

Network Working Group
Request for Comments: 4485
Category: Informational

J. Rosenberg
Cisco Systems
H. Schulzrinne
Columbia University
May 2006

Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The Session Initiation Protocol (SIP) is a flexible yet simple tool for establishing interactive communications sessions across the Internet. Part of this flexibility is the ease with which it can be extended. In order to facilitate effective and interoperable extensions to SIP, some guidelines need to be followed when developing SIP extensions. This document outlines a set of such guidelines for authors of SIP extensions.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Should I Define a SIP Extension?	3
3.1. SIP's Solution Space	4
3.2. SIP Architectural Model	5
4. Issues to Be Addressed	7
4.1. Backwards Compatibility	7
4.2. Security	10
4.3. Terminology	10
4.4. Syntactic Issues	10
4.5. Semantics, Semantics, Semantics	13
4.6. Examples Section	14
4.7. Overview Section	14
4.8. IANA Considerations Section	14
4.9. Document-Naming Conventions	16
4.10. Additional Considerations for New Methods	16
4.11. Additional Considerations for New Header Fields or Header Field	17
4.12. Additional Considerations for New Body Types	18
5. Interactions with SIP Features	18
6. Security Considerations	19
7. Acknowledgements	19
8. References	19
8.1. Normative References	19
8.2. Informative References	20

1. Introduction

The Session Initiation Protocol (SIP) [2] is a flexible yet simple tool for establishing interactive communications sessions across the Internet. Part of this flexibility is the ease with which it can be extended (with new methods, new header fields, new body types, and new parameters), and there have been countless proposals that have been made to do just that. An IETF process has been put into place that defines how extensions are to be made to the SIP protocol [10]. That process is designed to ensure that extensions are made that are appropriate for SIP (as opposed to being done in some other protocol), that these extensions fit within the model and framework provided by SIP and are consistent with its operation, and that these extensions solve problems generically rather than for a specific use case. However, [10] does not provide the technical guidelines needed to assist that process. This specification helps to meet that need.

This specification first provides a set of guidelines to help decide whether a certain piece of functionality is appropriately done in SIP. Assuming the functionality is appropriate, it then points out

issues that extensions should deal with from within their specification. Finally, it discusses common interactions with existing SIP features that often cause difficulties in extensions.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant implementations.

3. Should I Define a SIP Extension?

The first question to be addressed when defining a SIP extension is whether a SIP extension is the best solution to the problem. SIP has been proposed as a solution for numerous problems, including mobility, configuration and management, QoS control, call control, caller preferences, device control, third-party call control, and MPLS path setup, to name a few. Clearly, not every problem can be solved by a SIP extension. More importantly, some problems that could be solved by a SIP extension probably shouldn't.

To assist engineers in determining whether a SIP extension is an appropriate solution to their problem, we present two broad criteria. First, the problem SHOULD fit into the general purview of SIP's solution space. Secondly, the solution MUST conform to the general SIP architectural model.

Although the first criteria might seem obvious, we have observed that numerous extensions to SIP have been proposed because some function is needed in a device that also speaks SIP. The argument is generally given that "I'd rather implement one protocol than many". As an example, user agents, like all other IP hosts, need some way to obtain their IP address. This is generally done through DHCP [11]. SIP's multicast registration mechanisms might supply an alternate way to obtain an IP address. This would eliminate the need for DHCP in clients. However, we do not believe such extensions are appropriate. We believe that protocols should be defined to provide specific, narrow functions, rather than be defined for all protocols needed between a pair of devices. The former approach to protocol design yields modular protocols with broad application. It also facilitates extensibility and growth; single protocols can be removed and changed without affecting the entire system. We observe that this approach to protocol engineering mirrors object-oriented software engineering.

Our second criteria, that the extension must conform to the general SIP architectural model, ensures that the protocol remains manageable and broadly applicable.

3.1. SIP's Solution Space

In order to evaluate the first criteria, it is necessary to define exactly what SIP's solution space is, and what it is not.

SIP is a protocol for initiating, modifying, and terminating interactive sessions. This process involves the discovery of users, (or, more generally, entities that can be communicated with, including services, such as voicemail or translation devices) wherever they may be located, so that a description of the session can be delivered to the user. It is assumed that these users or communications entities are mobile, and that their point of attachment to the network changes over time. The primary purpose of SIP is a rendezvous function, to allow a request initiator to deliver a message to a recipient wherever they may be. Such a rendezvous is needed to establish a session, but it can be used for other purposes related to communications, such as querying for capabilities or delivery of an instant message.

Much of SIP focuses on this discovery and rendezvous component. Its ability to fork, its registration capabilities, and its routing capabilities are all present for the singular purpose of finding the desired user wherever they may be. As such, features and capabilities such as personal mobility, automatic call distribution, and follow-me are well within the SIP solution space.

Session initiation also depends on the ability of the called party to have enough information about the session itself to make a decision on whether to join. That information includes data about the caller, the purpose for the invitation, and parameters of the session itself. For this reason, SIP includes this kind of information.

Part of the process of session initiation is the communication of progress and the final results of establishment of the session. SIP provides this information as well.

SIP itself is independent of the session, and the session description is delivered as an opaque body within SIP messages. Keeping SIP independent of the sessions it initiates and terminates is fundamental. As such, there are many functions that SIP explicitly does not provide. It is not a session management protocol or a conference control protocol. The particulars of the communications within the session are outside of SIP. This includes features such as media transport, voting and polling, virtual microphone passing, chairman election, floor control, and feedback on session quality.

SIP is not a resource reservation protocol for sessions. This is fundamentally because (1) SIP is independent of the underlying

session it establishes, and (2) the path of SIP messages is completely independent from the path that session packets may take. The path independence refers to paths within a provider's network and the set of providers itself. For example, it is perfectly reasonable for a SIP message to traverse a completely different set of autonomous systems than the audio in a session SIP establishes.

SIP is not a general purpose transfer protocol. It is not meant to send large amounts of data unrelated to SIP's operation. It is not meant as a replacement for HTTP. This is not to say that carrying payloads in SIP messages is never a good thing; in many cases, the data is very much related to SIP's operation. In those cases, congestion-controlled transports end-to-end are critical.

SIP is not meant to be a general Remote Procedure Call (RPC) mechanism. None of its user discovery and registration capabilities are needed for RPC, and neither are most of its proxy functions.

SIP is not meant to be used as a strict Public Switched Telephone Network (PSTN) signaling replacement. It is not a superset of the Integrated Services Digital Network (ISDN) User Part (ISUP). Although it can support gatewaying of PSTN signaling and can provide many features present in the PSTN, the mere existence of a feature or capability in the PSTN is not a justification for its inclusion in SIP. Extensions needed to support telephony MUST meet the other criteria described here.

SIP is a poor control protocol. It is not meant to be used for one entity to tell another to pick up or answer a phone, to send audio using a particular codec, or to provide a new value for a configuration parameter. Control protocols have different trust relationships from that assumed in SIP and are more centralized in architecture than SIP is, as SIP is a very distributed protocol.

There are many network layer services needed to make SIP function. These include quality of service, mobility, and security, among others. Rather than build these capabilities into SIP itself, they SHOULD be developed outside of SIP and then used by it. Specifically, any protocol mechanisms that are needed by SIP, but that are also needed by many other application layer protocols SHOULD NOT be addressed within SIP.

3.2. SIP Architectural Model

We describe here some of the primary architectural assumptions that underlie SIP. Extensions that violate these assumptions should be examined more carefully to determine their appropriateness for SIP.

Session independence: SIP is independent of the session it establishes. This includes the type of session, be it audio, video, game, chat session, or virtual reality. SIP operation SHOULD NOT depend on some characteristic of the session. SIP is not specific to voice only. Any extensions to SIP MUST consider the application of SIP to a variety of different session types.

SIP and Session path independence: We have already touched on this once, but it is worth noting again. The set of routers, networks, and/or autonomous systems traversed by SIP messages are unrelated to the set of routers, networks, and/or autonomous systems traversed by session packets. They may be the same in some cases, but it is fundamental to SIP's architecture that they need not be the same. Standards-track extensions MUST NOT be defined that work only when the signaling and session paths are coupled. Non-standard P-header extensions [10] are required for any extension that only works in such a case.

Multi-provider and multi-hop: SIP assumes that its messages will traverse the Internet. That is, SIP works through multiple networks administered by different providers. It is also assumed that SIP messages traverse many hops (where each hop is a proxy). Extensions MUST NOT work only under the assumption of a single hop or specialized network topology. They SHOULD avoid the assumption of a single SIP provider (but see the use of P-Headers, per RFC 3427 [10]).

Transactional: SIP is a request/response protocol, possibly enhanced with intermediate responses. Many of the rules of operation in SIP are based on general processing of requests and responses. This includes the reliability mechanisms, routing mechanisms, and state maintenance rules. Extensions SHOULD NOT add messages that are not within the request-response model.

Proxies can ignore bodies: In order for proxies to scale well, they must be able to operate with minimal message processing. SIP has been engineered so that proxies can always ignore bodies. Extensions SHOULD NOT require proxies to examine bodies.

Proxies don't need to understand the method: Processing of requests in proxies does not depend on the method, except for the well-known methods INVITE, ACK, and CANCEL. This allows for extensibility. Extensions MUST NOT define new methods that must be understood by proxies.

INVITE messages carry full state: An initial INVITE message for a session is nearly identical (the exception is the tag) to a re-INVITE message to modify some characteristic of the session. This full state property is fundamental to SIP and is critical for robustness of SIP systems. Extensions SHOULD NOT modify INVITE processing such that data spanning multiple INVITES must be collected in order to perform some feature.

Generality over efficiency: Wherever possible, SIP has favored general-purpose components rather than narrow ones. If some capability is added to support one service but a slightly broader capability can support a larger variety of services (at the cost of complexity or message sizes), the broader capability SHOULD be preferred.

The Request URI is the primary key for forwarding: Forwarding logic at SIP servers depends primarily on the request URI (this is different from request routing in SIP, which uses the Route header fields to pass a request through intermediate proxies). It is fundamental to the operation of SIP that the request URI indicate a resource that, under normal operations, resolves to the desired recipient. Extensions SHOULD NOT modify the semantics of the request URI.

Heterogeneity is the norm: SIP supports heterogeneous devices. It has built-in mechanisms for determining the set of overlapping protocol functionalities. Extensions SHOULD NOT be defined that only function if all devices support the extension.

4. Issues to Be Addressed

Given an extension has met the litmus tests in the previous section, there are several issues that all extensions should take into consideration.

4.1. Backward Compatibility

One of the most important issues to consider is whether the new extension is backward compatible with baseline SIP. This is tightly coupled with how the Require, Proxy-Require, and Supported header fields are used.

If an extension consists of new header fields or header field parameters inserted by a user agent in a request with an existing method, and the request cannot be processed reasonably by a proxy and/or user agent without understanding the header fields or parameters, the extension MUST mandate the usage of the Require and/or Proxy-Require header fields in the request. These extensions

are not backwards compatible with SIP. The result of mandating usage of these header fields means that requests cannot be serviced unless the entities being communicated with also understand the extension. If some entity does not understand the extension, the request will be rejected. The UAC can then handle this in one of two ways. In the first, the request simply fails, and the service cannot be provided. This is basically an interoperability failure. In the second case, the UAC retries the request without the extension. This will preserve interoperability, at the cost of a "dual stack" implementation in a UAC (processing rules for operation with and without the extension). As the number of extensions increases, this leads to an exponential explosion in the sets of processing rules a UAC may need to implement. The result is excessive complexity.

Because of the possibility of interoperability and complexity problems that result from the usage of Require and Proxy-Require, we believe the following guidelines are appropriate:

- o The usage of these header fields in requests for basic SIP services (in particular, session initiation and termination) is NOT RECOMMENDED. The less frequently a particular extension is needed in a request, the more reasonable it is to use these header fields.
- o The Proxy-Require header field SHOULD be avoided at all costs. The failure likelihood in an individual proxy stays constant, but the path failure grows exponentially with the number of hops. On the other hand, the Require header field only mandates that a single entity, the UAS, support the extension. Usage of Proxy-Require is thus considered exponentially worse than usage of the Require header field.
- o If either Require or Proxy-Require are used by an extension, the extension SHOULD discuss how to fall back to baseline SIP operation if the request is rejected with a 420 response.

Extensions that define new methods do not need to use the Require header field. SIP defines mechanisms that allow a UAC to know whether a new method is understood by a UAS. This includes both the OPTIONS request and the 405 (Method Not Allowed) response with the Allow header field. It is fundamental to SIP that proxies need not understand the semantics of a new method in order to process it. If an extension defines a new method that must be understood by proxies in order to be processed, a Proxy-Require header field is needed. As discussed above, these kinds of extensions are frowned upon.

In order to achieve backwards compatibility for extensions that define new methods, the Allow header field is used. There are two

types of new methods - those that are used for established dialogs (initiated by INVITE, for example), and those that are sent as the initial request to a UA. Since INVITE and its response both SHOULD contain an Allow header field, a UA can readily determine whether the new method can be supported within the dialog. For example, once an INVITE dialog is established, a user agent could determine whether the REFER method [12] is supported if it is present in an Allow header field. If it wasn't, the "transfer" button on the UI could be "greyed out" once the call is established.

Another type of extension is that which requires a proxy to insert header fields or header field parameters into a request as it traverses the network, or for the UAS to insert header fields or header field parameters into a response. For some extensions, if the UAC or UAS does not understand these header fields, the message can still be processed correctly. These extensions are completely backwards compatible.

Most other extensions of this type require that the server only insert the header field or parameter if it is sure the client understands it. In this case, these extensions will need to make use of the Supported request header field mechanism. This mechanism allows a server to determine if the client can understand some extension, so that it can apply the extension to the response. By their nature, these extensions may not always be able to be applied to every response.

If an extension requires a proxy to insert a header field or parameter into a request and this header field or parameter needs to be understood by both UAC and UAS to be executed correctly, a combination of the Require and the Supported mechanism will need to be used. The proxy can insert a Require header field into the request if the Supported header field is present. An example of such an extension is the SIP Session Timer [13].

Yet another type of extension is that which defines new body types to be carried in SIP messages. According to the SIP specification, bodies must be understood by user agents in order to process a request. As such, the interoperability issues are similar to new methods. However, the Content-Disposition header field has been defined to allow a client or server to indicate that the message body is optional [2]. Extensions that define or require new body types SHOULD make them optional for the user agent to process.

When a body must be understood to properly process a request or response, it is preferred that the sending entity know ahead of time whether the new body is understood by the recipient. For requests that establish a dialog, inclusion of Accept in the request and its

success responses is RECOMMENDED. This will allow both parties to determine what body types are supported by their peers. Subsequent messaging between the peers would then only include body types that were indicated as being understood.

4.2. Security

Security is an important component of any protocol. Designers of SIP extensions need to carefully consider if additional security requirements are required over those described in RFC 3261. Frequently, authorization requirements and requirements for end-to-end integrity are the most overlooked.

SIP extensions MUST consider how (or if) they affect usage of the general SIP security mechanisms. Most extensions should not require any new security capabilities beyond general-purpose SIP. If they do, it is likely that the security mechanism has more general-purpose application and should be considered an extension in its own right.

Overall system security requires that both the SIP signaling and the media sessions it established be secured. The media sessions normally use their own security techniques, which are quite distinct from those used by SIP itself. Extensions should take care not to conflate the two. However, specifications that define extensions that impact the media sessions in any way SHOULD consider the interactions between SIP and session security mechanisms.

4.3. Terminology

RFC 3261 has an extensive terminology section that defines terms such as caller, callee, user agent, and header field. All SIP extensions MUST conform to this terminology. They MUST NOT define new terms that describe concepts already defined by a term in another SIP specification. If new terminology is needed, it SHOULD appear in a separate section towards the beginning of the document.

Careful attention must be paid to the actual usage of terminology. Many documents misuse the terms header, header field, and header field values, for example. Document authors SHOULD do a careful review of their documents for proper usage of these terms.

4.4. Syntactic Issues

Extensions that define new methods SHOULD use all capitals for the method name. Method names SHOULD be shorter than 10 characters and SHOULD attempt to convey the general meaning of the request. Method names are case sensitive, and therefore, strictly speaking, they don't have to be capitalized. However, using capitalized method

names keeps with a long-standing convention in SIP and many similar protocols, such as HTTP [15] and RTSP [16].

Extensions that define new header fields that are anticipated to be heavily used MAY define a compact form if those header fields are more than six characters. "Heavily used" means that the percentage of all emitted messages that contain that header field is over thirty percent. Usage of compact forms in these cases is only a MAY because there are better approaches for reducing message overhead [20]. Compact header fields MUST be a single character. When all 26 characters are exhausted, new compact forms will no longer be defined. Header field names are defined by the "token" production in RFC 3261, Section 25.1, and thus include the upper and lowercase letters, the digits 0 through 9, the HYPHEN-MINUS (-), FULL STOP (.), EXCLAMATION MARK (!), PERCENT SIGN (%), ASTERISK (*), LOW LINE (_), PLUS SIGN (+), GRAVE ACCENT (`), APOSTROPHE ('), and TILDE (~). They SHOULD be descriptive but reasonably brief. Although header field names are case insensitive, a single common capitalization SHOULD be used throughout the document. It is RECOMMENDED that each English word present in the header field name have its first letter capitalized. For example, "ThisIsANewHeader".

As an example, the following are poor choices for header field names:

```
ThisIsMyNewHeaderThatDoesntDoVeryMuchButItHasANiceName
--.!A
Function
```

Case sensitivity of parameters and values is a constant source of confusion, a difficulty that plagued RFC 2543 [17]. This has been simplified through the usage of the BNF constructs of RFC 4234 [5], which have clear rules of case sensitivity and insensitivity. Therefore, the BNF for an extension completely defines the matching rules.

Extensions MUST be consistent with the SIP conventions for case sensitivity. Methods MUST be case sensitive. Header field names MUST be case insensitive. Header field parameter names MUST be case insensitive. Header field values and parameter values are sometimes case sensitive, and sometimes case insensitive. However, generally, they SHOULD be case insensitive. Defining a case-sensitive component requires explicitly listing each character through its ASCII code.

Extensions that contain freeform text MUST allow that text to be UTF-8, as per the IETF policies on character set usage [3]. This ensures that SIP remains an internationalized standard. As a general guideline, freeform text is never needed by programs to perform protocol processing. It is usually entered by and displayed to the

user. If an extension uses a parameter that can contain UTF-8-encoded characters, and that extension requires a comparison to be made of this parameter to other parameters, the comparison MUST be case sensitive. Case-insensitive comparison rules for UTF-8 text are, at this time, impossible and MUST be avoided.

Extensions that make use of dates MUST use the SIP-Date BNF defined in RFC 3261. No other date formats are allowed. However, the usage of absolute dates to determine intervals (for example, the time at which some timer fires) is NOT RECOMMENDED. This is because it requires synchronized time between peers, and this is frequently not the case. Therefore, relative times, expressed in numbers of seconds, SHOULD be used.

Extensions that include network-layer addresses SHOULD permit dotted quad IPv4 addresses, IPv6 addresses in the format described in [4], and domain names.

Extensions that have header fields containing URIs SHOULD be explicit about which URI schemes can be used in that header field. Header fields SHOULD allow the broadest set of URI schemes possible that are a match for the semantics of the header field.

Header fields MUST follow the standard formatting for SIP, defined as follows:

```
header           = header-name HCOLON header-value
                  *(COMMA header-value)
header-name      = token
header-value     = value *(SEMI value-parameter)
value-parameter  = token [EQUAL gen-value]
gen-value        = token / host / quoted-string
value           = token / host / quoted-string
```

In some cases, this form is not sufficient. That is the case for header fields that express descriptive text meant for human consumption. An example is the Subject header field in SIP [2]. In this case, an alternate form is:

```
header           = header-name HCOLON [TEXT-UTF8-TRIM]
```

Developers of extensions SHOULD allow for extension parameters in their header fields.

Header fields that contain a list of URIs SHOULD follow the same syntax as the Contact header field in SIP. Implementors are also encouraged to wrap these URI in angle brackets, "<" and ">", at all times. We have found this to be a frequently misimplemented feature.

Beyond the compact form, there is no need to define compressed versions of header field values. Compression of SIP messages SHOULD be handled at lower layers, for example, using IP payload compression [18] or signalling compression [20].

Syntax for header fields is expressed in Augmented Backus-Naur Form and MUST follow the format of RFC 4234 [5]. Extensions MUST make use of the primitive components defined in RFC 3261 [2]. If the construction for a BNF element is defined in another specification, it is RECOMMENDED that the construction be referenced rather than copied. The reference SHOULD include both the document and section number. All BNF elements must be either defined or referenced.

It is RECOMMENDED that BNF be collected into a single section near the end of the document.

All tokens and quoted strings are separated by explicit linear white space. Linear white space, for better or worse, allows for line folding. Extensions MUST NOT define new header fields that use alternate linear white space rules.

All SIP extensions MUST verify that any BNF productions that they define in their grammar do not conflict with any existing grammar defined in other SIP standards-track specifications.

4.5. Semantics, Semantics, Semantics

Developers of protocols often get caught up in syntax issues, without spending enough time on semantics. The semantics of a protocol are far more important. SIP extensions MUST clearly define the semantics of the extensions. Specifically, the extension MUST specify the behaviors expected of a UAC, UAS, and proxy in processing the extension. This is often best described by having separate sections for each of these three elements. Each section SHOULD step through the processing rules in temporal order of the most common messaging scenario.

Processing rules generally specify actions to be taken (in terms of messages to be sent, variables to be stored, and rules to be followed) on receipt of messages and expiration of timers. If an action requires transmission of a message, the rule SHOULD outline requirements for insertion of header fields or other information in the message.

The extension SHOULD specify procedures to be taken in exceptional conditions that are recoverable, or that require some kind of user intervention. Handling of unrecoverable errors does not require specification.

4.6. Examples Section

The specification SHOULD contain a section that gives examples of call flows and message formatting. Extensions that define substantial new syntax SHOULD include examples of messages containing that syntax. Examples of message flows should be given to cover common cases and at least one failure or unusual case.

For an example of how to construct a good examples section, see the message flows and message formatting defined in the Basic Call Flows specification [21]. Note that complete messages SHOULD be used. Be careful to include tags, Via header fields (with the branch ID cookie), Max-Forwards, Content-Lengths, Record-Route, and Route header fields. Example INVITE messages MAY omit session descriptions, and Content-Length values MAY be set to "." to indicate that the value is not provided. However, the specification MUST explicitly call out the meaning of the "." and explicitly indicate that session descriptions were not included.

4.7. Overview Section

Too often, extension documents dive into detailed syntax and semantics without giving a general overview of operation. This makes understanding of the extension harder. It is RECOMMENDED that extensions have a protocol overview section that discusses the basic operation of the extension. Basic operation usually consists of the message flow, in temporal order, for the most common case covered by the extension. The most important processing rules for the elements in the call flow SHOULD be mentioned. Usage of the RFC 2119 [1] terminology in the overview section is NOT RECOMMENDED, and the specification should explicitly state that the overview is tutorial in nature only. This section SHOULD expand all acronyms, even those common in SIP systems, and SHOULD be understandable to readers who are not SIP experts. [27] provides additional guidance on writing good overview sections.

4.8. IANA Considerations Section

Documents that define new SIP extensions will invariably have IANA Considerations sections.

If your extension is defining a new event package, you MUST register that package. RFC 3265 [6] provides the registration template. See

[22] for an example of the registration of a new event package. As discussed in RFC 3427 [10], only standards-track documents can register new event-template packages. Both standards-track and informational specifications can register event packages.

If your extension is defining a new header field, you MUST register that header field. RFC 3261 [2] provides a registration template. See Section 8.2 of RFC 3262 [23] for an example of how to register new SIP header fields. Both standards-track and informational P-header specifications can register new header fields [10].

If your extension is defining a new response code, you MUST register that response code. RFC 3261 [2] provides a registration template. See Section 6.4 of RFC 3329 [19] for an example of how to register a new response code. As discussed in RFC 3427 [10], only standards-track documents can register new response codes.

If your extension is defining a new SIP method, you MUST register that method. RFC 3261 [2] provides a registration template. See Section 10 of RFC 3311 [24] for an example of how to register a new SIP method. As discussed in RFC 3427 [10], only standards-track documents can register new methods.

If your extension is defining a new SIP header field parameter, you MUST register that header field parameter per the guidelines in RFC 3968 [7]. Section 4.1 of that specification provides a template. Only IETF approved specifications can register new header field parameters. However, there is no requirement that these be standards track.

If your extension is defining a new SIP URI parameter, you MUST register that URI parameter per the guidelines in RFC 3969 [8]. Section 4.1 of that specification provides a template. Only standards-track documents can register new URI parameters.

Many SIP extensions make use of option tags, carried in the Require, Proxy-Require, and Supported header fields. Section 4.1 discusses some of the issues involved in the usage of these header fields. If your extension does require them, you MUST register an option tag for your extension. RFC 3261 [2] provides a registration template. See Section 8.1 of RFC 3262 [23] for an example of how to register an option tag. Only standards-track RFCs can register new option tags.

Some SIP extensions will require establishment of their own IANA registries. RFC 2434 [25] provides guidance on how and when IANA registries are established. For an example of how to set one up, see Section 6 of RFC 3265 [6] for an example.

4.9. Document-Naming Conventions

An important decision to be made about the extension is its title. The title **MUST** indicate that the document is an extension to SIP. It is **RECOMMENDED** that the title follow the basic form of "A [summary of function] for the Session Initiation Protocol (SIP)", where the summary of function is a one- to three-word description of the extension. For example, if an extension defines a new header field, called Make-Coffee, for making coffee, the title would read, "Making Coffee with the Session Initiation Protocol (SIP)". It is **RECOMMENDED** that these additional words be descriptive rather than naming the header field. For example, the extension for making coffee should not be named "The Make-Coffee Header for the Session Initiation Protocol".

For extensions that define new methods, an acceptable template for titles is "The Session Initiation Protocol (SIP) X Method" where X is the name of the method.

Note that the acronym SIP **MUST** be expanded in the titles of RFCs, as per [26].

4.10. Additional Considerations for New Methods

Extensions that define new methods **SHOULD** take into consideration and discuss the following issues:

- o Can it contain bodies? If so, what is the meaning of the presence of those bodies? What body types are allowed?
- o Can a transaction with this request method occur while another transaction, in the same and/or reverse direction, is in progress?
- o The extension **MUST** define which header fields can be present in requests of that method. It is **RECOMMENDED** that this information be represented as a new column of Table 2/3 of RFC 3261 [2]. The table **MUST** contain rows for all header fields defined in standards-track RFCs at the time of writing of the extension.
- o Can the request be sent within a dialog, or does it establish a dialog?
- o Is it a target refresh request?
- o Extensions to SIP that define new methods **MAY** specify whether offers and answers can appear in requests of that method or its responses. However, those extensions **MUST** adhere to the protocol

rules specified in [28] and MUST adhere to the additional constraints for offers and answers as specified in SIP [2].

- o Because of the nature of reliability treatment of requests with new methods, those requests need to be answered immediately by the UAS. Protocol extensions that require longer durations for the generation of a response (such as a new method that requires human interaction) SHOULD instead use two transactions - one to send the request, and another in the reverse direction to convey the result of the request. An example of that is SUBSCRIBE and NOTIFY [6].
- o The SIP specification [2] allows new methods to specify whether transactions using that new method can be canceled using a CANCEL request. Further study of the non-INVITE transaction [14] has determined that non-INVITE transactions must be completed as soon as possible. New methods must not plan for the transaction to pend long enough for CANCEL to be meaningful. Thus, new methods MUST declare that transactions initiated by requests with that method cannot be canceled. Future work may relax this restriction, at which point these guidelines will be revised.
- o New methods that establish a new dialog must discuss the impacts of forking. The design of such new methods should follow the pattern of requiring an immediate request in the reverse direction from the request establishing a dialog, similar to the immediate NOTIFY sent when a subscription is created per RFC 3265 [6].

The reliability mechanisms for all new methods must be the same as for BYE. The delayed response feature of INVITE is only available in INVITE, never for new methods. The design of new methods must encourage an immediate response. If the application being enabled requires a delay, the design SHOULD follow a pattern using multiple transactions, similar to RFC 3265's use of NOTIFYs with different Subscription-State header field values (pending and active in particular) in response to SUBSCRIBE [6].

4.11. Additional Considerations for New Header Fields or Header Field Parameters

The most important issue for extensions that define new header fields or header field parameters is backwards compatibility. See Section 4.1 for a discussion of the issues. The extension MUST detail how backwards compatibility is addressed.

It is often tempting to avoid creation of a new method by overloading an existing method through a header field or parameter. Header fields and parameters are not meant to fundamentally alter the meaning of the method of the request. A new header field cannot

change the basic semantic and processing rules of a method. There is no shortage of method names, so when an extension changes the basic meaning of a request, a new method SHOULD be defined.

For extensions that define new header fields, the extension MUST define the request methods the header field can appear in, and what responses it can be used in. It is RECOMMENDED that this information be represented as a new row of Table 2/3 of RFC 3261 [2]. The table MUST contain columns for all methods defined in standards-track RFCs at the time of writing of the extension.

4.12. Additional Considerations for New Body Types

Because SIP can run over UDP, extensions that specify the inclusion of large bodies (where large is several times the ethernet MTU) are frowned upon unless end-to-end congestion controlled transport can be guaranteed. If at all possible, the content SHOULD be included indirectly [9], even if congestion controlled transports are available.

Note that the presence of a body MUST NOT change the nature of the message. That is, bodies cannot alter the state machinery associated with processing a request of a particular method or a response.

Bodies enhance this processing by providing additional data.

5. Interactions with SIP Features

We have observed that certain capabilities of SIP continually interact with extensions in unusual ways. Writers of extensions SHOULD consider the interactions of their extensions with these SIP capabilities and document any unusual interactions, if they exist. The following are the most common causes of problems:

Forking: Forking by far presents the most troublesome interactions with extensions. This is generally because it can cause (1) a single transmitted request to be received by an unknown number of UASes, and (2) a single INVITE request to have multiple responses.

CANCEL and ACK: CANCEL and ACK are "special" SIP requests, in that they are exceptions to many of the general request processing rules. The main reason for this special status is that CANCEL and ACK are always associated with another request. New methods SHOULD consider the meaning of cancellation, as described above. Extensions that define new header fields in INVITE requests SHOULD consider whether they also need to be included in ACK and CANCEL. Frequently they do, in order to allow a stateless proxy to route the CANCEL or ACK identically to the INVITE.

Routing: The presence of Route header fields in a request can cause it to be sent through intermediate proxies. Requests that establish dialogs can be record-routed, so that the initial request goes through one set of proxies, and subsequent requests through a different set. These SIP features can interact in unusual ways with extensions.

Stateless Proxies: SIP allows a proxy to be stateless. Stateless proxies are unable to retransmit messages and cannot execute certain services. Extensions that depend on some kind of proxy processing SHOULD consider how stateless proxies affect that processing.

Dialog Usages: SIP allows for requests that normally create their own dialog (such as SUBSCRIBE) to be used within a dialog created by another method (such as INVITE). In such a case, there are said to be multiple usages of that dialog. Extensions SHOULD consider their interaction with dialog usages. In particular, extensions that define new error response codes SHOULD describe whether that response code causes the dialog and all usages to terminate, or just a specific usage.

6. Security Considerations

The nature of this document is such that it does not introduce any new security considerations. However, many of the principles described in the document affect whether a potential SIP extension design is likely to support the SIP security architecture.

7. Acknowledgements

The authors would like to thank Rohan Mahy and Spencer Dawkins for their comments. Robert Sparks contributed important text on CANCEL issues. Thanks to Allison Mankin for her support.

8. References

8.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [3] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

- [4] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [5] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.
- [6] Roach, A.B., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.
- [7] Camarillo, G., "The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP)", BCP 98, RFC 3968, December 2004.
- [8] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, December 2004.
- [9] Burger, E., Ed., "A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages", RFC 4483, May 2006.

8.2. Informative References

- [10] Mankin, A., Bradner, S., Mahy, R., Willis, D., Ott, J., and B. Rosen, "Change Process for the Session Initiation Protocol (SIP)", BCP 67, RFC 3427, December 2002.
- [11] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.
- [12] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.
- [13] Donovan, S. and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)", RFC 4028, April 2005.
- [14] Sparks, R., "Problems Identified Associated with the Session Initiation Protocol's (SIP) Non-INVITE Transaction", RFC 4321, January 2006.
- [15] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [16] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

- [17] Handley, M., Schulzrinne, H., Schooler, E., and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.
- [18] Shacham, A., Monsour, B., Pereira, R., and M. Thomas, "IP Payload Compression Protocol (IPComp)", RFC 3173, September 2001.
- [19] Arkko, J., Torvinen, V., Camarillo, G., Niemi, A., and T. Haukka, "Security Mechanism Agreement for the Session Initiation Protocol (SIP)", RFC 3329, January 2003.
- [20] Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., and J. Rosenberg, "Signaling Compression (SigComp)", RFC 3320, January 2003.
- [21] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., and K. Summers, "Session Initiation Protocol (SIP) Basic Call Flow Examples", BCP 75, RFC 3665, December 2003.
- [22] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.
- [23] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.
- [24] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.
- [25] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.
- [26] Reynolds, J. and R. Braden, "Instructions to Request for Comments (RFC) Authors", Work in Progress, July 2004.
- [27] Rescorla, E. and IAB, "Writing Protocol Models", RFC 4101, June 2005.
- [28] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

Authors' Addresses

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

Phone: +1 973 952-5000
EMail: jdrosen@cisco.com
URI: <http://www.jdrosen.net>

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027
US

EMail: schulzrinne@cs.columbia.edu
URI: <http://www.cs.columbia.edu/~hgs>

Full Copyright Statement

Copyright (C) The Internet Society (2006).

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

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

Intellectual Property

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

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

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

Acknowledgement

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

