Internet Engineering Task Force
Internet Draft

SIP WG
J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia U.
P. Kyzivat
Cisco

draft-ietf-sip-callerprefs-08.txt
March 2, 2003
Expires: September 2003

Caller Preferences and Callee Capabilities for the Session Initiation Protocol (SIP)

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 Uniform Resource Identifiers (URI) a request gets routed to, and to specify certain request handling directives in proxies and redirect servers. It does so by defining three new request header fields, Accept-Contact, Reject-Contact, and Request-Disposition, which specify the caller's preferences. The extension also defines new parameters for the Contact header field that describe the

capabilities and characteristics of a User Agent (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) [22]. 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 caller 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 provide specific request handling directives for a server. The other, called feature preferences, are present in the Accept-Contact and Reject-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 field 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 [3].

One example of the a service enabled by caller preferences is a "one number" service. A user can have a single identity (their SIP URI) for all of their devices - their cell phone, PDA, work phone, home phone, and so on. If the caller wants to reach the user at their business phone, they simply select "business phone" from a pull-down menu of options when calling that URI. Users would no longer need to maintain and distribute separate identities for each device.

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 [4] 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 [23], 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 [23], a feature tag is a name that identifies a feature. An example is "methods".

Media Feature: As defined in RFC 2703, [23], 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 [23], a feature set is Information about a sender, recipient 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. Feature preferences can be made explicitly with the Accept-Contact and Reject-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 and Reject-Contact header fields. The feature parameters represent an encoding of a feature set. Each set of feature parameters maps to a feature set predicate.

Capability: As defined in RFC 2703 [23], 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.

Target Set: A target set is a set of candidate URI that a proxy or redirect server can send or redirect a request to. Frequently, target sets are obtained from a registration, but they need not be.

Explicit Preference: A caller preference indicated explicitly in the Accept-Contact or Reject-Contact header fields.

Implicit Preference: A caller preference that is implied through the presence of other aspects of a request. For example, if

the request method is INVITE, it represents an implicit caller preference to route the request to a UA that supports the INVITE method.

Filter: A single expression in a feature set 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.

Conjunction: A boolean AND 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 an encoded feature tag, as defined in RFC 2703 [23], 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 a 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 "msgserver" 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 and Reject-Contact header fields. These header fields also contain feature sets, represented by the same feature parameters that are used in the Contact header field. 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.

Proxies use the information in the Accept-Contact and Reject-Contact header fields to select amongst contacts in their target set. When neither of those header fields are present, the proxy computes 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 [5] 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 [6], which provides a minor update to RFC 2533, and RFC 2703 [23] 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 control. 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 five separate cases. The first is registration, where a UA can declare its capabilities. The second is expression of preferences in a request, 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. The fifth is UAS processing of an OPTIONS request.

6.1 Expressing Capabilities in a Registration

When a UA registers, it can choose to indicate a feature set associated with a registered contact. Whether or not a UA does so depends on what the registered URI represents. If the registered URI represents a UA instance (the common case in registrations), a UA compliant to this specification SHOULD indicate a feature set using the mechanisms described here. If, however, the registered URI represents an address-of-record, or some other resource that is not representable by a single feature set, it SHOULD NOT include a feature set. As an example, if a user wishes to forward calls from sip:user1@example.com to sip:user2@example.org, it could generate a registration that looks like, in part:

```
REGISTER sip:example.com SIP/2.0
To: sip:user1@example.com
Contact: sip:user2@example.org
```

In this case, the registered contact is not identifying a UA, but rather, another address-of-record. In such a case, the registered contact would not indicate a feature set.

If a UA does not include feature parameters for a contact, that contact will be immune from the caller preference processing. Therefore, if a registering client does not want caller preferences applied to a contact, it omits all feature parameters. Addresses-of-record in particular often need to be immune from caller preferences processing. If they were not, such a URI might be eliminated from consideration, even though a downstream UA satisfies the desired constraints.

However, in some cases a UA may wish to express feature parameters for an address-of-record. One example is an AOR which represents a mutliplicity of devices in a home network, and routes to a proxy server in the user's home. Since all devices in the home are for personal use, the AOR itself can be described with the "class=personal" feature parameter. A registration that forwards calls to this home AOR could make use of that feature parameter. Generally speaking, a feature parameter can only be associated with an address-of-record if all devices bound to that address-of-record share the exact same set of values for that feature parameter.

The remainder of this section assumes that a UA would like to associate a feature set with a contact that it is registering. To do that, it constructs a feature predicate for that contact. In the text that follows, this process is described in terms of RFC 2533 [2] (and its minor update, [6]) 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 (called a conjunction). 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, range, or negation of 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 [5]; this document registers several that are appropriate for SIP. It is also permissible to use the URI tree [5] for expressing vendor-specific feature tags. Feature tags in any other trees created through IANA MAY also be used.

A UA SHOULD include the "uri-user" and "uri-domain" feature tag in its feature parameters. The value of those tags SHOULD be equal to the user and domain part of the registered URI, respectively. Setting them differently is likely to result in odd behavior, and should only be done if some unforseen service neccesitates it. Note that the "uri-user" feature tag is a quoted string (implying case sensitive matching), and the "uri-domain" feature tag is a token, implying case

insensitive matching.

Note that the "schemes" feature tag is not a peer of the "uri-user" and "uri-domain" feature tags. That is, it does not indicate the scheme of the registered URI. Rather, it indicates schemes that a UA is capable of sending requests to, should such a URI be received in a web page or Contact header field of a redirect response.

It is RECOMMENDED that a UA provide complete information in its feature predicate. 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. 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 omission of a value from a list means that the value is not supported.

When using the "methods" feature tag, a UA MUST NOT include values that correspond to methods not standardized in IETF standards track RFCs. When using the "events" feature tag, a UA MUST NOT include values that correspond to event packages not standardized in IETF standards track RFCs. When using the "schemes" feature tag, a UA MUST NOT include values that correspond to schemes not standardized in IETF standards track RFCs. When using the "sip-extensions" feature tag, a UA MUST NOT include values that correspond to option tags not standardized in IETF standards track RFCs.

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)
   (msgserver=TRUE)
   (automata=TRUE)
   (attendant=TRUE)
   (mobility=fixed)
   (| (methods=INVITE) (methods=BYE) (methods=OPTIONS) (methods=ACK)
      (methods=CANCEL))
```

```
    (uri-user="user")
    (uri-domain=host.example.com)
```

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"
  ;msgserver="TRUE";automata;attendant;mobility="fixed"
  ;methods="INVITE,BYE,OPTIONS,ACK,CANCEL"
  ;uri-user="<user>"
  ;uri-domain="host.example.com"
Content-Length: 0
```

Note that a voicemail server is usually an automata and an attendant, as defined below.

6.2 Expressing Preferences in a Request

A caller wishing to express preferences for a request includes Accept-Contact, Reject-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 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 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 and Reject-Contact header field values. Each header field value contains a set of 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. 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.

A UAC can express explicit preferences for the methods and event packages supported by a UA. It is RECOMMENDED that a UA include a term in an Accept-Contact feature set with the "methods" feature tag, whose value includes the method of the request. When a UA sends a SUBSCRIBE request, it is RECOMMENDED that a UA include a term in an Accept-Contact feature set with the "events" feature tag, whose value includes the event package of the request. Whether these terms are placed into a new feature set, or whether they are included in each feature set, is at the discretion of the implementor. In most cases, the right effect is achieved by including a term in each feature set.

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 "*", purely to align the syntax with guidelines for SIP extensions [24], and is parameterized by a set of feature parameters. Any UA whose capabilities match the feature set described by the feature parameters matches the value. 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. Each value of the Accept-Contact header field contains a "*" and is parameterized by a set of feature parameters. Any UA whose capabilities match the feature set described by the feature parameters matches the value. The q-value parameter provides a weighting operation. A q-value parameter with a particular value means that the caller's preference for a UA described by the feature parameters equals that value. The processing rules at a proxy will also favor those UA that are a "better" match to a particular value. Here, better means that more of its capabilities explicitly match the feature preferences. The value may also contain an "explicit" parameter, which indicates that only UA whose capabilities explicitly match are considered a match. If one of the values contains the "require" parameter, it means that the UA must match that value. 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 requests, 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 Processing Request Handling and Feature Set Preferences

When a UAS compliant to this specification receives a request whose request-URI corresponds 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.

This processing occurs after the client authenticates and authorizes the request, but before the remainder of the general UAS processing described in Section 8.2.1 of RFC 3261.

If a UA registers against two separate addresses-of-record, and the contacts registered for each have different capabilities, a UA MUST use different URIs in each registration. This is so that the UA can uniquely determine the feature set that is associated with the request URI of an incoming request.

If, after performing this processing, there are no URI left in the target set, the UA SHOULD reject the request with a 480 response. If there is a URI remaining (there was only one to begin with), the UA proceeeds with request processing as per RFC 3261.

> Having a UAS perform the matching operations as if it were a proxy allows certain caller preferences to be honored even if the proxy doesn't support the extension.

6.5 OPTIONS Processing

When a UAS compliant to this specification receives an OPTIONS request, it MAY add feature parameters to the Contact header field in the OPTIONS response 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 OPTIONS response. Indeed, if feature parameters were included in the registration generated by that UA, those same parameters SHOULD be used in the OPTIONS response.

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 and Reject-Contact header fields.

In addition to processing these headers, a proxy MAY add one if not present, or add a value to an existing header field, as if it were a UAC. This is useful for a proxy to request processing in downstream proxies in the implementation of a feature. However a proxy MUST NOT modify or remove an existing header field or header field value. This is particularly important when S/MIME is used. The message signature could include the caller preferences header fields, allowing the UAS to verify that, even though proxies may have added header fields, the original caller preferences were still present.

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 with 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. If it does perform the processing, it MUST do so as described below.

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 and Reject-Contact header fields.

For each value of those header fields, it extracts the feature parameters. These are the header field parameters whose name is one of the base-tags (see Section 10), or whose name begins with a plus (+). The proxy converts all of those parameters to the syntax of RFC 2533, based on the rules in Section 11.

The result will be a set of feature set predicates in conjunctive normal form, each of which is associated with one of the two 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. If there was a req-parameter associated with a header field value in the Accept-Contact header field, the feature set predicate derived from that header field value is said to have its require flag set. Similarly, if there was an explicit-param associated with a header field value in the Accept-Contact header field, the feature set predicate derived from that header field value is said to have its explicit flag set.

7.2.2 Extracting Implicit Preferences

If, and only if, the proxy did not find any explicit preferences in the request (because there was no Accept-Contact or Reject-Contact header field), the proxy extracts implicit preferences. These preferences are ones implied by the presence of other information in the request.

First, the proxy creates a conjunction with no terms. This conjunction represents a feature set that will be associated with the Accept-Contact header field, as if it were included there. Note that there is no modification of the message implied - only an association for the purposes of processing. Furthermore, this feature set has its require flag set, but not its explicit flag.

The proxy then adds terms to the conjunction for the two implicit preference types below.

7.2.2.1 Methods

One 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 adds a term to the conjunction of the following form:

(methods=[method of request])

7.2.2.2 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 support this implicit preference, the proxy adds a term to the conjunction of the following form:

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

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 the target URI was obtained through a registration, the proxy computes the contact predicate by extracting the feature parameters from the Contact header field and the converting them to a feature predicate. To extract the feature parameters, the proxy follows these steps:

1. Create an initial, empty list of feature parameters.

2. If the Contact URI parameters included the "attendant", "audio", "automata", "class", "duplex", "data", "control", "mobility", "description", "events", "priority", "methods", "schemes", "application", "video", "msgserver", "language", "isfocus", "uri-user", "uri-domain" or "type" parameters, those are copied into the list.

3. If any Contact URI parameter name begins with a "+", it is copied into the list if the list does not already contain that name with the plus removed. In other words, if the "video" feature parameter is in the list, the "+video" parameter would not be placed into the list. This conflict should never arise if the client were compliant to this specification, since it is illegal to use the + form for encoding of a feature tag in the base set.

If the URI in the target set had no feature parameters, it is said to

be immune to caller preference processing. This means that the URI is removed from the target set temporarily, the caller preferences processing described below is executed, and then the URI is added back in.

Assuming the URI has feature parameters, they are converted to RFC 2533 syntax using the rules of Section 11.

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, consider the following registered Contact header field:

```
Contact: <sip:user@example.com>;audio;video;mobility="fixed";
  +message="TRUE";other-param=66372;
  methods="INVITE,OPTIONS,BYE,CANCEL,ACK";schemes="sip,http";
  uri-user="<user>";uri-domain="example.com"
```

This would be converted into the following predicate:

```
(& (audio=TRUE)
   (video=TRUE)
   (mobility=fixed)
   (message=TRUE)
   (| (methods=INVITE) (methods=OPTIONS) (methods=BYE)
      (methods=CANCEL) (methods=ACK))
   (| (schemes=sip) (schemes=http))
   (uri-user="user")
   (uri-domain="example.com"))
```

Note that "other-param" was not considered a featuer parameter, since it is neither a base tag nor did it begin with a leading +.

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 an implementation to know whether the feature tag is a numeric, token, or quoted string (booleans can be treated as tokens). Quoted strings are always matched using a case-sensitive matching operation. Tokens are matched using case-insensitive matching. Numerics are matched using normal mathematical comparisons.

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.

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 Accept-Contact header field. For each contact that remains in the target set, the proxy constructs a matching set, Ms. Initially, this set contains all of the Accept-Contact predicates. Each of those predicates is examined. It is matched to the contact predicate using the matching operation of RFC 2533 [2]. If the result is not a match, and the Accept-Contact predicate had its require flag set, the URI corresponding to that contact predicate is discarded from the contact set. If the result is not a match, but the Accept-Contact predicate did not have its require flag set, that contact URI is not discarded from the contact set, however, the Accept-Contact predicate is

removed from the matching set for that contact.

For each contact that remains in the target set, the proxy computes a score for that contact against each predicate the contact's matching set. Let the number of terms in the Accept-Contact predicate conjunction be equal to N. Each term in that predicate contains a single feature tag. If the contact predicate has a term containing that same feature tag, the score is incremented by 1/N. If the feature tag was not present in the contact predicate, the score remains unchanged. Based on these rules, the score can range between zero and one.

The require and explicit tags are then applied, resulting in potential modification of the score and the target set. This process is summarized in Figure 1. If the score for the contact predicate against that Accept-Contact predicate was less than one, and the Accept-Contact predicate had an explicit tag, if the predicate also had a require tag, the Contact URI corresponding to that contact predicate is dropped. If, however, the predicate did not have a require tag, the score is set to zero. If there was no explicit tag, the score is unchanged.

The next step is to combine the scores and the q-values associated with the predicates in the matching set, to arrive at an overall caller preference, Qa. For those URIs in the target set which remain, there will be a score which indicates its match against each Accept-Contact predicate in the matching set. If there are M Accept-Contact predicates in the matching set, there will be M scores S1 through SM, for each contact. There will also be a preference associated with each Accept-Contact predicate (derived from the q-value parameter, as discussed in Section 7.2.1), X1..XM. The caller preference, Qa, is computed as shown in Figure 2.

Note that in the limit as all Si go to zero, Qa equals the arithmetic average of Xi.

This algorithm was chosen carefully so as to exhibit certain properties:

- If Si is 1 for i=j, and zero for all other i, Qa=Xi. In other words, if a contact predicate matches one of the Accept-Contact predicates with a score of one (referred to as an explicit match), and all others match with a score of zero (referred to as an implicit match), the caller's preference equals the q-value of that predicate.

- If Si is the same for all predicates in the matching set, Qa is equal to the average of the q-values for the predicates.

- If the contact predicate matches only one Accept-Contact predicate, Qa is equal to the q-value of that predicate, independent of the score.

The final step is to combine the overall caller preference for the contact (Qa) with the q-value provided for that contact by the callee (which we denote as Qb). The proxy can use any averaging mechanism at its disposal, prefentially treating the callers preference and the callee's preference as policy dictates. In the absence of policy indicating otherwise, the two values are arithmetically averaged. This results in an overall q-value for that contact, Qo, equal to:

$$Qo = \frac{Qa + Qb}{2}$$

At this point, any URI that were removed from the target set because they were immune from caller preferences are added back in, and Qo for that URI is set to its original q-value, or 1.0 if there was no q-value specified.

If there were no URIs in the target set after the application of the processing in this section, and the caller preferences were based on implicit preferences (Section 7.2.2), the processing in this section is discarded, and the original target set, along with their original q-values, is used.

This handles the case where implicit preferences for the method or event packages resulted in the elimination of all potential targets. By going back to the original target set, those URIs will be tried, and result in the generation of a 405 or 489. The UAC can then use this information to try again, or report the error to the user. Without reverting to the original target set, the UAC would see a 480 response, and have no knowledge of why their request failed. Of course, the target set can also be empty after the application of explicit preferences. This will result in the generation of a 480 by the proxy. This behavior is acceptable, and indeed, desirable in the case of explicit preferences. When the caller makes an explicit preference,

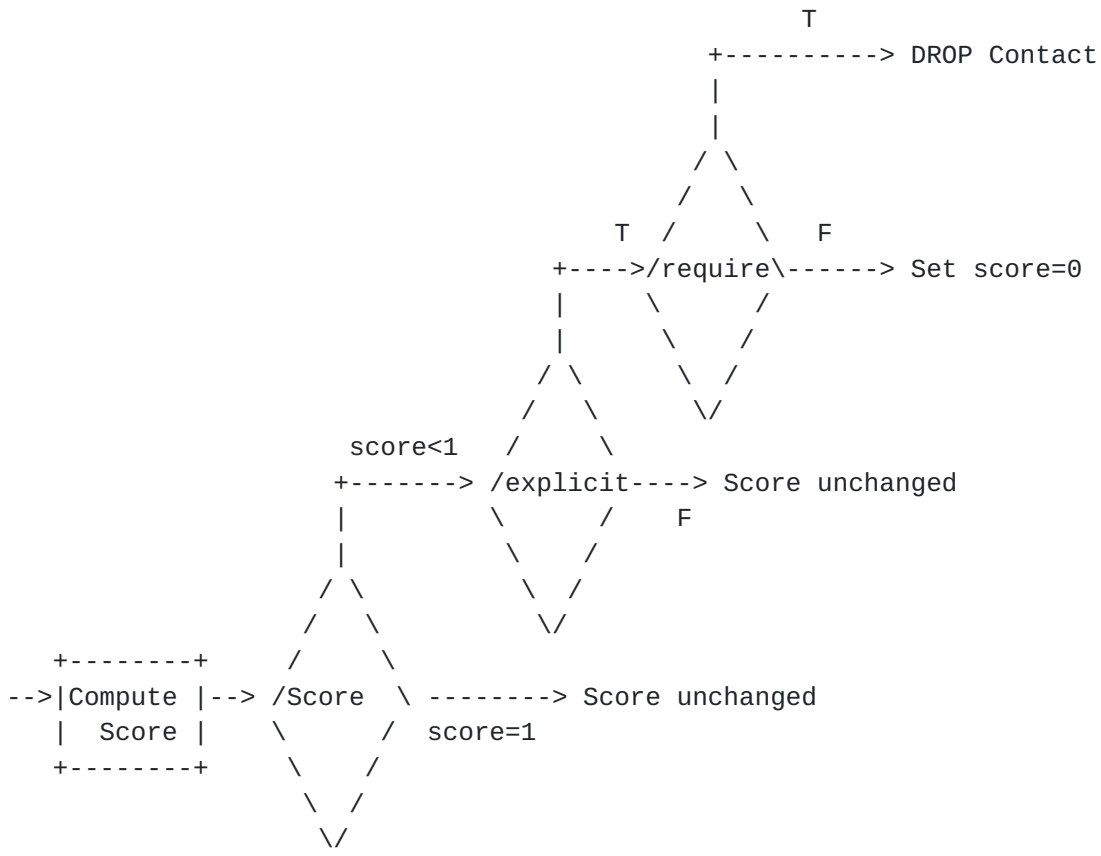
T
DROP Contact
T
F
/require\
Set score=0
score<1
/explicit
Score unchanged
F
+--------+
-->|Compute |--> /Score \ --------> Score unchanged
| Score |
+--------+
score=1

Figure 1: Score Computation

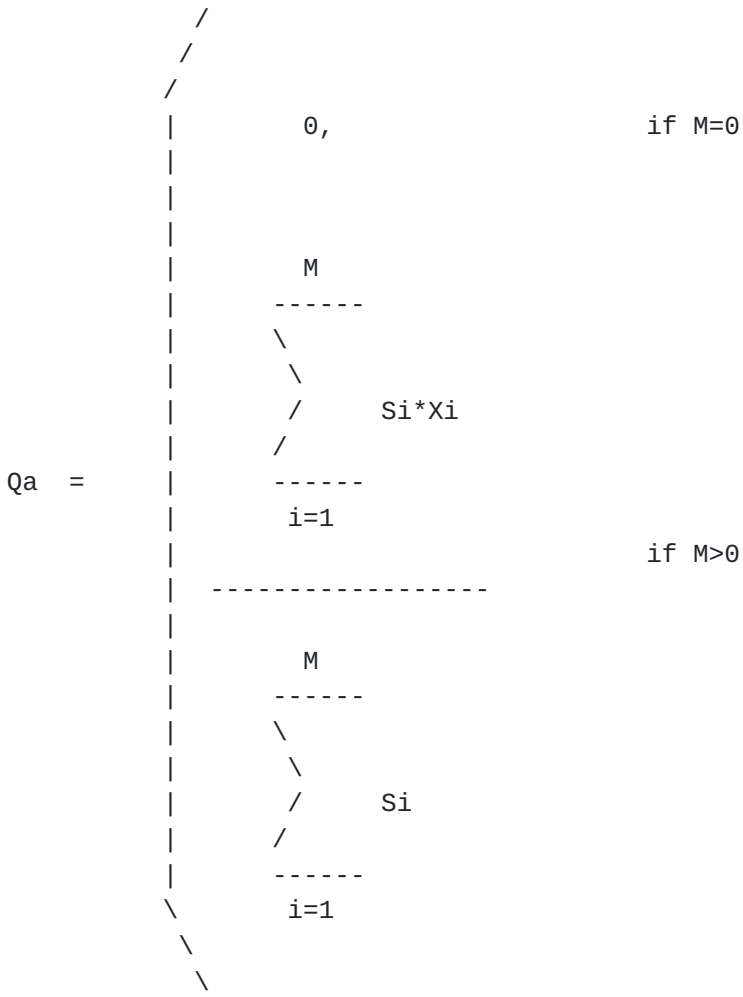

Figure 2: Computation of Qa

it is agreeing that its request might fail because of a preference mismatch. One might try to return an error indicating the capabilities of the callee, so that the caller could perhaps try again. However, doing so results in the leaking of potentially sensitive information to the caller without authorization from the callee, and therefore this specification does not provide a means for it.

Any proxy processing that takes the q-values as inputs (for example, a forking operation as described in Section 16.6 of RFC 3261 [1]) would use Qo instead of the original q-value associated with the contact, for this specific transaction only. To avoid preferring one

contact to another because of a relatively small difference in their overall q-value, it is RECOMMENDED that the values be rounded to the nearest tenth before they are used by the proxy.

If a proxy server is recursing, it applies the caller preferences to the Contact header fields returned in the redirect responses. Any URI remaining after the application of caller preferences are added to the proxy's target set if it is not already in the target set. This list is then resorted based on q values. The server uses this list for subsequent proxy operations.

If the server is redirecting, it returns all entries in the target set, including a q-value of Qo for each Contact URI as obtained through the process above. This includes any URI with a zero q-value. However, it MUST NOT include the feature parameters for the entries in the target set. If it did, the upstream proxy server would apply the same caller preferences once more, resulting in a double application of those preferences. If the redirect server does wish to include the feature parameters in the Contact header field, it MUST redirect using the original target set and original q-values, before the application of caller preferences.

It is the usage of these modified q-values that allows the caller preferences to be taken into account, while at the same time giving the proxy flexibility in how it processes the request.

7.4.1 Example

Consider the following example, which is contrived but illustrative of the various components of the matching process. There are five registered Contacts for sip:user@example.com. They are:

```
Contact: sip:u1@h.example.com;audio;video;methods="INVITE,BYE";q=0.1
Contact: sip:u2@h.example.com;audio="FALSE";
  methods="INVITE";msgserver;q=0.2
Contact: sip:u3@h.example.com;audio;msgserver;
  methods="INVITE";video;q=0.3
Contact: sip:u4@h.example.com;audio;methods="INVITE,OPTIONS";q=0.4
Contact: sip:u5@h.example.com;q=0.5
```

an INVITE sent to sip:user@example.com contained the following caller preferences header fields:

```
Reject-Contact: *;msgserver;video
Accept-Contact: *;audio;require;q=0.5, *;video;explicit;q=0.4,
  *;methods="BYE";class="business";q=1.0
```

There are no implicit preferences in this example, because explicit preferences are provided.

The proxy first removes u5 from the target set, since it is immune from caller preferences processing.

Next, the proxy processes the Reject-Contact header field. It is a match for all four remaining contacts, but only an explicit match for u3. Thats because u3 is the only one that explicitly indicated support for video, and explicitly indicated it is a messaging server. So, u3 gets discarded, and the others remain.

Next, each of the remaining three contacts is compared against each of the three Accept-Contact predicates. u1 is a match to all three, earning a score of 1.0 for the first two predicates, and 0.5 for the third (the methods feature tag was present in the contact predicate, but the class tag was not). u2 doesn't match the first predicate. Because that predicate has a require tag, u2 is discarded. u4 matches the first predicate, earning a score of 1.0. u4 does match the second predicate, but since the match is not explicit (the score is 0.0, in fact), the score is set to zero (it was already zero, so nothing changes). u4 does not match the third predicate.

At this point, u1 and u4 remain. u1 matched all three Accept-Contact predicates, so that its matching set contains all three, with scores of 1, 1, and 0.5. u4 matches the first two predicates, with scores of 1.0 and 0.0.

Qa for u1 is then computed as:

$$\frac{1.0*0.5 + 1.0*0.4 + 0.5*1.0}{1.0 + 1.0 + 0.5} = 0.56$$

Qa for u4 is then computed as:

$$\frac{1.0*0.5 + 0.0*0.4}{1.0 + 0.0} = 0.5$$

Qo for u1 is the average of 0.56 and the registered q-value of 0.1, which equals 0.33. Qo for u4 is the average of 0.5 and the registered q-value of 0.4, which equals 0.45. Rounding these to the nearest tenth, Qo for u1 is 0.3 and Qo for u4 is 0.5.

Now, u5 is added back in. It retains its original q-value of 0.5. Since its q-value matches that of u4, both u4 and u5 would be tried in parallel. Should both fail, u1 would be tried.

8 Header Field Definitions

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

Tables 1 and 2 are an extension of Tables 2 and 3 in [1] for the Accept-Contact, Reject-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], "NOT" is for the NOTIFY method [7], and "MSG" is for the MESSAGE method [3].

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept-Contact	R	ar	o	o	o	o	o	-
Reject-Contact	R	ar	o	o	o	o	o	-
Request-Disposition	R	ar	o	o	o	o	o	o

Table 1: Accept-Contact, Reject-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. The directives are grouped into types. There can only be one directive of each type per request (i.e., you can't have both "proxy" and "redirect" in the same Request-Disposition header

Header field	where	proxy	PRA	UPD	SUB	NOT	INF	MSG
Accept-Contact	R	ar	o	o	o	o	o	o
Reject-Contact	R	ar	o	o	o	o	o	o
Request-Disposition	R	ar	o	o	o	o	o	o

Table 2: Accept-Contact, Reject-Contact, and Request-Disposition header fields

field).

When the caller specifies a directive, the server SHOULD honor that directive.

The following types of directives are defined:

proxy-directive: This type of directive indicates whether the caller would like each server to proxy ("proxy") or redirect ("redirect").

cancel-directive: This type of directive indicates whether the caller would like each proxy server to send a CANCEL request downstream ("cancel") 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 ("no-cancel"). 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 type of directive indicates whether a proxy should fork a request ("fork"), or proxy to only a single address ("no-fork"). 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 type of directive indicates whether a proxy server receiving a 3xx response should send requests to the addresses listed in the response ("recurse"), or forward the list of addresses upstream towards the caller ("no-recurse"). The directive is ignored if "redirect" has been requested.

parallel-directive: For a forking proxy server, this type of directive indicates whether the caller would like the proxy

server to proxy the request to all known addresses at once ("parallel"), 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 ("sequential"). 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 ("queue") or rejected immediately ("no-queue"). 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 "tunneled" through this header field.

8.2 Accept-Contact and Reject-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 enc-feature-tag is an encoded version of 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 parameters to include feature-param, whose BNF is described in Section 10. Feature-param is a feature parameter that describes a feature of the UA associated with the URI in the Contact header field. Feature parameters are identifiable because they either belong to the well known set of base feature tags, or they begin with a plus sign.

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 [5]. 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, data and control feature tags (each of which indicate a media type, as defined in RFC 2327 [13] 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:192.0.2.1;type="video/H263";video="TRUE"
```

If a new SDP media type were to be defined, such as "message", a new feature tag registration SHOULD be created for it. The name of the feature tag MUST equal that of the media type, unless there is an unlikely naming collision between the new media type and an existing feature tag registration. As a result of this, implementations can safely construct caller preferences and callee capabilities for the new media type before it is registered, as long as there is no naming conflict.

If a new media feature tag is registered with the intent of using that tag with this specification, the registration is done for the unencoded form of the tag (see Section 11). In other words, if a new feature tag "foo" is registered, the IANA registration would be for the tag "foo" and not "+foo". When that parameter is used within the Contact, Accept-Contact and Reject-Contact header fields, it would be encoded using its + form.

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 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 Application

Media feature tag name: application

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports application as a media type. This feature tag exists primarily for completeness. Since so many MIME types are underneath application, indicating the ability to support applications provides little useful information. In most cases, the concrete MIME type is a better parameter to use in a predicate representing a preference.

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 supports gaming application.

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

9.4 Data

Media feature tag name: data

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports data as a 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 supports a data streaming application.

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

9.5 Control

Media feature tag name: control

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports control as a 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 supports a floor control application.

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

9.6 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.]]

9.7 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.]]

9.8 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.]]

9.9 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 stationary.

mobile: The device can move around with the user.

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.]]

9.10 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.]]

9.11 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 [25].

winfo: SIP event package for watcher information [26].

refer: The SIP REFER event package [27].

dialog: The SIP dialog event package [28].

conference: The SIP conference event package [29].

reg: The SIP registration event package [30].

message-summary: The SIP message summary event package [31].

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 [31].

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

9.12 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. A value of X means that the device is willing to take requests with priority X and higher.

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 10. The device supports non-urgent calls.

normal: Integral value of 20. The device supports normal calls.

urgent: Integral value of 30. The device supports urgent calls.

emergency: Integral value of 40. 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 the emergency cell phone of a user.

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

9.13 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. 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 [3].

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.]]

9.14 SIP Extensions

Media feature tag name: sip-extensions

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The sip-extensions feature tag is a list of SIP extensions (each of which is defined by an option-tag registered with IANA) that are understood by the UA. Understood, in this context, means that the option tag would be included in a Supported header field in a request.

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

100rel: The UA supports reliability of provisional responses [9].

path: The UA supports the SIP Path header field [14].

precondition: The UA supports the preconditions mechanism described in RFC 3312 [15].

privacy: The UA supports the privacy extension described in RFC 3323 [16].

sec-agree: The UA supports the security agreement extension described in RFC 3329 [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 phone that supports quality of service preconditions instead of one that does not.

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

9.15 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 [18] that are supported by a UA. Supported implies, for example, that the UA would know how to handle a URI of that scheme in the Contact header field of a redirect response.

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 [19].

http: The HTTP URI scheme [20].

https: The HTTPS URI scheme [32].

cid: The CID URI scheme [21].

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.]]

9.16 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 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.]]

9.17 Message Server

Media feature tag name: msgserver

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 messaging server which will record messages for a user. An example of such a device is a voicemail server.

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.]]

9.18 Is Focus

Media feature tag name: isfocus

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: This feature tag indicates that the UA is a conference server, also known as a focus, and will mix together the media for all calls to the same URI [33].

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: Indicating to a UA that the server it has connected to is a conference server.

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

9.19 URI User

Media feature tag name: uri-user

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The uri-user feature tag provides the user part of the SIP URI that represents 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: Requesting to route a call to a

specific device, identified by a URI.

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

9.20 URI Domain

Media feature tag name: uri-domain

ASN.1 Identifier: New assignment by IANA.

Summary of the media feature indicated by this tag: The uri-domain feature tag indicates the hostname of a device.

Values appropriate for use with this feature tag: Token with a case-insensitive 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: Requesting to route a call to a specific device, identified by a URI.

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 ac-value
                     *(COMMA ac-value)
Reject-Contact    =  ("Reject-Contact" / "j") HCOLON rc-value
                     *(COMMA rc-value)
ac-value          =  "*" *(SEMI ac-params)
rc-value          =  "*" *(SEMI rc-params)
ac-params         =  feature-param / c-p-q / req-param
                     / explicit-param / generic-param
rc-params         =  feature-param / req-param
                     / explicit-param / generic-param
feature-param     =  enc-feature-tag [EQUAL LDQUOT (tag-value-list
                     / string-value ) RDQUOT]
enc-feature-tag   =  base-tags / other-tags
base-tags         =  "attendant" / "audio" / "automata" /
                     "class" / "duplex" / "data" /
                     "control" / "mobility" / "description" /
                     "events" / "priority" / "methods" /
                     "schemes" / "application" / "video" /
                     "msgserver" / "language" / "type" /
                     "isfocus" / "uri-user" / "uri-domain"
other-tags        =  "+" ftag-name
ftag-name         =  ALPHA *( ALPHA / DIGIT / "!" / "'" /
                     "." / "-" / "%" )
tag-value-list    =  tag-value *("," tag-value)
tag-value         =  ["!"] (token-nobang / boolean / numeric)
token-nobang      =  1*(alphanum / "-" / "." / "%" / "*"
                     / "_" / "+" / "`" / "'" / "~" )
boolean           =  "TRUE" / "FALSE"
numeric           =  "#" numeric-relation number
numeric-relation  =  ">=" / "<=" / "=" / (number ":")
number            =  [ "+" / "-" ] 1*DIGIT ["." 0*DIGIT]
string-value      =  "<" qdtext ">"
req-param         =  "require"
explicit-param    =  "explicit"
```

Note that the tag-value-list uses an actual comma instead of the COMMA construction. Thats because it appears within a quoted string, where line folding cannot take place.

The productions for c-p-q, name-addr, addr-spec, qdtext and generic-param can be found in RFC 3261 [1].

Despite the BNF, there MUST NOT be more than one c-p-q, req-param or explicit-param in an ac-params or rc-params. Furthermore, there can only be one instance of any feature tag in feature-param.

Any numbers present in a feature parameter MUST be representable

using an ANSI C double.

The following production updates the one in RFC 3261 for contact-params:

```
contact-params  =  c-p-q / c-p-expires / feature-param
                   / contact-extension
```

11 Mapping Feature Parameters and Feature Set Predicates

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

Starting from a set of feature-param, the procedure is as follows. Construct a conjunction. Each term in the conjunction derives from one feature-param. If the feature-param has no value, it is equivalent, in terms of the processing which follows, as if it had a value of "TRUE".

If the feature-param value is a tag-value-list, the element of the conjunction is a disjunction. There is one term in the disjunction for each tag-value in the tag-value-list.

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

```
(! <filter from remainder>)
```

where "filter from remainder" refers to the filter that would be constructed from the tag-value if the bang had not been present.

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

```
(name comparator compare-value)
```

where name is the name of the feature parameter after it has been decoded (see below), and comparator is either =, >= or <= depending of the initial characters in the phrase. If the remainder of the text in the tag-value after the equal contains a decimal point (implying a rational number), the decimal point is shifted right N times until it is an integer, I. Compare-value above is then set to "I / 10**N", where 10**N is the result of computing the number 10 to the Nth power.

> RFC 2533 uses a fractional notation to describe rational numbers. This specification use a decimal form. The above text merely converts between the two representations. Practically speaking, this conversion is not needed since the numbers are the same in either case. However, it is described in case implementations wish to directly plug the predicates generated by the rules in this section into an RFC 2533 implementation.

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

(name=[remainder])

where name is the feature-tag after it has been decoded (see below), and remainder is the remainder of the text in the tag-value after the #, with any decimal numbers converted to a rational form, and the colon replaced by a double dot (..).

If the tag-value does not begin with an octothorpe (it is a token-nobang or boolean), the filter is of the form:

(name=tag-value)

where name is the feature-tag after it has been decoded (see below).

If the feature-param contains a string-value (based on the fact that it begins with a left angle bracket ("<") and ends with a right angle bracket (">")), the filter is of the form:

(name="qdtext")

Note the explicit usage of quotes around the qdtext, 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 former, and sensitive for the latter. 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.

Feature tags, as specified in RFC 2506, cannot be directly represented as header field parameters in the Contact, Accept-Contact and Reject-Contact header fields. This is due to an inconsistency in the grammars, and in the need to differentiate feature parameters from parameters used by other extensions. As such, feature tag values are encoded from RFC2506 format to yield an enc-feature-tag, and then are decoded into RFC 2506 format. The decoding process is simple. If there is a leading plus (+) sign, it is removed. Any exclamation point (!) is converted to a colon (:) and any single quote (') is converted to a forward slash (/). The encoding process is similarly performed. Any forward slashes in the feature tag are converted to a single quote, and any colons are converted to an exclamation point. If the feature tag name is not amongst the base tags specified in Section 10, a plus sign is added to the front of the feature tag to create the encoded feature tag. The plus sign MUST NOT be added if the feature tag name is amongst the base tags.

As an example, the Accept-Contact header:

```
Accept-Contact:*;mobility="fixed";events="!presence,winfo";language="en,de"
 ;description="<PC>";+newparam;+rangeparam="#-4:+5.125"
```

would be converted to the following feature predicate:

```
(& (mobility=fixed)
   (| (! (events=presence)) (events=winfo))
   (| (language=en) (language=de))
   (description="PC")
   (newparam=TRUE)
   (rangeparam=-4..5125/1000))
```

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. The feature tag has to be encoded into a feature parameter using the process described above.

12 Security Considerations

The presence of caller preferences in a request has an effect on the ways in which the request is handled at a server. As a result, it is especially important that requests with caller preferences be integrity-protected. The same holds true for registrations with feature parameters in the Contact header field. User agents who are concerned with protecting the integrity of their requests SHOULD use the SIPS URI scheme.

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 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 [5]. 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 three 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 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 Acknowledgments

The initial set of media feature tags used by this specification were influenced by Scott Petrack's CMA design. Jonathan Lennox, Rohan Mahy and John Hearty provided helpful comments. Graham Klyne provided assistance on the usage of RFC 2533.

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

Paul Kyzivat
Cisco Systems
Mail Stop LWL3/12/2
900 Chelmsford St.
Lowell, MA 01851
email: pkzivat@cisco.com

16 Normative References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. 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] "Session initiation protocol (SIP) extension for instant messaging," RFC 3428, Internet Engineering Task Force, Dec. 2002.

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

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

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

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

[14] D. Willis and B. Hoeneisen, "Session initiation protocol (SIP) extension header field for registering non-adjacent contacts," RFC 3327, Internet Engineering Task Force, Dec. 2002.

[15] "Integration of resource management and session initiation protocol (SIP)," RFC 3312, Internet Engineering Task Force, Oct. 2002.

[16] J. Peterson, "A privacy mechanism for the session initiation protocol (SIP)," RFC 3323, Internet Engineering Task Force, Nov.

2002.

[17] J. Arkko, V. Torvinen, G. Camarillo, A. Niemi, and T. Haukka, "Security mechanism agreement for the session initiation protocol (SIP)," RFC 3329, Internet Engineering Task Force, Jan. 2003.

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

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

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

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

17 Informative References

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

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

[24] J. Rosenberg and H. Schulzrinne, "Guidelines for authors of extensions to the session initiation protocol (SIP)," internet draft, Internet Engineering Task Force, Nov. 2002. Work in progress.

[25] J. Rosenberg, "A presence event package for the session initiation protocol (SIP)," internet draft, Internet Engineering Task Force, Jan. 2003. Work in progress.

[26] J. Rosenberg, "A watcher information event template-package for the session initiation protocol (SIP)," internet draft, Internet Engineering Task Force, Jan. 2003. Work in progress.

[27] R. Sparks, "The SIP refer method," internet draft, Internet Engineering Task Force, Dec. 2002. Work in progress.

[28] 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.

[29] 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.

[30] J. Rosenberg, "A session initiation protocol (SIP) event package for registrations," internet draft, Internet Engineering Task Force, Oct. 2002. Work in progress.

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

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

[33] J. Rosenberg, "A framework for conferencing with the session initiation protocol," internet draft, Internet Engineering Task Force, Feb. 2003. Work in progress.

[34] M. Smith and T. Howes, "LDAP: string representation of search filters," internet draft, Internet Engineering Task Force, Aug. 2002. Work in progress.

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 set 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 predicate defines a relation over an N-dimensional space; its input is any point in that space (i.e. a feature collection), and is true for all points that are in the region thus defined.

RFC 2533 describes a syntax for writing down these N-dimensional boolean functions, borrowed from LDAP [34]. 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 the caller preferences 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.

Intellectual Property Statement

Director.

Full Copyright Statement