Internet Engineering Task Force   SIP WG
Internet Draft
J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia U.
draft-ietf-sip-callerprefs-07.txt
November 4, 2002
Expires: May 2003

Session Initiation Protocol (SIP) Caller Preferences and Callee Capabilities

STATUS OF THIS MEMO

Abstract

This document describes a set of extensions to the Session Initiation Protocol (SIP) which allow a caller to express preferences about request handling in servers. These preferences include the ability to select which URIs a request gets routed to, and to specify certain request handling directives in proxies and redirect servers. It does so by defining four new request headers, Accept-Contact, Reject-Contact, Require-Contact and Request-Disposition, which specify the caller's preferences. The extension also defines new parameters for the Contact header that describe the capabilities and characteristics of a UA.

Table of Contents

1 Introduction

When a Session Initiation Protocol (SIP) [1] server receives a request, there are a number of decisions it can make regarding processing of the request. These include:

- o whether to proxy or redirect the request
- o which URIs to proxy or redirect to
- o whether to fork or not
- o whether to search recursively or not
- o whether to search in parallel or sequentially

The server can base these decisions on any local policy. This policy can be statically configured, or can be based on programmatic execution or database access.

However, the administrator of the server is the not the only entity with an interest in request processing. There are at least three parties which have an interest: (1) the administrator of the server, (2) the user that sent the request, and (3) the user to whom the request is directed. The directives of the administrator are embedded in the policy of the server. The preferences of the user to whom the request is directed (referred to as the callee, even though the request may not be INVITE) can be expressed most easily through a script written in some type of scripting language, such as the Call Processing Language (CPL) [16]. However, no mechanism exists to incorporate the preferences of the user that sent the request (also referred to as the caller, even though the request may not be INVITE). For example, the caller might want to speak to a specific user, but want to reach them only at work, because the call is a business call. As another example, the caller might want to reach a user, but not their voicemail, since it is important that the caller talk to the called party. In both of these examples, the caller's preference amounts to having a proxy make a particular routing choice based on the preferences of the caller.

This extension allows the requestor to have these preferences met. It does so by specifying mechanisms by which a caller can provide preferences on processing of a request. There are two types of preferences. One of them, called request handling preferences, are encapsulated in the Request-Disposition header field. They provides specific request handling directives for a server. The other, called feature preferences, are present in the Accept-Contact, Reject-Contact, and Require-Contact header fields. They allow the caller to

provide a feature set [2] that expresses its preferences on the characteristics of the UA that is to be reached. These are matched with a feature set carried in the Contact header of a REGISTER request, which describes the capabilities of the UA represented by the Contact URI. The extension is very general purpose, and not tied to a particular service. Rather, it is a tool that can be used in the development of many services.

Indeed, the feature sets uploaded to the server in REGISTER requests can be used for a variety of purposes, not just meeting caller preferences. Applications can use this information to tailor information sent to a user as part of an instant message, for example [17].

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 [3] and indicate requirement levels for compliant SIP implementations.

3 Definitions

Caller: Within the context of this specification, a caller refers to the user on whose behalf a UAC is operating. It is not limited to a user who's UAC sends the INVITE method.

Feature: As defined in RFC 2703 [18], a piece of information about the media handling properties of a message passing system component or of a data resource. For example, the SIP methods supported by a UA represent a feature.

Feature Tag: As defined in RFC 2703 [18], a feature tag is a name that identifies a feature.

Media Feature: As defined in RFC 2703, [18], a media feature is information that indicates facilities assumed to be available for the message content to be properly rendered or otherwise presented. Media features are not intended to include information that affects message transmission.

In the context of this specification, a media feature is information that indicates facilities for handling SIP requests, rather than specifically for content. In that sense, it is used synonymously with feature.

Feature Collection: As defined in RFC 2533 [2], a feature collection is a collection of different media features and

associated values. This might be viewed as describing a specific rendering of a specific instance of a document or resource by a specific recipient.

Feature Set: As defined in RFC 2703 [18], a feature set is Information about a sender, recipient, data file or other participant in a message transfer which describes the set of features that it can handle. Where a 'feature' describes a single identified attribute of a resource, a 'feature set' describes full set of possible attributes.

Feature Preferences: Caller preferences that described desired properties of a UA that the request is to be routed to. These preferences are carried in the Accept-Contact, Reject-Contact and Require-Contact header fields.

Request Handling Preferences: Caller preferences that describe desired request treatment at a server. These preferences are carried in the Request-Disposition header field.

Feature Parameters: A set of SIP header field parameters that can appear in the Contact, Accept-Contact, Reject-Contact and Require-Contact header fields. The feature parameters represent an encoding of a feature set. There is a one-one mapping between a set of feature parameters and a feature set predicate, as both represent alternative encodings of a feature set.

Capability: As defined in RFC 2703 [18], a capability is an attribute of a sender or receiver (often the receiver) which indicates an ability to generate or process a particular type of message content.

Filter: A single expression in a feature predicate.

Simple Filter: An expression in a feature predicate which is a comparison (equality or inequality) of a feature tag against a feature value.

Disjunction: A boolean OR operation across some number of terms.

Predicate: A boolean expression.

Feature Set Predicate: From RFC 2533 [2], a feature set predicate is a function of an arbitrary feature collection value which returns a Boolean result. A TRUE result is taken to mean that the corresponding feature collection belongs to some set of media feature handling capabilities

defined by this predicate.

Contact Predicate: The feature set predicate associated with a URI registered in the Contact header field of a REGISTER request. The contact predicate is derived from the feature parameters in the Contact header field.

4 Overview of Operation

This extension defines a set of additional parameters to the Contact header field, called feature parameters. Each parameter name is a feature tag, as defined in RFC 2703 [18], that defines a capability for the UA associated with the Contact header field value. For example, there is a parameter for the SIP methods supported by the UA. Each feature parameter has a value; that value is the set of feature values for that feature tag. Put together, all of the feature parameters specify a feature set that is supported by the UA associated with that Contact header field value.

When a UA registers, it places these parameters in the Contact header field value to provide a feature set for each URI it is registering. The feature parameters are also mirrored in the Contact header field in a REGISTER response. The proxy can use this feature set to route requests based on caller preferences. Furthermore, Contact header fields in requests and responses that establish a dialog can contain these parameters. That allows a UA in a dialog to indicate its feature set to its peer. For example, by including the "voicemail" feature tag with value "TRUE" in the 200 OK to an INVITE, the UAS can indicate to the UAC that it is a voicemail server. This information is useful for user interfaces, as well as automated call handling.

When a caller sends a request, it can optionally include new header fields which request certain handling at a server. These preferences fall into two categories. The first category, called request handling preferences, are carried in the Request-Disposition header field. They describe specific behavior that is desired at a server. Request handling preferences include whether the caller wishes the server to proxy or redirect, and whether sequential or parallel search is desired. These preferences can be applied at every proxy or redirect server on the call signaling path.

The second category of preferences, called feature preferences, are carried in the Accept-Contact, Reject-Contact, and Require-Contact header fields. These header fields also contain feature sets, represented by the same feature parameters that are used in the Contact header. Here, the feature parameters represent the caller's preferences. The Accept-Contact header field contains feature sets that describe UAs that the caller would like to reach. The Reject-

Contact header field contains feature sets which, if matched by a UA, imply that the request should not be routed to that UA. The Require-Contact header field contains feature sets which, if not matched by a UA, imply that the request should not be routed to that UA. Require-Contact and Accept-Contact are similar, but Require-Contact is more forceful. Contacts which don't match are outright rejected, whereas with Accept-Contact, they are tried as fallbacks.

Proxies use the information in the Accept-Contact, Reject-Contact and Require-Contact header fields to select amongst registered contacts. Proxies also compute implicit preferences from the request. These are caller preferences that are not explicitly placed into the request, but can be inferred from the presence of other message components. As an example, if the request method is INVITE, this is an implicit preference to route the call to a UA that supports the INVITE method.

Both request handling and feature preferences can appear in any request, not just INVITE. However, they are only useful in requests where proxies need to determine a request target. If the domain in the request URI is not owned by any proxies along the request path, those proxies will never access a location service, and therefore, never have the opportunity to apply the caller preferences. This makes sense; typically, the request URI will identify a UAS for mid-dialog requests. In those cases, the routing decisions were already made on the initial request, and it makes no sense to redo them for subsequent requests in the dialog.

5 Usage of the Content Negotiation Framework

This specification makes heavy use of the terminology and concepts in the content negotiation work carried out within the IETF, and documented in several RFCs. The ones relevant to this specification are RFC 2506 [4] which provides a template for registering media feature tags, RFC 2533 [2] which presents a syntax and matching algorithm for media feature sets, RFC 2738 [5], which provides a minor update to RFC 2533, and RFC 2703 [18] which provides a general framework for content negotiation.

In case the reader does not have the time to read those specifications, Appendix A provides a brief overview of the concepts and terminology in those documents that is critical for understanding this specification.

Since the content negotiation work was primarily meant to apply to documents or other resources with a set of possible renderings, it is not immediately apparent how it is used to model the SIP entities at hand. The goal of this specification is to allow a UA to express its feature set, and for a caller to express a feature set that describes

properties of a desirable (or undesirable) UA. Therefore, we are using feature sets to describe SIP user agents.

A feature set is composed of a set of feature collections, each of which represents a specific rendering supported by the entity described by the feature set. In the context of a SIP user agent, a feature collection represents an instantaneous modality. That is, if you look at the run time processing of a SIP UA, and take a snapshot in time, the feature collection describes what it is doing at that very instant.

This model is important, since it provides guidance on how to determine whether something is a value for a particular feature tag, or a feature tag by itself. If two properties can be exhibited by a UA simultaneously, so that both are present in an instantaneous modality, they need to be represented by separate media feature tags. For example, a UA may be able to support some number of media types - audio, video, and messaging. Should each of these be different values for a single "media-types" feature tag, or should each of them be a separate boolean feature tag? The model provides the answer. Since, at any instant of time, a UA could be handling both audio and video, they need to be separate media feature tags. However, the SIP methods supported by a UA can each be represented as different values for the same media feature tag (the "methods" tag), because fundamentally, a UA processes a single request at a time. It may be multi-threading, so that it appears that this is not so, but at a purely functional level, it is true.

Clearly, there are weaknesses in this model, but it serves as a useful guideline for applying the concepts of RFC 2533 to the problem at hand.

6 UA Behavior

UA behavior covers four separate cases. The first is registration, where a UA can declare its capabilities. The second is expression of preferences, where a UA can tell a proxy how it wants the request to be processed and routed. The third is expressing of capabilities, through a feature set, in the Contact header field of a target refresh request or response. The fourth is UAS processing of the request handling and feature preferences.

6.1 Expressing Capabilities in a Registration

When a UA registers, it MAY construct a feature predicate for each Contact URI it registers. In the text that follows, this process is described in terms of RFC 2533 [2] (and its minor update, [5]) syntax and constructs, followed by a conversion to the syntax used in this

specification. However, this represents a logical flow of processing. There is no requirement that an implementation actually use RFC 2533 syntax as an intermediate step.

The feature predicate constructed by a UA MUST be an AND of terms. Each term is either an OR of simple filters (called a disjunction), or a single simple filter. In the case of an OR of simple filters, each filter MUST indicate feature values for the same feature tag (i.e., the disjunction represents a set of values for a particular feature tag), and each element of the conjunction MUST be for a different feature tag. Each filter can be an equality, the negation of an equality, or in the case of numeric feature tags, an inequality or range. This feature predicate is then converted to a list of feature parameters using the procedure specified in Section 11. Those feature parameters are added to the the Contact header field value containing the URI that the parameters apply to.

A UA MAY use any feature tags that are registered through IANA in the IETF or global trees [4]; this document registers several that are appropriate for SIP. It is also permissible to use the URI tree [4] for expressing vendor-specific feature tags. Feature tags in any other trees created through IANA MAY also be used.

A UA MAY include the "schemes" feature tag in its feature parameters. However, this tag MUST include a value that matches the scheme of the URI being registered. For example, if a SIP URI is being registered, the schemes parameter can include a SIP and TEL URI [6]. If this feature tag is omitted, the proxy will assume an implicit value for it, equal to the scheme of the registered URI.

It is RECOMMENDED that a UA provide complete information in its feature parameters. That is, it SHOULD provide information on as many feature tags as possible. The mechanisms in this specification work best when user agents register complete feature sets. This includes features that are supported, and those that are not. For example, if a UA does not support video, it SHOULD include a 'video="FALSE"' parameter in its registered Contact. Furthermore, when a UA registers values for a particular feature tag, it MUST list all values that it supports. For example, when including the methods feature tag, a UA MUST list all methods it supports. The matching algorithms in this specification assume that ommission of a value from a list means that the value is not supported.

The REGISTER request MAY contain a Require header field with the value "pref" if the client wants to be sure that the registrar understands the extensions defined in this specification. In absence of the Require header field, a server that does not understand this extension will simply ignore the Contact header field parameters.

As an example, a UA that supports audio and video media types, is a voicemail server, and is not mobile would construct a feature predicate like this:

```
(& (audio=TRUE)
   (video=TRUE)
   (voicemail=TRUE)
   (mobility=fixed)
   (| (methods=INVITE) (methods=BYE) (methods=OPTIONS) (methods=ACK)
      (methods=CANCEL)))
```

These would be converted into feature parameters and included in the REGISTER request:

```
REGISTER sip:example.com SIP/2.0
From: sip:user@example.com;tag=asd98
To: sip:user@example.com
Call-ID: hh89as0d-asd88jkk@host.example.com
CSeq: 9987 REGISTER
Max-Forwards: 70
Via: SIP/2.0/UDP host.example.com;branch=z9hG4bKnashds8
Contact: <sip:user@host.example.com>;audio="TRUE";video="TRUE"
  ;voicemail="TRUE";mobility="fixed"
  ;methods="INVITE,BYE,OPTIONS,ACK,CANCEL"
Content-Length: 0
```

6.2 Expressing Preferences in a Request

A caller wishing to express preferences for a request includes Accept-Contact, Reject-Contact, Require-Contact or Request-Disposition header fields in the request, depending on their particular preferences. No additional behavior is required after the request is sent.

The Accept-Contact, Reject-Contact, Require-Contact and Request-Disposition header fields in an ACK for a non-2xx final response, or in a CANCEL request, MUST be equal to the values in the original request being acknowledged or cancelled. This is to ensure proper operation through stateless proxies.

If the UAC wants to be sure that servers understand the header fields described in this specification, it MAY include a Proxy-Require header field with a value of "pref". However, this is NOT RECOMMENDED, as it leads to interoperability problems. In any case, caller preferences can only be considered preferences - there is no guarantee that the requested service or capability is executed. As such, inclusion of a Proxy-Require header field does not mean the preferences will be executed, just that the caller preferences extension is understood by the proxies.

6.2.1 Request Handling Preferences

The Request-Disposition header field specifies caller preferences for how a server should process a request. Its value is a list of tokens, each of which specifies a particular processing directive.

The syntax of the header field can be found in Section 10, and the semantics of the directives are described in Section 8.1.

6.2.2 Feature Set Preferences

A UAC can indicate caller preferences for the capabilities of a UA that should be reached or not reached as a result of sending a SIP request. To do that, it adds one or more Accept-Contact, Reject-Contact, and Require-Contact header field values. Each header field value is either a URI or the wildcard "*", along with feature parameters that define a feature set. In the case of Accept-Contact, each value can also have a q-value parameter.

Each feature set MUST follow the constraints of Section 6.1. That is, when represented by a feature set predicate, each predicate MUST be a conjunction of terms. Each term is either an OR of simple filters (called a disjunction), or a single simple filter. In the case of an OR of simple filters, each filter MUST indicate feature values for the same feature tag (i.e., the disjunction represents a set of values for a particular feature tag), and each element of the conjunction MUST be for a different feature tag. Each filter can be an equality, the negation of an equality, or in the case of numeric feature tags, an inequality or range.

The feature sets placed into these header fields MAY overlap; that is, a UA MAY indicate preferences for feature sets that match according to the matching algorithm of RFC 2533 [2]. The UA MAY use any feature tag in an IANA registry or in a vendor defined URI tree.

Note that the UAC can express explicit preferences for the methods, event packages and priorities supported by a UA. As described in Section 7.2.2, a proxy will compute implicit preferences from the

request if explicit ones are not provided.

The Reject-Contact header field allows the UAC to specify that a UA should not be contacted if it matches any of the values of the header field. Each value of the Reject-Contact header field contains a URI or a "*" and is parameterized by a set of feature parameters. Any UA whose capabilities match the feature set described by the feature parameters, and whose URI matches the URI in the value (if specified), matches the value. A value of "*" indicates a wildcard operation on the URI, so that any URI matches. As with registrations, it is not necessary for a UAC to construct the feature set in RFC 2533 syntax as an intermediate step. The only requirement is that the feature parameters, if converted back to RFC 2533 format, meet the requirements above.

The Require-Contact header field allows the UAC to specify that a UA should not be contacted if it doesn't match all of the values of the header field. Each value of the Require-Contact header field contains a URI or a "*" and is parameterized by set of feature parameters. Any UA whose capabilities match the feature set described by the feature parameters, and whose URI matches the URI in the value (if specified), matches the value. A value of "*" indicates a wildcard operation on the URI, so that any URI matches. As with registrations, it is not necessary for a UAC to construct the feature set in RFC 2533 syntax as an intermediate step. The only requirement is that the feature parameters, if converted back to RFC 2533 format, meet the requirements above.

The Accept-Contact header field allows the UAC to specify that a UA should be contacted if it matches some or all of the values of the header field. If a UA matches none of the values, it should be contacted as a last resort. Each value of the Accept-Contact header field contains a URI or a "*" and is parameterized by a set of feature parameters. Any UA whose capabilities match the feature set described by the feature parameters, and whose URI matches the URI in the value (if specified), matches the value. A value of "*" indicates a wildcard operation on the URI, so that any URI matches. The q-value provides a weighting operation, allowing the UAC to request preferential routing to UAs that match that value above other values. As with registrations, it is not necessary for a UAC to construct the feature set in RFC 2533 syntax as an intermediate step. The only requirement is that the feature parameters, if converted back to RFC 2533 format, meet the requirements above.

6.3 Indicating Feature Sets in Remote Target URIs

Target refresh requests and responses are used to establish and modify the remote target URI. The remote target URI is contained in

the Contact header field. A UAC or UAS MAY add feature parameters to the Contact header field value in target refresh requests and responses, for the purpose of indicating the capabilities of the UA. To do that, it constructs a feature set predicate according to the constraints of Section 6.1, and converts it to a set of feature parameters using the rules in Section 11. These are then added as Contact header field parameters in the request or response.

The feature parameters can be included in both initial requests and mid-dialog request, and MAY change mid-dialog to signal a change in UA capabilities.

There is overlap in the caller preferences mechanism with the Allow, Accept, Accept-Language, and Allow-Events [7] header fields, which can also be used in target refresh requests. Specifically, the Allow header field and methods feature tag indicate the same information. The Accept header field and the type feature tag indicate the same information. The Accept-Language header field and the language feature tag indicate the same information. The Allow-Events header field and the events feature tag indicate the same information. It is possible that other header fields and feature tags defined in the future may also overlap. When there exists a feature tag that describes a capability that can also be represented with a SIP header field, a UA MUST use the header field to describe the capability. A UA receiving a message that contains both the header field and the feature tag MUST use the header field, and not the feature tag.

6.4 Request Handling and Feature Set Preferences

When a UAS compliant to this specification receives a request whose request-URI correspods to one of its registered Contacts, it SHOULD apply the behavior described in Section 7 as if it were a proxy for the domain in the request-URI. The UAS acts as if its location database contains a single request target for the request-URI. That target is associated with a feature set. The feature set is the same as the one placed in the registration of the URI in the request-URI. It also adds the uri-user and uri-domain terms to the conjunction as described in Section 7.2.1.

> Having a UAS perform the matching operations as if it were a proxy has many benefits. First, it allows caller preferences to be honored even if the proxy doesn't support the extension. Secondly, and perhaps more importantly, feature set processing of preferences for the URI will only occur at a UA, not at a proxy. Thats because the UA is the only one that adds the uri-user and uri-domain terms to the feature set describing a request target.

7 Proxy Behavior

Proxy behavior consists of two orthogonal sets of rules - one for processing the Request-Disposition header field, and one for processing the URI and feature set preferences in the Accept-Contact, Reject-Contact, and Require-Contact header fields.

7.1 Request-Disposition Processing

If the request contains a Request-Disposition header field, the server SHOULD execute the directives as described in Section 8.1, unless it has local policy configured to direct it otherwise.

7.2 Preference and Capability Matching

A proxy compliant to this specification MUST NOT apply the preferences matching operation described here to a request unless it is the owner of the domain in the request URI, and accessing a location service that has capabilities associated with request targets. However, if it is the owner of the domain, and accessing a location service that has capabilities associated wth request targets, it SHOULD apply the processing described in this section. Typically, this is a proxy that is using a registration database to determine the request targets. However, if a proxy knows about capabilities through some other means, it SHOULD apply the processing defined here as well.

The processing is described through a conversion from the syntax described in this specification to RFC 2533 syntax, followed by a matching operation and a sorting of resulting contact values. The usage of RFC 2533 syntax as an intermediate step is not required, it only serves as a useful tool to describe the behavior required of the proxy. A proxy can use any steps it likes so long as the results are identical to the ones that would be achieved with the processing described here.

7.2.1 Extracting Explicit Preferences

The first step in proxy processing is to extract explicit preferences. To do that, it looks for the Accept-Contact, Reject-Contact and Require-Contact header fields.

For each value of those header fields, it SHOULD convert all parameters except for the q-value to the syntax of RFC 2533, based on the rules in Section 11. If a value of the header field was not a "*", it SHOULD take the URI in that value, and add two terms to the top level conjunction:

(uri-user=<user part of URI>)

and

(uri-domain=<host portion of URI>)

If the user part of the SIP URI is absent, the uri-user term is not added, only the uri-domain one. No URI parameters are used. Note that these are not "real" feature tags; they are not registered with IANA and cannot appear anywhere in actual form. They are merely added in order to perform the matching operation.

The result will be a set of feature set predicates in conjunctive normal form, each of which is associated with one of the three preference header fields. If there was a q parameter associated with a header field value in the Accept-Contact header field, the feature set predicate derived from that header field value is assigned a preference equal to that q value.

7.2.2 Extracting Implicit Preferences

The proxy then applies any "implicit" preferences. These preferences are ones not explicitly stated in the three header fields, but implied by the presence of other header fields in the request.

7.2.2.1 Priority

The Priority header field is an indication of a caller preference - a desire to be routed to a UA that can handle requests of the desired priority. If the request contained a Priority header field, the proxy looks for feature tags with the value "priority" in all feature set predicates. If that feature tag is not used in any of the predicates, the proxy creates a new feature set predicate, and associates it with the Accept-Contact header field (note that there is no modification of the message implied - only an association for the purposes of processing). The new predicate looks like:

(& (priority>=[numeric value of the Priority header field]))

The numeric value of the Priority header field is obtained through the procedures described in Section 9.11. For example, if the request had a Priority header field with a value of urgent, the proxy would create the following predicate:

```
(& (priority >= 3))
```

7.2.2.2 Methods

Another implicit preference is the method. When a UAC sends a request with a specific method, it is an implicit preference to have the request routed only to UAs that support that method. To support this implicit preference, the proxy looks for feature tags with the value "methods" in all feature set predicates. If that feature tag is not used in any of the predicates, the proxy examines the predicates associated with the Require-Contact header field. If there are no predicates associated with that header field, the proxy creates a new empty feature set predicate, and associates it with the Require-Contact header field (note that there is no modification of the message implied - only an association for the purposes of processing). In this case, an empty predicate is one with a conjunction, but no terms in that conjunction yet.

For all predicates associated with the Require-Contact header field (including the one which may have just been created), the proxy SHOULD add a term to the conjunction of the following form:

```
(methods=[method of request])
```

7.2.2.3 Event Packages

For requests that establish a subscription [7], the Event header field is another expression of an implicit preference. It expresses a desire for the request to be routed only to a server than supports the given event package. To implement that implicit preference, the proxy looks for feature tags with the value "events" in all feature set predicates. If that feature tag is not used in any of the predicates, the proxy examines the predicates associated with the Require-Contact header field. If there are no predicates associated with that header field, the proxy creates a new empty feature set

predicate, and associates it with the Require-Contact header field (note that there is no modification of the message implied - only an association for the purposes of processing). In this case, an empty predicate is one with a conjunction, but no terms yet.

For all predicates associated with the Require-Contact header field (including the one which may have just been created), the proxy SHOULD add a term of the following form:

```
(events=[value of the Event header field])
```

7.2.2.4 Media Types

Another implicit preference is for the sessions that are to be established. If a UA generates an INVITE request with a session description that includes video, this is an implicit preference to be connected to a UA that supports video. To implement this implicit preference, the proxy looks for feature tags with the values "audio", "video", "application", "message", "text" or "image" in all feature set predicates. If none of those feature tags are used in any of the predicates, the proxy MAY create a new feature set predicate, and associate it with the Accept-Contact header field (note that there is no modification of the message implied - only an association for the purposes of processing). This predicate has a term for each top-level media type listed in the session description, with a value of TRUE. For example, if the request is an INVITE request, with a Session Description Protocol (SDP) [19] body, where the SDP contains an audio and a video media description, the proxy would construct the following predicate:

```
(& (audio=TRUE)
   (video=TRUE))
```

This implicit preference is added with MAY strength, and not SHOULD, since it requires the proxy to examine the body of the request. This can have performance implications, and won't always be possible. For example, if the body is encrypted, the proxy cannot examine it.

7.2.2.5 Languages

The languages understood by the caller is another form of implicit preference. The Accept-Language header field contains a list of the languages that content should be returned in. It is reasonable to imply that the caller would like the call to be routed to a user that speaks those languages as well. To implement that implicit preference, the proxy looks for feature tags with the value "language" in all feature set predicates. If that feature tag is not used in any of the predicates, the proxy creates a new feature set predicate for each value in the Accept-Language header field, and associates it with the Accept-Contact header field (note that there is no modification of the message implied - only an association for the purposes of processing). Each predicate is of the following form:

```
(& (language=[value of the Accept-Language header field]))
```

Furthermore, if an Accept-Language header field value had a q-value associated with it, that q-value is associated with the corresponding feature set predicate.

7.3 Constructing Contact Predicates

The proxy then takes each URI in the target set (the set of URI it is going to proxy or redirect to), and obtains its capabilities as an RFC 2533 formatted feature set predicate. This is called a contact predicate. If target URI was obtained through a registration, the proxy computes the contact predicate by taking all Contact URI parameters except for the q and expires parameters, and converting them to RFC 2533 syntax using the rules of Section 8.1.

If the contact predicate doesn't already contain a "schemes" feature tag, the proxy SHOULD add a term containing one, whose value is equal to the scheme of the URI.

The resulting predicate is associated with a q-value. If the contact predicate was learned through a REGISTER request, the q-value is equal to the q-value in the Contact header field parameter, else "1.0" if not specified.

As an example, if a REGISTER request had the following Contact URI:

```
Contact: sip:1.2.3.4;mobility="fixed";q=0.8
```

The proxy would compute the following contact predicate, associating it with a q-value of 0.8:

```
(& (mobility=fixed)
   (schemes=sip))
```

7.4 Matching

It is important to note that the proxy does not have to know anything about the meaning of the feature tags that it is comparing in order to perform the matching operation. The rules for performing the comparison depend on syntactic hints present in the values of each feature tag. For example, a predicate such as:

```
foo>=4
```

implies that the feature tag foo is a numeric value. The matching rules in RFC 2533 only require to know whether the feature tag is a numeric, token, quoted string, etc.

First, the proxy applies the predicates associated with the Reject-Contact header field.

For each contact predicate, each Reject-Contact predicate (that is, each predicate associated with the Reject-Contact header field) is examined. If that Reject-Contact predicate contains a filter for a feature tag, and that feature tag is not present anywhere in the contact predicate, that Reject-Contact predicate is discarded for the processing of that contact predicate. If the Reject-Contact predicate is not discarded, it is matched to the contact predicate using the matching operation of RFC 2533 [2]. If the result is a match, the URI corresponding to that contact predicate is discarded from the target set (and of course, its contact predicate is discarded as well).

The result is that Reject-Contact will only discard URIs where the UA has explicitly indicated support for the features that are not wanted.

Next, the proxy applies the predicates associated with the Require-Contact header field.

For each contact predicate that remains, each Require-Contact predicate is examined. The Require-Contact predicate is matched to the contact predicate using the matching operation of RFC 2533 [2]. If the result is not a match, the URI corresponding to that contact predicate is discarded from the target set, as is the contact predicate itself.

For each contact predicate that remains, each Accept-Contact predicate is examined. The Accept-Contact predicate is matched to the contact predicate using the matching operation of RFC 2533 [2]. If the result is a match, the URI associated with the contact predicate is considered a candidate URI. The set of Accept-Contact predicates which matched the contact predicate is called its matching set.

The q-value of URIs from the target set are then modified for this transaction only, in order to incorporate the caller's preferences. If the URI in the target set is not a candidate URI, its q-value is set to zero. If the URI is a candidate URI, its q-value is combined with those from the matching set. This document does not prescribe a specific algorithm for combining q-values. Among many possibilities, a server MAY set the q-value to the average of the original value specified in the registration, and the average q-value amongst the predicates in the matching set. This gives equal weight to caller and callee preferences. The only requirement for the combining process is that if a target URI has a q-value of q1, and the q values amongst the predicates in the matching set are q2,q3,..qn, the combined q value, qm, must satisfy:

```
MIN(q1,q2,q3,..qn) <= qm <= MAX(q1,q2,q3,..,qn)
```

> Note that this preference computation only determines the ordering of request attempts, so that the properties of the preference computation are of secondary importance. The q-value ordering provides only limited flexibility to indicate, for example, that a particular parameter is more important than another one or that combinations of two parameters should be weighed heavily.

If the server proxies, the target set is then sorted according to the updated q-value. Processing from this point depends on the configuration and policy of the server. If the server elects to do a sequential proxy, it SHOULD try the highest q-value contact entry first, trying addresses with decreasing q-values as each attempt

fails. If the server elects to do a parallel proxy, it SHOULD group contact entries with "close" q-values together, and try the group with the highest q-value first, then the group with the next lowest q-values, and so on. The precise method of the grouping is left to the implementor. A reasonable choice is to round each q-value to the nearest tenth, and group those with the same rounded value.

If a proxy server is recursing, it SHOULD apply the caller preferences to the Contact header fields returned in the redirect responses. Any target URI remaining after the application of caller preferences SHOULD be added to the list of untried addresses. This list is then resorted based on q values. The server uses this list for subsequent proxy operations.

If the server is redirecting, it SHOULD return all entries in the target set, including a q-value for each as obtained through the combining process. This SHOULD include any URI with a zero q-value.

If the server is executing any other type of policy, as a general guideline, it SHOULD prefer target URI with higher q values than those with lower q values.

8 Header Field Definitions

This specification defines four new header fields - Accept-Contact, Reject-Contact, Require-Contact and Request-Disposition.

Table 1 is an extension of Tables 2 and 3 in [1] for the Accept-Contact, Reject-Contact, Require-Contact and Request-Disposition header fields. The column "INF" is for the INFO method [8], "PRA" is for the PRACK method [9], "UPD" is for the UPDATE method [10], "SUB" is for the SUBSCRIBE method [7], and "NOT" is for the NOTIFY method [7].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG	PRA	UPD	SUB	NOT	INF
Accept-Contact	R	r	o	o	o	o	o	-	o	o	o	o	o
Reject-Contact	R	r	o	o	o	o	o	-	o	o	o	o	o
Require-Contact	R	r	o	o	o	o	o	-	o	o	o	o	o
Request-Disposition	R	r	o	o	o	o	o	o	o	o	o	o	o

Table 1: Accept-Contact, Reject-Contact, Require-Contact and Request-Disposition header fields

8.1 Request Disposition

The Request-Disposition header field specifies caller preferences for how a server should process a request. Its value is a list of tokens, each of which specifies a particular directive. Its syntax is specified in Section 10. Note that a compact form, using the letter d, has been defined. There can only be one value of a directive per header field (i.e., you can't have both "proxy" and "redirect" in the same Request-Disposition header field).

When the caller specifies a directive, the server SHOULD treat it as a hint, not as a requirement and MAY ignore the directive.

The directives have the following semantics:

proxy-directive: This directive indicates whether the caller would like each server to proxy or redirect. If the server is incapable of performing the requested directive, it SHOULD ignore it.

cancel-directive: This directive indicates whether the caller would like each proxy server to send a CANCEL request downstream in response to a 200 OK from the downstream server (which is the normal mode of operation, making it somewhat redundant), or whether this function should be left to the caller. If a proxy receives a request with this parameter set to "no-cancel", it SHOULD NOT CANCEL any outstanding branches on receipt of a 2xx. However, it would still send CANCEL on any outstanding branches on receipt of a 6xx.

fork-directive: This directive indicates whether a proxy should fork a request, or proxy to only a single address. If the server is requested not to fork, the server SHOULD proxy the request to the "best" address (generally the one with the highest q value). The directive is ignored if "redirect" has been requested.

recurse-directive: This directive indicates whether a proxy server receiving a 3xx response should send requests to the addresses listed in the response (i.e., recurse), or forward the list of addresses upstream towards the caller. The directive is ignored if "redirect" has been requested.

parallel-directive: For a forking proxy server, this directive indicates whether the caller would like the proxy server to proxy the request to all known addresses at once, or go through them sequentially, contacting the next address only after it has received a non-2xx or non-6xx final response for the previous one. The directive is ignored if

"redirect" has been requested.

queue-directive: If the called party is temporarily unreachable, e.g., because it is in another call, the caller can indicate that it wants to have its call queued rather than rejected immediately. If the call is queued, the server returns "182 Queued". A queued call can be terminated as described in [1].

Example:

```
Request-Disposition: proxy, recurse, parallel
```

The set of request disposition directives is purposefully not extensible. This is to avoid a proliferation of new extensions to SIP that are "tunnelled" through this header field.

8.2 Accept-Contact, Reject-Contact, and Require-Contact Header Fields

The syntax for these header fields is described in Section 10. A compact form, with the letter a, has been defined for the Accept-Contact header field, and with the letter j for the Reject-Contact header field.

The feature-tag is any valid feature tag, a number of which are applicable to SIP, and defined in Section 9. Note that string-value uses the qdtext production from RFC 3261. This production allows UTF-8 characters. This is in contrast to RFC 2533, which only allows ASCII characters in quoted strings. Usage of UTF-8 here is permissible since these values are never compared except using case sensitive matching rules.

8.3 Contact Header Field

This specification extends the Contact header field. In particular, it allows for the Contact header field parameter to include tag-set, whose BNF is described in Section 10. Tag-set is a set of feature parameters that describes the feature set of the UA associated with the URI in the Contact header field.

It is important to note that there is no way to differentiate, by syntax, Contact parameters that are part of tag-set or just other extensions. It turns out that this does not matter. If a proxy should mistakenly take a contact parameter used by another extension, and assume it is a feature parameter when its not, it will be ignored by

the matching algorithm unless the same parameter appears in the Accept-Contact or Reject-Contact header fields. However, it won't ever appear in these header fields, since those header fields only ever contain feature parameters, and the parameter is not actually a feature parameter.

9 Media Feature Tag Definitions

This specification defines an initial set of media feature tags for use with this specification. New media feature tags MAY be registered with IANA, based on the process defined for feature tag registrations [4]. This section also serves as the IANA registration for these feature tags.

Any registered feature tags MAY be used with this specification. However, several existing ones appear to be particularly applicable. These include the language feature tag [11], which can be used to specify the language of the human or automata represented by the UA, and the type feature tag [12], which can be used to specify the MIME types of the media formats supported by the UA. However, the usage of the audio, video, application, message, text and image feature tags (each of which indicate a top level media type supported by the UA) are preferred to indicating support for specific media formats. When the type feature tag is present, there SHOULD also be a feature tag present for the its top-level MIME type with a value of TRUE. In other words, if a UA indicates in a registration that it supports the video/H263 MIME type, it should also indicate that it supports video generally:

```
Contact: sip:1.2.3.4;type="video/H263";video="TRUE"
```

9.1 Attendant

Media feature tag name: attendant

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device is an automated or human attendant that will answer if the actual user of the device is not available.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a phone that has an auto-attendant feature.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.2 Audio

Media feature tag name: audio

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports audio as a MIME media type.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a phone that can support audio.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.3 Automata

Media feature tag name: automata

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The automata feature tag is a boolean value that indicates whether the UA represents an automata (such as a voicemail server, conference server, or recording device) or a human.

Values appropriate for use with this feature tag: Boolean. TRUE

indicates that the UA represents an automata.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing to communicate with a message recording device instead of a user.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.4 Class

Media feature tag name: class

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates the setting, business or personal, in which a communications device is used.

Values appropriate for use with this feature tag: Token with an equality relationship. Typical values include:

business: The device is used for business communications.

personal: The device is used for personal communications.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing between a business phone and a home phone.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.5 Duplex

Media feature tag name: duplex

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The duplex media feature tag lists whether a communications device can simultaneously send and receive media ("full"), alternate between sending and receiving ("half"), can only receive ("receive-only") or only send ("send-only").

Values appropriate for use with this feature tag: Token with an equality relationship. Typical values include:

full: The device can simultaneously send and receive media.

half: The device can alternate between sending and receiving media.

receive-only: The device can only receive media.

send-only: The device can only send media.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing to communicate with a broadcast server, as opposed to a regular phone, when making a call to hear an announcement.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.6 Image

Media feature tag name: image

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports image as a MIME media type.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a phone that can support image transfer.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.7 Message

Media feature tag name: message

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports message as a MIME media type.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a phone that can support messaging.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.8 Mobility

Media feature tag name: mobility

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The mobility feature tag indicates whether the device is fixed, wireless, or somewhere in-between.

Values appropriate for use with this feature tag: Token with an equality relationship. Typical values include:

fixed: The device is wired.

mobile: The device is wireless.

The feature tag is intended primarily for use in the following

applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing to communicate with a wireless phone instead of a desktop phone.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.9 Description

Media feature tag name: description

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The description feature tag provides a textual description of the device.

Values appropriate for use with this feature tag: String with an equality relationship.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Indicating that a device is of a certain make and model.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.10 Event Packages

Media feature tag name: events

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The event packages [7] supported by a SIP UA. The values for this tag equal the event package names that are registered by each event package.

Values appropriate for use with this feature tag: Token with an

equality relationship. Typical values include:

presence: SIP event package for for user presence [20].

winfo: SIP event package for watcher information [21].

refer: The SIP REFER event package [22].

dialog: The SIP dialog event package [23].

conference: The SIP conference event package [24].

reg: The SIP registration event package [25].

message-summary: The SIP message summary event package [26].

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing to communicate with a server that supports the message waiting event package, such as a voicemail server [26].

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.11 Priority

Media feature tag name: priority

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The priority feature tag indicates the call priorities the device is willing to handle.

Values appropriate for use with this feature tag: An integer. Each integral value corresponds to one of the possible values of the Priority header field as specified in SIP [1]. The mapping is defined as:

non-urgent: Integral value of 1. The device supports non-urgent calls.

normal: Integral value of 2. The device supports normal calls.

urgent: Integral value of 3. The device supports urgent calls.

emergency: Integral value of 4. The device supports emergency calls.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing to communicate with a the emergency cell phone of a user, instead of their regular phone.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.12 Methods

Media feature tag name: methods

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The methods (note the plurality) feature tag indicates the SIP methods supported by this UA. In this case, "supported" means that the UA can receive requests with this method. In that sense, it has the same connotation as the Allow header field.

Values appropriate for use with this feature tag: Token with an equality relationship. Typical values include:

INVITE: The SIP INVITE method [1].

ACK: The SIP ACK method [1].

BYE: The SIP BYE method [1].

CANCEL: The SIP CANCEL method [1].

OPTIONS: The SIP OPTIONS method [1].

REGISTER: The SIP REGISTER method [1].

INFO: The SIP INFO method [8].

UPDATE: The SIP UPDATE method [10].

SUBSCRIBE: The SIP SUBSCRIBE method [7].

NOTIFY: The SIP NOTIFY method [7].

PRACK: The SIP PRACK method [9].

MESSAGE: The SIP MESSAGE method [17].

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing to communicate with a presence application on a PC, instead of a PC phone application.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.13 Schemes

Media feature tag name: schemes

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The set of URI schemes [13] that are supported by a UA.

Values appropriate for use with this feature tag: Token with an equality relationship. Typical values include:

sip: The SIP URI scheme [1].

sips: The SIPS URI scheme [1].

tel: The tel URI scheme [6].

http: The HTTP URI scheme [14].

https: The HTTPS URI scheme [27].

cid: The CID URI scheme [15].

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Choosing get redirected to a phone number when a called party is busy, rather than a web page.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.14 Text

Media feature tag name: text

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports text as a MIME media type.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a phone that can support text.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.15 Video

Media feature tag name: video

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports video as a MIME media type.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Routing a call to a phone that can support video.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

9.16 Voicemail

Media feature tag name: voicemail

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device is a voicemail system which will record messages for a user.

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is most useful in a communications application, for describing the capabilities of a device, such as a phone or PDA.

Examples of typical use: Requesting that a call not be routed to voicemail.

Related standards or documents: RFC XXXX [[Note to IANA: Please replace XXXX with the RFC number of this specification.]]

10 Augmented BNF

```
Request-Disposition  =  ( "Request-Disposition" | "d" ) HCOLON
                        directive *(COMMA directive)
directive            =  proxy-directive / cancel-directive /
                        fork-directive / recurse-directive /
                        parallel-directive / queue-directive)
proxy-directive      =  "proxy" / "redirect"
```

```
cancel-directive     =  "cancel" / "no-cancel"
fork-directive       =  "fork" / "no-fork"
recurse-directive    =  "recurse" / "no-recurse"
parallel-directive   =  "parallel" / "sequential"
queue-directive      =  "queue" / "no-queue"
```

```
Accept-Contact   =  ("Accept-Contact" / "a") HCOLON feature-set
                    *(COMMA feature-set)
Reject-Contact   =  ("Reject-Contact" / "j") HCOLON feature-set-noq
                    *(COMMA feature-set-noq)
Require-Contact  =  "Require-Contact"
                    HCOLON feature-set-noq *(COMMA feature-set-noq)
feature-set      =  ( name-addr / addr-spec / "*")
                    *(SEMI tag-set) [q-param]
feature-set-noq  =  ( name-addr / addr-spec / "*")
                    *(SEMI tag-set)
tag-set          =  feature-tag EQUAL LDQUOT (tag-value-list
                    / string-value / boolean / numeric) RDQUOT
feature-tag      =  ftag ; From RFC 2533
tag-value-list   =  tag-value *("," tag-value)
tag-value        =  ["!"] token-nobang
token-nobang     =  1*(alphanum / "-" / "." / "%" / "*"
                    / "_" / "+" / "`" / "'" / "~" )
boolean          =  "TRUE" / "FALSE"
numeric          =  "#" (lessthan / greaterthan / equality /
                    range)
lessthan         =  ">=" number
greaterthan      =  "<=" number
equality         =  "=" number
range            =  "R" number ".." number
number           =  integer / rational
integer          =  [ "+" / "-" ] 1*DIGIT
rational         =  [ "+" / "-" ] 1*DIGIT "/" 1*DIGIT
string-value     =  LDQUOT "<" qdtext ">" RDQUOT
q-param          =  "q" EQUAL qvalue
```

```
contact-params  =  c-p-q / c-p-expires / tag-set
                =  / contact-extension
```

11 Mapping Feature Parameters and Feature Set Predicates

Mapping between feature parameters and feature set predicates, formatted according to the syntax of RFC 2533 [2] is trivial.

Starting from a set of feature parameters, the procedure is as follows. Construct a conjunction. Each term in the conjunction derives from one feature parameter. If the feature parameter value is a comma separated list, the element of the conjunction is a disjunction. There is one term in the disjunction for each value in the comma separated list. Call each value a "phrase". If the feature parameter value was not a comma separate list, the term in the conjunction is obtained from the value. That value is also a "phrase".

Consider now the construction of a filter from the phrase. If the phrase starts with a bang (!), the filter is of the form:

```
(! (name=remainder))
```

where name is the name of the feature parameter, and remainder is the remainder of the text in the phrase after the bang.

If the phrase starts with an octothorpe (#), the filter is a numeric comparison. The comparator is either =, >= or <= based on the next characters in the phrase. In this case, the filter is of the form:

```
(name comparator remainder)
```

where name is the name of the feature parameter, comparator is either =, >= or <=, and remainder is the remainder of the text in the phrase after the equal.

If the value after the octothorpe is R, the filter is a range. The format of the filter is:

```
(name=[remainder])
```

where name is the name of the feature parameter, and remainder is the remainder of the text in the phrase after the R. According to the BNF, this will be of the form "value..value", which specifies the range.

If the phrase begins with a left angle bracket ("<") and ends with a right angle bracket (">"), this implies that the value is a string, rather than a token. This is converted to a filter of the form:

```
(name="bracketed")
```

where name is the name of the feature parameter, and bracketed is the text from the phrase between the left and right angle brackets. Note the explicit usage of quotes, which indicate that the value is a string. In RFC 2533, strings are compared using case sensitive rules, and tokens, case insensitive.

> In RFC 2533, when an feature tag value is unquoted, its a token, and when quoted, its a string. The comparison rules are case insensitive for the latter, and sensitive for the former. The presence of quotes, or lack thereof, is the means by which an implementation can tell whether to apply sensitive or insensitive comparison rules. In the syntax described here, we cannot use quoted strings, since there is already a quoted string around each contact parameter value. So, we use an angle bracket to signify that the value is to be interpreted as a case sensitive string. If no brackets are present, the proxy would perform matching operations in a case insensitive manner, and if they are present, case sensitive.

Otherwise, the filter is of the following form:

```
(name=phrase)
```

where name is the name of the feature parameter, and phrase is the phrase.

As an example, the Contact header:

```
Contact:*;mobility="fixed";events="!presence,winfo";language="en,de"
 ;description="<PC>"
```

would be converted to the following feature predicate:

```
(& (mobility=fixed)
   (| (! (events=presence)) (events=winfo))
   (| (language=en) (language=de))
   (description="PC"))
```

As another example, the following Accept-Contact header field:

```
Accept-Contact: *;methods="SUBSCRIBE";resolution="#R5..100"
```

would be converted to the following feature set predicate:

```
(& (methods=SUBSCRIBE)
   (resolution=[5..100]))
```

The conversion of an RFC 2533 formatted feature set to a set of feature parameters proceeds in the same way, but in reverse. The conversion can only be done for feature sets constrained as described in Section 6.1.

12 Security Considerations

The presence of caller preferences in a request has a significant effect on the ways in which the request is handled at a server. As a result, is is especially important that requests with caller preferences be authenticated and integrity-protected. The same holds true for registrations with feature parameters in the Contact header field.

Processing of caller preferences requires set operations and searches

which can require some amount of computation. This enables a DOS attack whereby a user can send requests with substantial numbers of caller preferences, in the hopes of overloading the server. To counter this, servers SHOULD reject requests with too many rules. A reasonable number is around 20.

Feature sets contained in REGISTER requests can reveal sensitive information about a user or UA (for example, the languages spoken). If this information is sensitive, confidentiality SHOULD be provided by using S/MIME or the SIPS URI scheme, as described in RFC 3261 [1].

13 IANA Considerations

There are a number of IANA considerations associated with this specification.

13.1 Media Feature Tags

This specification registers a number of new Media feature tags according to the procedures of RFC 2506 [4]. Those registrations are contained in Section 9, and are meant to be placed into the IETF tree for media feature tags.

13.2 SIP Header Fields

This specification registers four new SIP header fields, according to the process of RFC 3261 [1].

The following is the registration for the Accept-Contact header field:

RFC Number: RFC XXXX [Note to IANA: Fill in with the RFC number of this specification.]

Header Field Name: Accept-Contact

Compact Form: a

The following is the registration for the Reject-Contact header field:

RFC Number: RFC XXXX [Note to IANA: Fill in with the RFC number of this specification.]

Header Field Name: Reject-Contact

Compact Form: j

The following is the registration for the Require-Contact header field:

RFC Number: RFC XXXX [Note to IANA: Fill in with the RFC number of this specification.]

Header Field Name: Require-Contact

Compact Form: none defined

The following is the registration for the Request-Disposition header field:

RFC Number: RFC XXXX [Note to IANA: Fill in with the RFC number of this specification.]

Header Field Name: Request-Disposition

Compact Form: d

13.3 SIP Option Tags

This specification registers a single SIP option tag, pref. The required information for this registration, as specified in RFC 3261, is:

Name: pref

Description: This option tag is used in a Proxy-Require header field by a UAC to ensure that caller preferences are honored at each proxy along the path. However, this usage is discouraged. It can also be used in the Require header field of a registration to ensure that the registrar supports the caller preferences extensions.

14 Acknowledgements

The initial set of media feature tags used by this specification were influenced by Scott Petrack's CMA design. Jonathan Lennox, and John Hearty provided helpful comments. Graham Klyne provided assistance on the usage of RFC 2533. Paul Kyzivat contributed significantly to this work, assisting in the generation of use cases, and poking holes in past versions of the document.

15 Author's Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

16 Normative References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002.

[2] G. Klyne, "A syntax for describing media feature sets," RFC 2533, Internet Engineering Task Force, Mar. 1999.

[3] S. Bradner, "Key words for use in RFCs to indicate requirement levels," RFC 2119, Internet Engineering Task Force, Mar. 1997.

[4] K. Holtman, A. Mutz, and T. Hardie, "Media feature tag registration procedure," RFC 2506, Internet Engineering Task Force, Mar. 1999.

[5] G. Klyne, "Corrections to "A syntax for describing media feature sets"," RFC 2738, Internet Engineering Task Force, Dec. 1999.

[6] A. Vaha-Sipila, "URLs for telephone calls," RFC 2806, Internet Engineering Task Force, Apr. 2000.

[7] A. B. Roach, "Session initiation protocol (sip)-specific event notification," RFC 3265, Internet Engineering Task Force, June 2002.

[8] S. Donovan, "The SIP INFO method," RFC 2976, Internet Engineering Task Force, Oct. 2000.

[9] J. Rosenberg and H. Schulzrinne, "Reliability of provisional responses in session initiation protocol (SIP)," RFC 3262, Internet Engineering Task Force, June 2002.

[10] J. Rosenberg, "The session initiation protocol (SIP) UPDATE method," RFC 3311, Internet Engineering Task Force, Oct. 2002.

[11] P. Hoffman, "Registration of charset and languages media features tags," RFC 2987, Internet Engineering Task Force, Nov. 2000.

[12] G. Klyne, "MIME content types in media feature expressions," RFC 2913, Internet Engineering Task Force, Sept. 2000.

[13] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax," RFC 2396, Internet Engineering Task Force, Aug. 1998.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol -- HTTP/1.1," RFC 2616, Internet Engineering Task Force, June 1999.

[15] E. Levinson, "Content-id and message-id uniform resource locators," RFC 2392, Internet Engineering Task Force, Aug. 1998.

17 Informative References

[16] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements," RFC 2824, Internet Engineering Task Force, May 2000.

[17] B. Campbell and J. Rosenberg, "Session initiation protocol extension for instant messaging," Internet Draft, Internet Engineering Task Force, Sept. 2002. Work in progress.

[18] G. Klyne, "Protocol-independent content negotiation framework," RFC 2703, Internet Engineering Task Force, Sept. 1999.

[19] M. Handley and V. Jacobson, "SDP: session description protocol," RFC 2327, Internet Engineering Task Force, Apr. 1998.

[20] J. Rosenberg, "Session initiation protocol (SIP) extensions for presence," Internet Draft, Internet Engineering Task Force, May 2002. Work in progress.

[21] J. Rosenberg, "A session initiation protocol (SIP)event template-package for watcher information," Internet Draft, Internet Engineering Task Force, May 2002. Work in progress.

[22] R. Sparks, "The SIP refer method," Internet Draft, Internet Engineering Task Force, July 2002. Work in progress.

[23] J. Rosenberg and H. Schulzrinne, "A session initiation protocol

(SIP) event package for dialog state," Internet Draft, Internet Engineering Task Force, June 2002. Work in progress.

[24] J. Rosenberg and H. Schulzrinne, "A session initiation protocol (SIP) event package for conference state," Internet Draft, Internet Engineering Task Force, June 2002. Work in progress.

[25] J. Rosenberg, "A sip event package for registration state," Internet Draft, Internet Engineering Task Force, Oct. 2002. Work in progress.

[26] R. Mahy, "A message summary and message waiting indication event package for the session initiation protocol (SIP)," Internet Draft, Internet Engineering Task Force, June 2002. Work in progress.

[27] E. Rescorla, "HTTP over TLS," RFC 2818, Internet Engineering Task Force, May 2000.

A Overview of RFC 2533

This section provides a brief overview of RFC 2533 and related specifications that form the content negotiation framework.

A critical concept in the framework is that of a feature set. A feature set is information about an entity (in our case, a UA), which describes a set of features it can handle. A feature set can be thought of as a region in N-dimensional space. Each dimension in this space is a different media feature, identified by a media feature tag. For example, one dimension (or axis) might represent languages, another might represent methods, and another, MIME types. A feature collection represents a single point in this space. It represents a particular rendering or instance of an entity (in our case, a UA). For example, a "rendering" of a UA would define an instantaneous mode of operation that it can support. One such rendering would be processing the INVITE method, which carried the application/sdp MIME type, sent to a UA for a user that is speaking English.

A feature set can therefore be defined as a set of feature collections. In other words, a feature set is a region of N-dimensional feature-space, that region being defined by the union of points - feature collections - that make up the space. If a particular feature collection is in the space, it means that the rendering described by that feature collection is supported by the device with that feature set.

How does one represent a feature set? There are many ways to describe an N-dimensional space. One way is to identify mathematical functions which identify its contours. Clearly, that is too complex to be

useful. The solution taken in RFC 2533 is to define the space with a feature set predicate. A feature set predicate is a boolean function over an N-dimensional space. The input to the function is a point in that space - a feature collection. If the result of the boolean function is TRUE, the feature collection is a member of the space. If the result of the boolean function is FALSE, the feature collection is not in the space.

RFC 2533 describes a syntax for writing down these N-dimensional boolean functions. It uses a prolog-style syntax which is fairly self-explanatory. This representation is called a feature set predicate. The base unit of the predicate is a filter, which is a boolean expression encased in round brackets. A filter can be complex, where it contains conjunctions and disjunctions of other filters, or it can be simple. A simple filter is one that expresses a comparison operation on a single media feature tag.

For example, consider the feature set predicate:

```
(& (foo=A)
   (bar=B)
   (| (baz=C) (& (baz=D) (bif=E))))
```

This defines a function over four media features - foo, bar, baz and bif. Any point in feature space with foo equal to A, bar equal to B, and either baz equal to C, or baz equal to D and bif equal to E, is in the feature set defined by this feature set predicate.

Note that the predicate doesn't say anything about the number of dimensions in feature space. The predicate operates on a feature space of any number of dimensions, but only those dimensions labeled foo, bar, baz and bif matter. The result is that values of other media features don't matter. The feature collection foo=A,bar=B,baz=C,bop=F is in the feature set described by the predicate, even though the media feature tag "bop" isn't mentioned. Feature set predicates are therefore inclusive by default. A feature collection is present unless the boolean predicate rules it out. This was a conscious design choice in RFC 2533.

RFC 2533 also talks about matching a preference with a capability set. This is accomplished by representing both with a feature set. A preference is a feature set - its a specification of a number of feature collections, any one of which would satisfy the requirements of the sender. A capability is also a feature set - its a

specification of the feature collections that the recipient supports. There is a match when the spaces defined by both feature sets overlap. When there is overlap, there exists at least one feature collection that exists in both feature sets, and therefore a modality or rendering desired by the sender which is supported by the recipient.

This leads directly to the definition of a match. Two feature sets match if there exists at least one feature collection present in both feature sets.

Computing a match for two general feature set predicates is not easy. Section 5 of RFC 2533 presents an algorithm for doing it by expanding an arbitrary expression into disjunctive normal form. However, the feature set predicates used by this specification are constrained. They are always in conjunctive normal form, with each term in the conjunction describing values for different media features. This makes computation of a match easy. It is computed independently for each media feature, and then the feature sets overlap if media features specified in both sets overlap. Computing the overlap of a single media feature is very straightforward, and is a simple matter of computing whether two finite sets overlap.

Full Copyright Statement