

Network Working Group  
Internet-Draft  
Expires: August 16, 2005

R. Sparks  
Xten  
February 15, 2005

**Multiple Dialog Usages in the Session Initiation Protocol  
draft-sparks-sipping-dialogusage-01**

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of [section 3 of RFC 3667](#). 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 become aware will be disclosed, in accordance with [RFC 3668](#).

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 August 16, 2005.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

Several methods in the Session Initiation Protocol can create an association between endpoints known as a dialog. Some of these methods can also create a new association within an existing dialog. These multiple associations, or dialog usages, require carefully coordinated processing as they have independent life-cycles, but share common dialog state.

This memo argues that multiple dialog usages should be avoided. It



discusses alternatives to their use and clarifies essential behavior for elements that cannot currently avoid them.

This is an informative document and makes no normative statements of any kind.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">2.</a>	Examples of Multiple Usages . . . . .	<a href="#">3</a>
<a href="#">2.1</a>	Transfer . . . . .	<a href="#">3</a>
<a href="#">2.2</a>	Reciprocal Subscription . . . . .	<a href="#">5</a>
<a href="#">3.</a>	Proper Handling of Multiple Usages . . . . .	<a href="#">7</a>
3.1	A survey of the effect of failure responses on usages and dialogs . . . . .	<a href="#">8</a>
<a href="#">3.2</a>	Transaction timeouts . . . . .	<a href="#">13</a>
<a href="#">3.3</a>	Matching requests to usages . . . . .	<a href="#">14</a>
<a href="#">3.4</a>	Target refresh requests . . . . .	<a href="#">15</a>
<a href="#">3.5</a>	Refreshing and Terminating Usages . . . . .	<a href="#">15</a>
<a href="#">3.6</a>	Refusing new usages . . . . .	<a href="#">15</a>
<a href="#">3.7</a>	Replacing usages . . . . .	<a href="#">16</a>
<a href="#">4.</a>	Avoiding Multiple Usages . . . . .	<a href="#">16</a>
<a href="#">5.</a>	Conclusion . . . . .	<a href="#">21</a>
<a href="#">6.</a>	Acknowledgments . . . . .	<a href="#">21</a>
<a href="#">7.</a>	Informative References . . . . .	<a href="#">21</a>
	Author's Address . . . . .	<a href="#">22</a>
<a href="#">A.</a>	Change Log . . . . .	<a href="#">22</a>
<a href="#">A.1</a>	00->01 . . . . .	<a href="#">22</a>
	Intellectual Property and Copyright Statements . . . . .	<a href="#">24</a>



## **1. Introduction**

Several methods in SIP can establish a dialog. When they do so, they also establish an association between the endpoints within that dialog. This association has been known for some time as a "dialog usage" in the developer community. A dialog initiated with an INVITE request has an invite usage. A dialog initiated with a SUBSCRIBE request has a subscribe usage.

Dialogs with multiple usages arise from actions like a REFER or SUBSCRIBE issued inside a dialog established with an INVITE request. Multiple REFERS within a dialog create multiple subscriptions, each of which is a new dialog usage sharing common dialog state. This state includes:

- o the Call-ID
- o the local Tag
- o the remote Tag
- o the local CSeq
- o the remote CSeq
- o the Route-set
- o the local contact
- o the remote target

A dialog comes into existence with the creation of the first usage, and continues to exist until the last usage is terminated (reference counting). Unfortunately, many of the usage management aspects of SIP, such as authentication, were originally designed with the implicit assumption that there was one usage per dialog. The resulting mechanisms have mixed effects, some influencing the usage, and some influencing the entire dialog.

The current specifications define two usages, invite and subscribe. A dialog can share up to one invite usage and arbitrarily many subscribe usages. The pseudo-dialog behavior of REGISTER could be considered a third usage. Fortunately, no existing implementations have attempted to mix a registration usage with any other usage.

## **2. Examples of Multiple Usages**

### **2.1 Transfer**

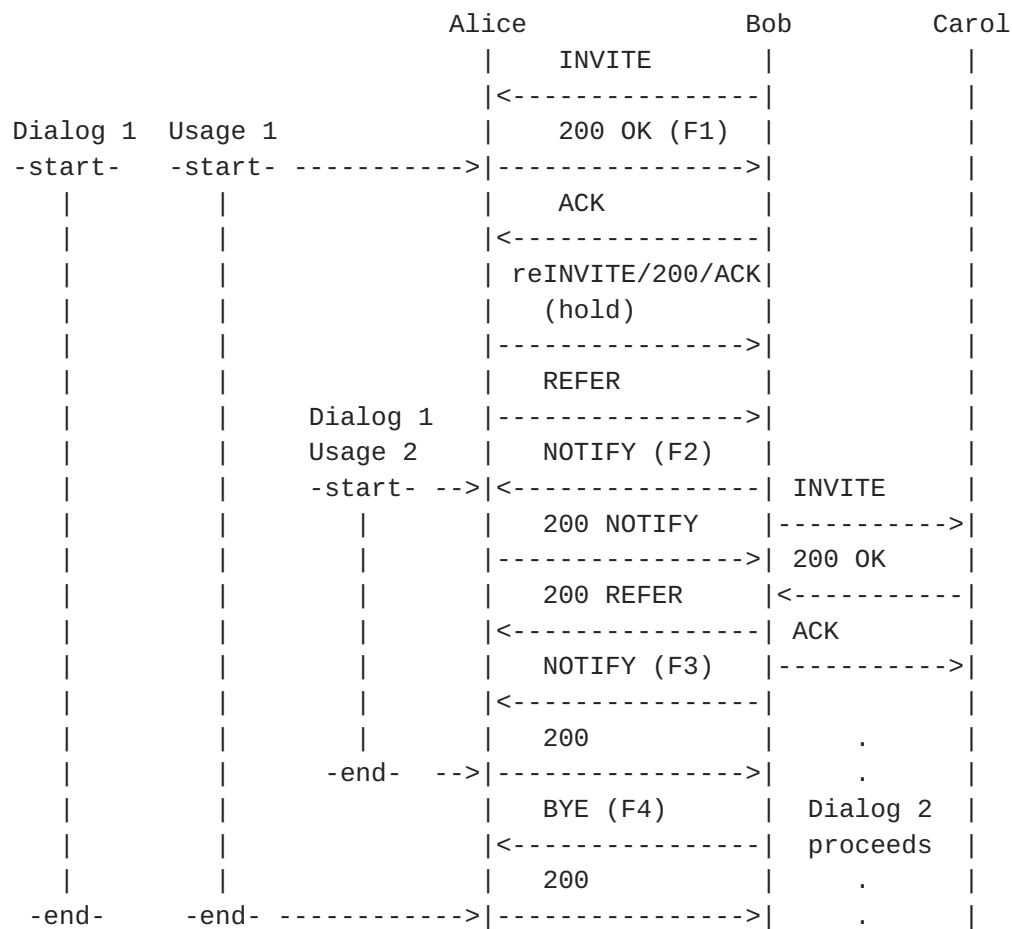
In Figure 1, Alice transfers a call she received from Bob to Carol. A dialog (and an invite dialog usage) between Alice and Bob came into being with the 200 OK labeled F1. A second usage (a subscription to event refer) springs into being with the NOTIFY labeled F2. This second usage ends when the subscription is terminated by the NOTIFY transaction labeled F3. The dialog still has one usage (the invite usage), which lasts until the BYE transaction labeled F4. At this

Sparks

Expires August 16, 2005

[Page 3]

point, the dialog has no remaining usages, so it ceases to exist.



Message Details (abridged to show only dialog or usage details)

F1

```
SIP/2.0 200 OK
Call-ID: dialog1@bob.example.com
CSeq: 100 INVITE
To: <sip:Alice@alice.example.com>;tag=alicetag1
From: <sip:Bob@bob.example.com>;tag=bobtag1
Contact: <sip:aliceinstance@alice.example.com>
```

F2

```
NOTIFY sip:aliceinstance@alice.example.com SIP/2.0
Event: refer
Call-ID: dialog1@bob.example.com
CSeq: 101 NOTIFY
To: <sip:Alice@alice.example.com>;tag=alicetag1
From: <sip:Bob@bob.example.com>;tag=bobtag1
Contact: <sip:bobinstance@bob.example.com>
```

F3

Sparks

Expires August 16, 2005

[Page 4]



```
NOTIFY sip:aliceinstance@alice.example.com SIP/2.0
Event: refer
Subscription-State: terminated;reason=noresource
Call-ID: dialog1@bob.example.com
CSeq: 102 NOTIFY
To: <sip:Alice@alice.example.com>;tag=alicetag1
From: <sip:Bob@bob.example.com>;tag=bobtag1
Contact: <sip:bobinstance@bob.example.com>
Content-Type: message/sipfrag

SIP/2.0 200 OK
```

F4

```
BYE sip:aliceinstance@alice.example.com SIP/2.0
Call-ID: dialog1@bob.example.com
CSeq: 103 BYE
To: <sip:Alice@alice.example.com>;tag=alicetag1
From: <sip:Bob@bob.example.com>;tag=bobtag1
Contact: <sip:bobinstance@bob.example.com>
```

Figure 1

## **2.2 Reciprocal Subscription**

In Figure 2, Alice subscribes to Bob's presence. For simplicity, assume Bob and Alice are both serving their presence from their endpoints instead of a presence server. For space, the figure leaves out any rendezvous signaling through which Alice discovers Bob's endpoint.

Bob is interested in Alice's presence too, so he subscribes to Alice (in most deployed presence/IM systems, people watch each other). He decides skip the rendezvous step since he's already in a dialog with Alice, and sends his SUBSCRIBE inside that dialog (a few early SIMPLE clients behaved exactly this way).

The dialog and its first usage comes into being at F1, which establishes Alice's subscription to Bob. Its second usage begins at F2, which establishes Bob's subscription to Alice. These two subscriptions are independent - they have distinct and different expirations, but they share all the dialog state.

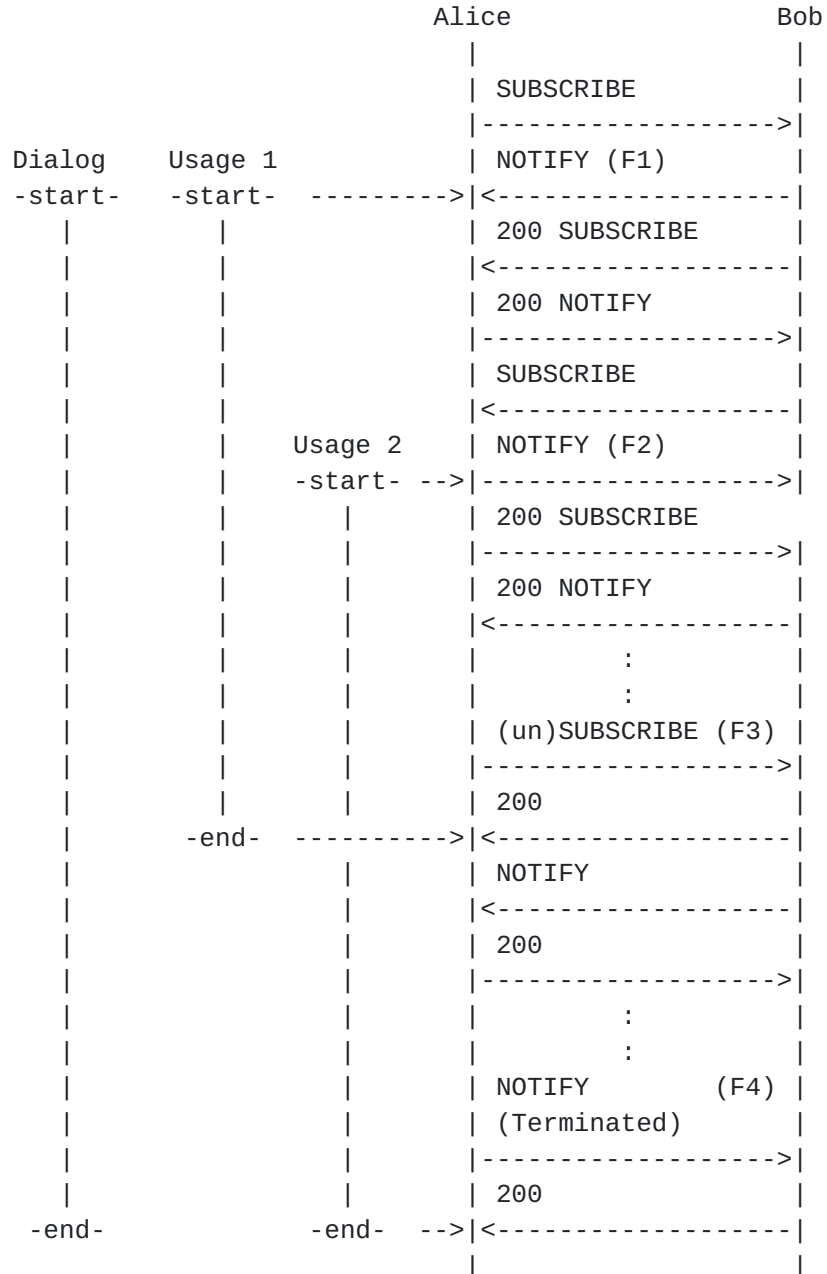
The first usage ends when Alice decides to unsubscribe at F3. Bob's

Sparks

Expires August 16, 2005

[Page 5]

subscription to Alice, and thus the dialog, continues to exist. Alice's UA must maintain this dialog state even though the subscription that caused it to exist in the first place is now over. The second usage ends when Alice decides to terminate Bob's subscription at F4 (she's probably going to reject any attempt on Bob's part to resubscribe until she's ready to subscribe to Bob again). Since this was the last usage, the dialog also terminates.



Message Details (abridged to show only dialog or usage details)

F1

NOTIFY sip:aliceinstance@alice.example.com SIP/2.0

Sparks

Expires August 16, 2005

[Page 6]

```
Event: presence
Subscription-State: active;expires=600
Call-ID: alicecallid1@alice.example.com
From: <sip:Bob@bob.example.com>;tag=bobtag2
To: <sip:Alice@alice.example.com>;tag=alicetag2
CSeq: 100 NOTIFY
Contact: <sip:bobinstance@bob.example.com>

F2
NOTIFY sip:bobinstance@bob.example.com SIP/2.0
Event: presence
Subscription-State: active;expires=1200
Call-ID: alicecallid1@alice.example.com
To: <sip:Bob@bob.example.com>;tag=bobtag2
From: <sip:Alice@alice.example.com>;tag=alicetag2
CSeq: 500 NOTIFY
Contact: <sip:aliceinstance@alice.example.com>

F3
SUBSCRIBE sip:bobinstance@bob.example.com SIP/2.0
Event: presence
Expires: 0
Call-ID: alicecallid1@alice.example.com
To: <sip:Bob@bob.example.com>;tag=bobtag2
From: <sip:Alice@alice.example.com>;tag=alicetag2
CSeq: 501 SUBSCRIBE
Contact: <sip:aliceinstance@alice.example.com>

F4
NOTIFY sip:bobinstance@bob.example.com SIP/2.0
Event: presence
Subscription-State: terminated;reason=deactivated
Call-ID: alicecallid1@alice.example.com
To: <sip:Bob@bob.example.com>;tag=bobtag2
From: <sip:Alice@alice.example.com>;tag=alicetag2
CSeq: 502 NOTIFY
Contact: <sip:aliceinstance@alice.example.com>
```

Figure 2

### **3. Proper Handling of Multiple Usages**

The examples in [Section 2](#) show straightforward cases where it is fairly obvious when the dialog begins and ends. Unfortunately, there are many scenarios where such clarity is not present. For instance,

Sparks

Expires August 16, 2005

[Page 7]

in Figure 1, what would it mean if the response to the NOTIFY (F2) were a 481? Does that simply terminate the refer subscription, or does it destroy the entire dialog? This section explores the problem spots with multiple usages that have been identified to date.

### **3.1 A survey of the effect of failure responses on usages and dialogs**

For this survey, consider a subscribe usage inside a dialog established with an invite usage. Unless stated otherwise, we'll discuss the effect on each usage and the dialog when a client issuing a NOTIFY inside the subscribe usage receives a failure response (such as a transferee issuing a NOTIFY to event refer).

This survey is written from the perspective of a client receiving the error response. The effect on dialogs and usages at the server issuing the response is the same.

3xx responses: Redirection mid-dialog is not well understood in SIP, but whatever effect it has impacts the entire dialog and all of its usages equally. In our example scenario, both the subscription and the invite usage would be redirected by this single response.

400 and unrecognized 4xx responses: These responses affect only the NOTIFY transaction, not the subscription, the dialog it resides in (beyond affecting the local CSeq), or any other usage of that dialog. In general, the response is a complaint about this transaction, not the usage or dialog the transaction occurs in.

401 Unauthorized ,407 Proxy Authentication Required: This request, not the subscription or dialog, is being challenged. The usages and dialog are not terminated.

402 Payment Required: This is a reserved response code. If encountered, it should be treated as an unrecognized 4xx.

403 Forbidden: This response terminates the subscription, but has no effect on any other usages of the dialog. In our example scenario, the invite usage continues to exist. Similarly, if the 403 came in response to a reINVITE, the invite usage would be terminated, but not the subscription.

404 Not Found: This response destroys the dialog and all usages sharing it. The Request-URI that is being 404ed is the remote target set by the Contact provided by the peer. Getting this response means something has gone fundamentally wrong with the dialog state.





405 Method Not Allowed: In our example scenario, this response destroys the subscription, but not the invite usage or the dialog. It's an aberrant case for NOTIFYs to receive a 405 since they only come as a result to something that creates subscription. In general, a 405 within a given usage affects only that usage, but does not affect other usages of the dialog.

406 Not Acceptable: These responses concern details of the message in the transaction. Subsequent requests in this same usage may succeed. Neither the usage nor dialog is terminated, other usages sharing this dialog are unaffected.

408 Request Timeout: Receiving a 408 will have the same effect on usages and dialogs as a real transaction timeout as described in [Section 3.2](#).

410 Gone: This response destroys the dialog and all usages sharing it. The Request-URI that is being rejected is the remote target set by the Contact provided by the peer. Similar to 404, getting this response means something has gone fundamentally wrong with the dialog state, its slightly less aberrant in that the other endpoint recognizes that this was once a valid URI that it isn't willing to respond to anymore.

412 Conditional Request Failed:

413 Request Entity Too Large:

414 Request-URI Too Long:

415 Unsupported Media Type: These responses concern details of the message in the transaction. Subsequent requests in this same usage may succeed. Neither the usage nor dialog is terminated, other usages sharing this dialog are unaffected.

416 Unsupported URI Scheme: Similar to 404 and 410, this response came to a request whose Request-URI was provided by the peer in a Contact header field. Something has gone fundamentally wrong, and the dialog and all of its usages are destroyed.

420 Bad Extension, 421 Extension Required: These responses are objecting to the request, not the usage. The usage is not affected. The dialog is only affected by a change in its local CSeq. No other usages of the dialog are affected.

423 Interval Too Brief: This response won't happen in our example scenario, but if it came in response to a reSUBSCRIBE, the subscribe usage is not destroyed (or otherwise affected). No other usages of the dialog are affected.



429 Provide Referrer Identity: This response won't be returned to a NOTIFY as in our example scenario, but when it is returned to a REFER, it is objecting to the REFER request itself, not any usage the REFER occurs within. The usage is unaffected. Any other usages sharing this dialog are unaffected. The dialog is only affected by a change in its local CSeq.

480 Temporarily Unavailable: [RFC 3261](#) is unclear on what this response means for mid-usage requests. Clarifications will be made to show that this response affects only the usage in which the request occurs. No other usages are affected. If the response included a Retry-After header field, further requests in that usage should not be sent until the indicated time has past. Requests in other usages may still be sent at any time.

481 Call/Transaction Does Not Exist: This response indicates that the peer has lost its copy of the dialog state. The dialog and any usages sharing it are destroyed.

OPEN ISSUE: There has been some list discussion indicating a need for a response code that only terminates the usage, but not the dialog (Motivation: being able to destroy a subscription by refusing a NOTIFY without destroying the dialog that that subscription exists in, particularly when the subscription exists because of an in-dialog REFER).

482 Loop Detected: This response is aberrant mid-dialog. It will only occur if the Record-Route header field was improperly constructed by the proxies involved in setting up the dialog's initial usage, or if a mid-dialog request forks and merges (which should never happen). Future requests using this dialog state will also fail. The dialog and any usages sharing it are destroyed.

483 Too Many Hops: Similar to 482, receiving this mid-dialog is aberrant. Unlike 482, recovery may be possible by increasing Max-Forwards (assuming that the requester did something strange like using a smaller value for Max-Forwards in mid-dialog requests than it used for an initial request). If the request isn't tried with an increased Max-Forwards, then the agent should attempt to gracefully terminate this usage and all other usages that share its dialog.

484 Address Incomplete, 485 Ambiguous: Similar to 404 and 410, these responses came to a request whose Request-URI was provided by the peer in a Contact header field. Something has gone fundamentally wrong, and the dialog and all of its usages are destroyed.

Sparks

Expires August 16, 2005

[Page 10]

486 Busy Here: This response is non-sensical in our example scenario, or in any scenario where this response comes inside an established usage. If it occurs in that context, it should be treated as an unknown 4xx response. The usage, and any other usages sharing its dialog are unaffected. The dialog is only affected by the change in its local CSeq. If this response is to a request that is attempting to establish a new usage within an existing dialog (such as an INVITE sent within a dialog established by a subscription), the request fails, no new usage is created, and no other usages are affected.

487 Request Terminated: This response speaks to the disposition of a particular request (transaction). The usage in which that request occurs is not affected by this response (it may be affected by another associated request within that usage). No other usages sharing this dialog are affected.

488 Not Acceptable Here: This response is objecting to the request, not the usage. The usage is not affected. The dialog is only affected by a change in its local CSeq. No other usages of the dialog are affected.

489 Bad Event: In our example scenario, [3] declares that the subscription usage in which the NOTIFY is sent is terminated. The invite usage is unaffected and the dialog continues to exist. This response is only valid in the context of SUBSCRIBE and NOTIFY. UAC behavior for receiving this response to other methods is not specified, but treating it as an unknown 4xx is a reasonable practice.

491 Request Pending: This response addresses in-dialog request glare. Its affect is scoped to the request. The usage in which the request occurs is not affected. The dialog is only affected by the change in its local CSeq. No other usages sharing this dialog are affected.

493 Undecipherable: This response objects to the request, not the usage. The usage is not affected. The dialog is only affected by a change in its local CSeq. No other usages of the dialog are affected.

494 Security Agreement Required: This response is objecting to the request, not the usage. The usage is not affected. The dialog is only affected by a change in its local CSeq. No other usages of the dialog are affected.

Sparks

Expires August 16, 2005

[Page 11]

500 and 5xx unrecognized responses: These responses are complaints against the request (transaction), not the usage. If the response contains a Retry-After header field value, the server thinks the condition is temporary and the request can be retried after the indicated interval. This usage, and any other usages sharing the dialog are unaffected. If the response does not contain a Retry-After header field value, the UA may decide to retry after an interval of its choosing or attempt to gracefully terminate the usage. Whether or not to terminate other usages depends on the application. If the UA receives a 500 (or unrecognized 5xx) in response to an attempt to gracefully terminate this usage, it can treat this usage as terminated. If this is the last usage sharing the dialog, the dialog is also terminated.

501 Not Implemented: This would be a degenerate response in our example scenario since the NOTIFY is being sent as part of an established subscribe usage. In this case, the UA knows the condition is unrecoverable and should stop attempting to send NOTIFYs on this usage. (It may or may not destroy the usage. If it remembers the bad behavior, it can reject any refresh subscription). In general, this response may or may not affect the usage (a 501 to an unknown method or an INFO will not end an invite usage). It will never affect other usages sharing this usage's dialog.

502 Bad Gateway: This response is aberrant mid-dialog. It will only occur if the Record-Route header field was improperly constructed by the proxies involved in setting up the dialog's initial usage. Future requests using this dialog state will also fail. The dialog and any usages sharing it are destroyed.

503 Service Unavailable: As per [2], the logic handling locating SIP servers for transactions may handle 503 requests (effectively sequentially forking at the endpoint based on DNS results). If this process does not yield a better response, a 503 may be returned to the transaction user. Like a 500 response, the error is a complaint about this transaction, not the usage. Because this response occurred in the context of an established usage (hence an existing dialog), the route-set has already been formed and any opportunity to try alternate servers (as recommended in [1] has been exhausted by the RFC3263 logic. The response should be handled as described for 500 earlier in this memo.

504 Server Time-out: It is not obvious under what circumstances this response would be returned to a request in an existing dialog. If it occurs it should have the same affect on the dialog and its usages as described for unknown 5xx responses.

Sparks

Expires August 16, 2005

[Page 12]



505 Version Not Supported, 513 Message Too Large: These responses are objecting to the request, not the usage. The usage is not affected. The dialog is only affected by a change in its local CSeq. No other usages of the dialog are affected.

580 Precondition Failure: This response is objecting to the request, not the usage. The usage is not affected. The dialog is only affected by a change in its local CSeq. No other usages of the dialog are affected.

600 and 6xx unrecognized responses: Unlike 400 Bad Request, a 600 response code says something about the recipient user, not the request that was made. This end user is stating an unwillingness to communicate. If the response contains a Retry-After header field value, the user is indicating willingness to communicate later and the request can be retried after the indicated interval. This usage, and any other usages sharing the dialog are unaffected. If the response does not contain a Retry-After header field value, the UA may decide to retry after an interval of its choosing or attempt to gracefully terminate the usage. Whether or not to terminate other usages depends on the application. If the UA receives a 600 (or unrecognized 6xx) in response to an attempt to gracefully terminate this usage, it can treat this usage as terminated. If this is the last usage sharing the dialog, the dialog is also terminated.

603 Decline: This response declines the action indicated by the associated request. It can be used, for example, to decline a hold or transfer attempt. Receiving this response does NOT terminate the usage it occurs in. Other usages sharing the dialog are unaffected.

604 Does Not Exist Anywhere: Like 404, this response destroys the dialog and all usages sharing it. The Request-URI that is being 604ed is the remote target set by the Contact provided by the peer. Getting this response means something has gone fundamentally wrong with the dialog state.

606 Not Acceptable: This response is objecting to aspects of the associated request, not the usage the request appears in. The usage is unaffected. Any other usages sharing the dialog are unaffected. The only affect on the dialog is the change in the local CSeq.

### **3.2 Transaction timeouts**

[1] states that a UAC should terminate a dialog (by sending a BYE) if



no response is received for a request sent within a dialog. This recommendation should have been limited to the invite usage instead of the whole dialog. [3] states that a timeout for a NOTIFY removes a subscription, but a SUBSCRIBE that fails with anything other than a 481 does not. Given these statements, it is unclear whether a refresh SUBSCRIBE issued in a dialog shared with an invite usage destroys either usage or the dialog if it times out.

Generally, a transaction timeout should affect only the usage in which the transaction occurred. Other uses sharing the dialog should not be affected. In the worst case of timeout due to total transport failure, it may require multiple failed messages to remove all usages from a dialog (at least one per usage).

There are some mid-dialog messages that never belong to any usage. If they timeout, they will have no effect on the dialog or its usages.

### **3.3 Matching requests to usages**

For many mid-dialog requests, identifying the usage they belong to is obvious. A dialog can have at most one invite usage, so any INVITE, UPDATE, PRACK, ACK, CANCEL, BYE, or INFO requests belong to it. The usage (i.e. the particular subscription) SUBSCRIBE, NOTIFY, and REFER requests belong to can be determined from the Event header field of the request. REGISTER requests within a (pseudo)-dialog belong to the registration usage. (As mentioned before, implementations aren't mixing registration usages with other usages, so this document isn't exploring the consequences of that bad behavior).

According to [1], "an OPTIONS request received within a dialog generates a 200 OK response that is identical to one constructed outside a dialog and does not have any impact on that dialog". Thus OPTIONS does not belong to any usage. Only those failures discussed in [Section 3.1](#) and [Section 3.2](#) that destroy entire dialogs will have any effect on the usages sharing the dialog with a failed OPTIONS request.

MESSAGE requests are not currently allowed inside a dialog (though some implementations use it that way, against the standard recommendation). As it is not meant to be part of any given dialog, it cannot be part of any given usage. A failed MESSAGE request should have similar effects on a dialog and its usages as a failed OPTIONS request.

Mid-dialog requests with unknown methods cannot be matched with a usage. Servers will return a failure response (likely a 501). The

Sparks

Expires August 16, 2005

[Page 14]

effect on the dialog and its usages at either the client or the server should be similar to that of a failed OPTIONS request.

### **3.4 Target refresh requests**

Target refresh requests update the remote target of a dialog when they are successfully processed. The currently defined target refresh requests are INVITE, UPDATE, SUBSCRIBE and NOTIFY (clarified in a bug against [RFC3565](#)) and REFER (clarified in a bug against [RFC3515](#)).

The remote target is part of the dialog state. When a target refresh request affects it, it affects it for ALL usages sharing that dialog. If a subscription and invite usage are sharing a dialog, sending a refresh SUBSCRIBE with a different contact will cause reINVITES from the peer to go to that different contact.

A UAS will only update the remote target if it sends a 200 class response to a target refresh request. A UAC will only update the remote target if it receives a 200 class response to a target refresh request. Again, any update to a dialog's remote target affects all usages of that dialog.

### **3.5 Refreshing and Terminating Usages**

Subscription and registration usages expire over time and must be refreshed (with a refresh SUBSCRIBE for example). This expiration is usage state, not dialog state. If several subscriptions share a dialog, refreshing one of them has no effect on the expiration of the others.

Normal termination of a usage has no effect on other usages sharing the same dialog. For instance terminating a subscription with a NOTIFY/Subscription-State: terminated will not terminate an invite usage sharing its dialog. Likewise, ending an invite usage with a BYE does not terminate any active Event: refer subscriptions established on that dialog.

Abnormal termination can effect all usages on a dialog. Rejecting a NOTIFY with a 481 (incorrectly recommended in the past as an inexpensive way to terminate a REFER subscription) destroys the dialog and all of its usages.

### **3.6 Refusing new usages**

As the survey of the effect of failure responses shows, care must be taken when refusing a new usage inside an existing dialog. Choosing the wrong response code will terminate the dialog and all of its

Sparks

Expires August 16, 2005

[Page 15]

usages. Generally, returning a 603 Decline is the safest way to refuse a new usage.

### **3.7 Replacing usages**

[6] defines a mechanism through which one usage can replace another. It can be used, for example, to associate the two dialogs a transfer target is involved in during an attended transfer. It is written using the term "dialog", but its intent was to only affect the invite usage of the dialog it targets. Any other usages inside that dialog are unaffected. For some applications, the other usages may no longer make sense, and the application may terminate them as well.

However, the interactions between Replaces and multiple dialog usages have not been well explored. More discussion of this topic is needed. Implementers should avoid this scenario completely.

## **4. Avoiding Multiple Usages**

Processing multiple usages correctly is not completely understood. What is understood is difficult to implement and is very likely to lead to interoperability problems. The best way to avoid the trouble that comes with such complexity is to avoid it altogether.

When designing new applications that use SIP dialogs, do not construct multiple usages. If a peer attempts to create a second usage inside a dialog, refuse it.

Unfortunately, there are existing applications, like transfer, that currently entail multiple usages, so the simple solution of "don't do it" will require some transitional work. This section looks at the pressures that led to these existing multiple usages and suggests alternatives.

When executing a transfer, the transferor and transferee currently share an invite usage and a subscription usage within the dialog between them. This is a result of sending the REFER request within the dialog established by the invite usage. Implementations were led to this behavior by two primary pressures:

1. There was no way to ensure that a REFER on a new dialog would reach the particular endpoint involved in a transfer. Many factors, including details of implementations and changes in proxy routing between an INVITE and a REFER could cause the REFER to be sent to the wrong place. Sending the REFER down the existing dialog ensured it got to the endpoint we were already talking to.
2. It was unclear how to associate an existing invite usage with a REFER arriving on a new dialog, where it was completely obvious

Sparks

Expires August 16, 2005

[Page 16]



what the association was when the REFER came on the invite usage's dialog.

3. There were concerns with authorizing out-of-dialog REFERs. The authorization policy for REFER in most implementations piggybacks on the authorization policy for INVITE (which is, in most cases, based simply on "I placed or answered this call").

GRUUs [7] have been defined specifically to address problem 1. Problem 2 can be addressed using a GRUU's grid parameter. However, this approach requires endpoints to create and maintain a GRUU per dialog, something the working group is not comfortable recommending.

The Join [5] and Replaces [6] mechanisms address problem 1 differently. Here, a new request is sent outside any dialog with the expectation that it will fork to possibly many endpoints, including the one we're interested in. This request contains a header field listing the dialog identifiers of a dialog in progress. Only the endpoint holding a dialog matching those identifiers will accept the request. The other endpoints the request may have forked to will respond with an error. This mechanism is reasonably robust, failing only when the routing logic for out-of-dialog requests changes such that the new request does not arrive at the endpoint holding the dialog of interest.

The reachability aspects of using a GRUU to address problem 1 can be combined with the association-with-other-dialogs aspects of the Join/Replaces solution. A REFER request sent out-of-dialog can be sent towards a GRUU, and identify an existing dialog as part of the context the receiver should use. A new header, Target-Dialog: perhaps, would be included in the REFER listing the dialog this REFER is associated with. Figure 3 sketches how this could be used to achieve transfer without reusing a dialog.

Alice	Bob	Carol
F1 INVITE (Bob's AOR)		
Call-ID: (call-id-one)		
Contact: (Alice's-GRUU)		
----->		
F2 200 OK		
To: <>;tag=totag1		
From: <>;tag=fromtag1		
Call-ID: (call-id one)		
Contact: (Bob's-GRUU)		
<-----		
ACK		
----->		

Sparks

Expires August 16, 2005

[Page 17]

```

:
(Bob places Alice on hold)
:
F3 INVITE (Carol's AOR)
  Call-ID: (call-id two)
  Contact: (Bob's-GRUU)
----->
F4 200 OK
  To: <>;tag=totag2
  From: <>;tag=fromtag2
  Call-ID: (call-id two)
  Contact: (Carol's-GRUU)
<-----
  ACK
----->
:
(Bob places Carol on hold)
:
F5 REFER (Alice's-GRUU)
  Call-ID: (call-id three)
  Refer-To: (Carol's-GRUU)
  Target-Dialog: (call-id one,totag1,fromtag1)
  Contact: (Bob's-GRUU)
<-----
  202 Accepted
----->
  NOTIFY (Bob's-GRUU)
  Call-ID: (call-id three)
----->
  200 OK
<-----
:
F6 INVITE (Carol's-GRUU)
  Call-ID: (call-id-four)
  Contact: (Alice's-GRUU)
----->
  200 OK
  Contact: (Carol's-GRUU)
<-----
  ACK
----->
F7 NOTIFY (Bob's-GRUU)
  Call-ID: (call-id three)
----->
  200 OK
<-----
  BYE (Alice's-GRUU)
  Call-ID: (call-id one)
<-----
  BYE (Carol's-GRUU)

```

Sparks

Expires August 16, 2005

[Page 18]

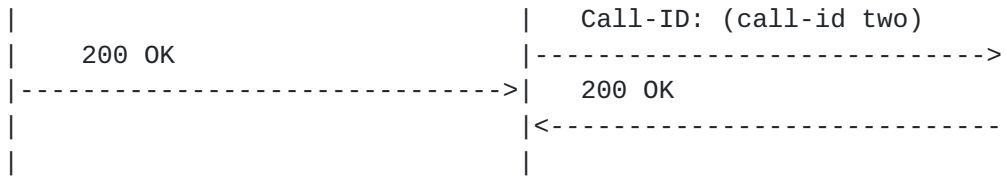


Figure 3: Transfer without dialog reuse

In message F1, Alice invites Bob indicating support for GRUUs (and offering a GRUU for herself):

Message F1 (abridged, detailing pertinent fields)

```

INVITE sip:bob@example.com SIP/2.0
Call-ID: 13jfdwer230jsdw@alice.example.com
Supported: gruu
Contact: <sip:aanewmr203raswdf@example.com>
  
```

Message F2 lets Alice know that Bob understands GRUUs. If Bob did not indicate this support, the original multi-usage approach to transfer would have to be used.

Message F2 (abridged, detailing pertinent fields)

```

SIP/2.0 200 OK
Supported: gruu
To: <sip:bob@example.com>;tag=totag1
From: <sip:alice@example.com>;tag=fromtag1
Contact: <sip:boaidfjjereis@example.com>
  
```

Bob decides to try to transfer Alice to Carol, so he puts Alice on hold and sends an INVITE to Carol. Carol and Bob negotiate GRUU support similar to what happened in F1 and F2.



Message F3 (abridged, detailing pertinent fields)

```
INVITE sip:carol@example.com SIP/2.0
Supported: gruu
Call-ID: 23rasdnfoa39i4jnasdf@bob.example.com
Contact: <sip:boaiidfjjereis@example.com>
```

Message F4 (abridged, detailing pertinent fields)

```
SIP/2.0 200 OK
Supported: gruu
To: <sip:carol@example.com>;tag=totag2
From: <sip:bob@example.com>;tag=fromtag2
Call-ID: 23rasdnfoa39i4jnasdf@bob.example.com
Contact: <sip:c239fniuweorw9sdfn@example.com>
```

After consulting Carol, Bob places her on hold and refers Alice to her using message F5. Notice that the Refer-To URI is Carol's GRUU, and that this is on a different Call-ID than message F1. (The URI in the Refer-To header is line-broken for readability in this draft, it would not be valid to break the URI this way in a real message)

Message F5 (abridged, detailing pertinent fields)

```
REFER sip:aanewmr203raswdf@example.com SIP/2.0
Call-ID: 39fa99r0329493asdsf3n@bob.example.com
Refer-To: <sip:c239fniuweorw9sdfn@example.com
          ?Replaces=23rasdnfoa39i4jnasdf@bob.example.com;
          to-tag=totag2;from-tag=fromtag2>
Target-Dialog: 13jfdwer230jsdw@alice.example.com;
               local-tag=fromtag1;remote-tag=totag1
Supported: gruu
Contact: <sip:boaiidfjjereis@example.com>
```

Alice uses the information in the Target-Dialog header field to determine that this REFER is associated with the dialog she already has in place with Bob. Alice is now in a position to use the same admission policy she used for in-dialog REFERS: "Do I have a call with this person?". She accepts the REFER. sends Bob the obligatory immediate NOTIFY, and proceeds to INVITE Carol with message F6.

Message F6 (abridged, detailing pertinent fields)

```
INVITE sip:c239fniuweorw9sdfn@example.com SIP/2.0
Call-ID: 4zsd9f234jasdfasn3jsad@alice.example.com
Replaces: 23rasdnfoa39i4jnasdf@bob.example.com;
          to-tag=totag2;from-tag=fromtag2
```





Supported: gruu  
Contact: <sip:aanewmr203raswdf@example.com>

Carol accepts Alice's invitation to replace her dialog (invite usage) with Bob and notifies him that the REFERenced INVITE succeeded with F7:

Message F7 (abridged, detailing pertinent fields)

NOTIFY sip:boaidfjjereis@example.com SIP/2.0  
Subscription-State: terminated;reason=noresource  
Call-ID: 39fa99r0329493asdsf3n@bob.example.com  
Contact: <sip:aanewmr203raswdf@example.com>  
Content-Type: message/sipfrag

SIP/2.0 200 OK

Bob then ends his invite usages with both Alice and Carol using BYEs.

## 5. Conclusion

Handling multiple usages within a single dialog is complex and introduces scenarios where the right thing to do is not clear. Implementations should avoid entering into multiple usages whenever possible. New applications should be designed to never introduce multiple usages.

There are some accepted SIP practices, including transfer, that currently require multiple usages. Recent work, most notably GRUU, makes those practices unnecessary. The standardization of those practices and the implementations should be revised as soon as possible to use only single-usage dialogs.

## 6. Acknowledgments

The ideas in this draft have been refined over several IETF meetings with many participants. Significant contribution was provided by Adam Roach, Alan Johnston, Ben Campbell, Cullen Jennings, Jonathan Rosenberg and Rohan Mahy. Members of the reSIProcate project (<http://www.sipfoundry.org/reSIProcate>) also shared their difficulties and discoveries while implementing multiple-usage dialog handlers.

## 7 Informative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.,

Sparks

Expires August 16, 2005

[Page 21]

- Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", [RFC 3261](#), June 2002.
- [2] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", [RFC 3263](#), June 2002.
- [3] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", [RFC 3265](#), June 2002.
- [4] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", [RFC 3515](#), April 2003.
- [5] Mahy, R. and D. Petrie, "The Session Initiation Protocol (SIP) "Join" Header", [RFC 3911](#), October 2004.
- [6] Mahy, R., Biggs, B. and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", [RFC 3891](#), September 2004.
- [7] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", [draft-ietf-sip-gruu-02](#) (work in progress), July 2004.

#### Author's Address

Robert J. Sparks  
Xten  
5100 Tennyson Parkway  
Suite 1000  
Plano, TX 75024

E-Mail: [RjS@xten.com](mailto:RjS@xten.com)

#### [Appendix A.](#) Change Log

##### [A.1](#) 00->01

- o Changed 480 to affect only the usage the response occurred in.
- o Closed the open issue on 482. Usages and dialogs are destroyed even though there is an edge condition in which the response is only stimulated by certain methods (due to method specific routing rules).
- o Closed the open issue on 483. Usages are not terminated since the request might succeed if retried with a greater initial Max-Forwards
- o Closed the open issue on 502, accepting -00s suggestion that the same reasoning used for 482 applies.
- o Redid the transfer example to not require a GRUU per usage, but instead leverage the target-dialog concepts common to Join and

Sparks

Expires August 16, 2005

[Page 22]

Replaces.

## Intellectual Property Statement

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

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

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

## Disclaimer of Validity

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

## Copyright Statement

Copyright (C) The Internet Society (2005). 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.

## Acknowledgment

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

Sparks

Expires August 16, 2005

[Page 24]