginfo>
```

5.4. XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?><xs:schema targetNamespace="urn:ietf:params:xml:ns:debuginfo"
```
<xs:element name="debuginfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="tns:debugconfig" minOccurs="0" maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="version" use="required"/>
    <xs:attribute ref="state" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="debugconfig">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="session"/>
    </xs:sequence>
    <xs:attribute ref="aor" use="required"/>
    <xs:attribute ref="id" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="session">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="start-trigger"/>
      <xs:element ref="stop-trigger"/>
      <xs:element ref="control"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="start-trigger">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="from"/>
      <xs:element ref="to"/>
      <xs:element ref="icsi"/>
      <xs:element ref="iari"/>
      <xs:element ref="method"/>
      <xs:element ref="time"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
6. Example Call Flows
6.1. Subscription to debug-event

<table>
<thead>
<tr>
<th>User</th>
<th>Proxy</th>
<th>Registrar</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) REGISTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) REGISTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3) 200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) 200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5) SUBSCRIBE</td>
<td>Event:debug</td>
<td></td>
</tr>
<tr>
<td>(6) 200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(7) NOTIFY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(8) 200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(9) SUBSCRIBE</td>
<td>Event:debug</td>
<td></td>
</tr>
<tr>
<td>(10) 200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(11) NOTIFY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(12) 200 OK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alice registers (1) and (2)

REGISTER sip:rl.atlanta.com SIP/2.0
Via: SIP/2.0/UDP u1.atlanta.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Alice <sip:alice@atlanta.com>
From: Alice <sip:alice@atlanta.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:alice@192.0.2.4>
Expires: 7200
Content-Length: 0

Registration is successful (3) and (4)
Since the Proxy is now aware of Alice’s registration, it is possible for
the Proxy to subscribe to Alice’s debug configuration now. Alice then
subscribes to her debug configuration (5):

```
SUBSCRIBE sip:alice@atlanta.com SIP/2.0
Via: SIP/2.0/UDP u1.atlanta.com;branch=z9hG4bKnashds7
From: sip:alice@atlanta.com;tag=123aa9
To: sip:alice@atlanta.com
Call-ID: 9987@u1.atlanta.com
CSeq: 9887 SUBSCRIBE
Contact: sip:u1.atlanta.com
Event: debug
Max-Forwards: 70
Accept: application/debuginfo+xml
```

The Registrar (which is acting as the notifier for the debug event
package) generates a 200 OK to the SUBSCRIBE (6):

```
200 OK Registrar -> Alice
SIP/2.0 200 OK
Via: SIP/2.0/UDP s1.atlanta.com;branch=z9hG4bKnashds7
;received=192.0.2.1
From: sip:alice@atlanta.com;tag=123aa9
To: sip:alice@atlanta.com;tag=xyzygg
Call-ID: 9987@u1.atlanta.com
CSeq: 9887 SUBSCRIBE
Contact: sip:s1.atlanta.com
Expires: 3600
```

The registrar then generates a notification (7) with the current
debugging configuration information for the address of record
“alice@atlanta.com”.

```
NOTIFY Registrar -> Alice
NOTIFY sip:u1.atlanta.com SIP/2.0
Via: SIP/2.0/UDP r1.atlanta.com;branch=z9hG4bKnasaii
From: sip:r1.atlanta.com;tag=xyzygg
```
To: sip:alice@atlanta.com;tag=123aa9
Call-ID: 9987@u1.atlanta.com
CSeq: 1288 NOTIFY
Contact: sip:r1.atlanta.com
Event: debug
Max-Forwards: 70
Content-Type: application/debuginfo+xml
Content-Length: ...

<?xml version="1.0"?>
<debuginfo xmlns="urn:ietf:params:xml:ns:debuginfo"
version="0" state="full">
<debugconfig aor="alice@atlanta.com" id="r01" state="active"
expires="10000">
<session id="r03">
<start-trigger>
<method>INVITE</method>
[from]alice@atlanta.com[from]
</start-trigger>
<stop-trigger>
<time-period>P7M30S</time-period>
</stop-trigger>
<control>
<trace-depth>minimum</trace-depth>
<debug-id>1A346D</debug-id>
</control>
</session>
</debugconfig>
</debuginfo>

NOTE: If multiple sessions are to be debugged, then multiple
<session></session> elements are included in the XML, each one with a
different debug-id attribute.

Alice’s user agent responds to the NOTIFY request (8):

200 OK Alice -> Registrar
SIP/2.0 200 OK
Via: SIP/2.0/UDP r1.atlanta.com;branch=z9hG4bKnashds7
;received=192.0.2.1
From: sip:r1.atlanta.com;tag=xyzygg
To: sip:alice@atlanta.com;tag=123aa9
Call-ID: 9987@l1.atlanta.com
CSeq: 1288 NOTIFY
Contact: sip:u1.atlanta.com

Typically, proxy pl.atlanta.com will already be subscribed to the debug
event package for sip:alice@atlanta.com. If not, proxy pl.atlanta.com
subscribes to the debug configuration for Alice (9):

```
SUBSCRIBE sip:alice@atlanta.com SIP/2.0
Via: SIP/2.0/UDP p1.atlanta.com;branch=z9hG4bKnashds7
From: sip:p1.atlanta.com;tag=123aa9
To: sip:alice@atlanta.com
Call-ID: 9987@p1.atlanta.com
CSeq: 9887 SUBSCRIBE
Contact: sip:p1.atlanta.com
Event: debug
Max-Forwards: 70
Accept: application/debuginfo+xml
```

The registrar (which is acting as the notifier for the debugging event package) generates a 200 OK to the SUBSCRIBE (10):

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP p1.atlanta.com;branch=z9hG4bKnashds7
;received=192.0.2.1
From: sip:p1.atlanta.com;tag=123aa9
To: sip:alice@atlanta.com;tag=xyzygg
Call-ID: 9987@p1.example.com
CSeq: 9987 SUBSCRIBE
Contact: sip:r1.atlanta.com
Expires: 3600
```

The registrar then generates a notification to the proxy with debugging configuration for the proxy (1):

```
NOTIFY sip:p1.atlanta.com SIP/2.0
Via: SIP/2.0/UDP r1.atlanta.com;branch=z9hG4bKnasaii
From: sip:p1.atlanta.com;tag=123aa9
To: sip:alice@atlanta.com;tag=xyzygg
Call-ID: 9987@p1.example.com
CSeq: 1288 NOTIFY
Contact: sip:r1.atlanta.com
Event: debug
Max-Forwards: 70
Content-Type: application/debuginfo+xml
Content-Length:

<?xml version="1.0"?>
<debuginfo xmlns="urn:ietf:params:xml:ns:debuginfo"
    version="0" state="full">
    <debugconfig aor="alice@atlanta.com" id="r01" state="active"
        expires="10000">
    <session id="r03">
        <start-trigger>
```

Dawes Expires November 4, 2008 [Page 20]
The XML documents for the user agent and the proxy may be different. In this example, the value in the &lt;debug-d&gt; element is identical for the user agent and the proxy, but for the proxy it is part of the start trigger event, whereas for the user agent it is control information.

### 6.2. Incoming session

In order to log signalling for the incoming session shown below, it is necessary to configure debugging on the registrar, proxy, and user agent. The P-Debug-ID private header shown in the figure below is defined in [draft-dawes-sipping-debug-id](draft-dawes-sipping-debug-id)

<table>
<thead>
<tr>
<th>User</th>
<th>Proxy</th>
<th>Registrar</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1.atlanta.com</td>
<td>p1.atlanta.com</td>
<td>r1.atlanta.com</td>
</tr>
<tr>
<td>(1) INVITE</td>
<td></td>
<td>(1) INVITE</td>
</tr>
<tr>
<td></td>
<td>(2) INVITE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P-Debug-ID:BB947A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;--------------</td>
<td></td>
</tr>
<tr>
<td>(3) INVITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Debug-ID:BB947A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) 200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Debug-ID:BB947A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The registrar r1.atlanta.com is triggered to begin logging signalling by a start trigger event of the INVITE method and the value alice@atlanta.com in the To: header field. The debugging configuration in the registrar is shown below.

```xml
<?xml version="1.0"?>
<debuginfo xmlns="urn:ietf:params:xml:ns:debuginfo" version="0" state="full">
  <debugconfig aor="alice@atlanta.com" id="r01" state="active" expires="10000">
    <session id="r03">
      <start-trigger>
        <method>INVITE</method>
        <to>alice@atlanta.com</to>
      </start-trigger>
      <stop-trigger>
        <reason>dialog-established</reason>
      </stop-trigger>
      <control>
        <trace-depth>minimum</trace-depth>
        <debug-id>BB947A</debug-id>
      </control>
    </session>
  </debugconfig>
</debuginfo>
```

The registrar detects an INVITE request with "alice@atlanta.com" in the To: header field. The registrar therefore starts logging and adds a P-Debug-ID header field containing the value in the <debug-id/> element in its debugging configuration. The registrar then forwards the request.

The debugging configuration in the proxy is shown below.
<?xml version="1.0"?>
<debuginfo xmlns="urn:ietf:params:xml:ns:debuginfo" version="0" state="full">
<debugconfig aor="alice@atlanta.com" id="r01" state="active" expires="10000">
<session id="r03">
<start-trigger>
<debug-id>BB947A</debug-id>
<to>alice@atlanta.com</to>
</start-trigger>
<stop-trigger>
<reason>dialog-established</reason>
</stop-trigger>
<control>
<trace-depth>minimum</trace-depth>
</control>
</session>
</debugconfig>
</debuginfo>

The request arriving at the proxy matches the values configured in its <start-trigger> element: a P-Debug-ID header containing the configured value and "alice@atlanta.com" in the To: header field. The proxy therefore starts logging and forwards the request.

The User Agent has the same <start-trigger> element as the proxy and therefore starts logging. The user agent copies the P-Debug-ID header field into the 200 OK response.

The User Agent, proxy and registrar all have the same <stop-trigger> element, and logging will stop as the 200 OK passes each entity, thereby establishing the dialog.

6.3. Outgoing session

In order to log signalling for the outgoing session shown below, it is necessary to configure debugging on the registrar, proxy, and user agent.
Alice sends a MESSAGE request that is debugged:

```
MESSAGE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP u1.atlanta.com;branch=z9hG4bKnashds8
From: sip:alice@atlanta.com;tag=123aa10
To: sip:bob@biloxi.com
P-Preferred-Identity: alice@atlanta.com
Call-ID: 9901@r1.atlanta.com
CSeq: 82779 MESSAGE
Max-Forwards: 70
P-Debug-ID: 9E2836
Content-Type: text/plain
Content-Length: ...
```

```
MESSAGE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP p1.atlanta.com;branch=z9hG4bKnashds8
Via: SIP/2.0/UDP u1.atlanta.com;branch=z9hG4bKnashds8
From: sip:alice@atlanta.com;tag=123aa10
To: sip:bob@biloxi.com
P-Preferred-Identity: alice@atlanta.com
Call-ID: 9901@nms1.atlanta.com
CSeq: 82779 MESSAGE
Max-Forwards: 69
P-Debug-ID: 9E2836
Content-Type: text/plain
Content-Length: ...
```
The user agent u1.atlanta.com is triggered to begin logging signalling by a start trigger event of the MESSAGE method and the value alice@atlanta.com in the From: header field. The <start-trigger> element in the user agent debugging configuration is shown below.
The user agent detects the configured start trigger event when it originates a MESSAGE request with "alice@atlanta.com" in the From: header field. The user agent therefore starts logging and adds a P-Debug-ID header field containing the value in the <debug-id/> element in its debugging configuration <control/> element. The user agent then forwards the request.

The debugging configuration in the proxy is shown below.

```xml
<?xml version="1.0"?>
<debuginfo xmlns="urn:ietf:params:xml:ns:debuginfo" version="0" state="full">
  <debugconfig aor="alice@atlanta.com" id="r01" state="active" expires="10000">
    <session id="r03">
      <start-trigger>
        <method>MESSAGE</method>
        <from>alice@atlanta.com</from>
      </start-trigger>
      <stop-trigger>
        <time-period>P7M30S</time-period>
      </stop-trigger>
      <control>
        <trace-depth>minimum</trace-depth>
      </control>
    </session>
  </debugconfig>
</debuginfo>
```
The request arriving at the proxy matches the values configured in its `<start-trigger>` element: a P-Debug-ID header containing the configured value and "alice@atlanta.com" in the From: header field. The proxy therefore starts logging and forwards the request.

The registrar has the same `<start-trigger>` element as the proxy and therefore starts logging and forwards the request.

The User Agent, proxy and registrar all have the same `<stop-trigger>` element, and logging will stop after a time duration of 7 minutes 30 seconds.

6.4. Prompting a user agent to subscribe to debug-event

Troubleshooting and regression testing will be quite rare events and only involve specific entities, therefore it is inefficient for all user agents to be subscribed to the debug event package all the time. A user agent can be prompted to subscribe to its own debug event package by an empty P-Debug-ID header field in a 200 OK response to a REGISTER request. The signalling is shown below.

```
User    | Proxy         | Registrar
ul.atlanta.com | pl.atlanta.com | r1.atlanta.com

(1) REGISTER
(2) REGISTER
(3) 200 OK
P-Debug-ID:
(4) SUBSCRIBE
Event: debug
```

7. Acknowledgements

This template was derived from an initial version written by Pekka Savola and contributed by him to the xml2rfc project.
8. IANA Considerations

All drafts are required to have an IANA considerations section (see the update of RFC 2434 [I-D.narten-iana-considerations-rfc2434bis] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

8.1. SIP Event Package Registration

Package name: debug
Type: package
Contact: Peter Dawes, peter.dawes@vodafone.com

8.2. application/debuinfo+xml MIME Registration

MIME media type name: application
MIME subtype name: debuginfo+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 provide SIP entities with configuration data for debugging SIP signaling.
Additional Information:
Magic Number: None
File Extension: .rif or .xml
9. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

10. References

10.1. Normative References


[draft-dawes-sipping-debug-id]
10.2. Informative References

[I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs",
draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.


Appendix A. Additional Stuff

This becomes an Appendix.

Author’s Address

Peter Dawes
Vodafone
Newbury, Berkshire RG14 2FN
UK

Phone: +44 7717 275009
Email: peter.dawes@vodafone.com
Full Copyright Statement

Copyright (C) The IETF Trust (2008).

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, THE IETF TRUST 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.