Network Working Group
Request for Comments: 3841
Category: Standards Track

J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia University
P. Kyzivat
Cisco Systems

August 2004

Caller Preferences for the Session Initiation Protocol (SIP)

Status of this Memo

Copyright Notice

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.

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 the processing of the request. These include:

- whether to proxy or redirect the request
- which URIs to proxy or redirect to
- whether to fork or not
- 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 execution of a program 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 method 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) [11]. 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 method may not be INVITE). For example, the caller might want to speak to a specific user, but wants 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, is 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 provided by a UA to its registrar [3]. 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.

One example of 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 BCP 14, RFC 2119 [4] and indicate requirement levels for compliant implementations.

3. Definitions

Much of the terminology used in this specification is presented in [3]. This specification defines the following additional terms:

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 whose UAC sends an INVITE request.

Feature Preferences: Caller preferences that describe desired properties of a UA to which the request is to be routed. Feature preferences can be made explicit 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.

Target Set: A target set is a set of candidate URIs to which a proxy or redirect server can send or redirect a request. 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.

4. Overview of Operation

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, is carried in the Request-Disposition header field. It describes 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, is carried in the Accept-Contact and Reject-Contact header fields. These header fields contain feature sets, represented by the same feature parameters that are used to indicate capabilities [3]. 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 because 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. UAC Behavior

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 determine whether servers along the path understand the header fields described in this specification, it includes a Proxy-Require header field with a value of "pref" [3] in its request. If the request should fail with a 420 response code, the UAC knows that the extension is not supported. In that case, it SHOULD retry, and may decide whether or not to use caller preferences. A UA should only use Proxy-Require if knowledge about support is essential for handling of the request. Note that, in any case, caller preferences can only be considered preferences - there is no guarantee that the requested service will be executed. As such, inclusion of a Proxy-Require header field does not mean that the preferences will be executed, just that the caller preferences extension is understood by the proxies.

5.1. 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 9.1.

5.2. 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. The syntax of the header field can be found in Section 10, and a discussion of their usage in Section 9.2.

Each feature set is constructed as described in Section 5 of [3]. 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].

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 "sip.methods" feature tag (note, however, that even though the name of this feature tag is sip.methods, it would be encoded into the Accept-Contact header field as just "methods"), 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 "sip.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.

As an example, the following Accept-Contact header field expresses a desire to route a call to a mobile device, using feature parameters taken from [3]:

Accept-Contact: *;mobility="mobile";methods="INVITE"

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 [12], 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 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 precise behavior depends heavily on whether the "require" and "explicit" parameters are present. When both of them are present, a proxy will only forward the request to contacts which have explicitly indicated that they support the desired feature set. Any others are discarded. As such, a UAC should only use "require" and "explicit" together when it wishes the call to fail unless a contact definitively matches. It's possible that a UA supports a desired feature, but did not indicate it in its registration. When a UAC uses both "explicit" and "require", such a contact would not be reached. As a result, this combination is often not the one a UAC will want.

When only "require" is present, it means that a contact will not be used if it doesn't match. If it does match, or if it's not known whether it's a complete match, the contact is still used. A UAC would use "require" alone when a non-matching contact is useless. This is common for services where the request simply can't be serviced without the necessary features. An example is support for specific methods or event packages. When only "require" is present, the proxy will also preferentially route the request to the UA which represents the "best" match. Here, "best" means that the UA has explicitly indicated it supports more of the desired features than any other. Note, however, that this preferential routing will never override an ordering provided by the called party. The preferential routing will only choose amongst contacts of equal q-value.

When only "explicit" is present, it means that all contacts provided by the callee will be used. However, if the contact isn't an explicit match, it is tried last amongst all other contacts with the same q-value. The principle difference, therefore, between this configuration and the usage of both "require" and "explicit" is the fallback behavior for contacts that don't match explicitly. Here, they are tried as a last resort. If "require" is also present, they are never tried.

Finally, if neither "require" nor "explicit" are present, it means that all contacts provided by the callee will be used. However, if the contact doesn't match, it is tried last amongst all other contacts with the same q-value. If it does match, the request is routed preferentially to the "best" match. This is a common configuration for preferences that, if not honored, will still allow for a successful call, and the greater the match, the better.

6. UAS Behavior

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 7.2 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. If a UA had registered against multiple separate addresses-of-record, and the contacts registered for each had different capabilities, it will have used a different URI in each registration, so it can determine which feature set to use.

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, 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 proceeds 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.

A UAS SHOULD process any queue directive present in a Request-Disposition header field in the request. All other directives MUST be ignored.

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 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 and it is the owner of the domain in the Request URI, the server SHOULD execute the directives as described in Section 9.1, 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 [2] 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 "audio", "automata", "class", "duplex", "data", "control", "mobility", "description", "events", "priority", "methods", "extensions", "schemes", "application", "video", "language", "type", "isfocus", "actor", or "text", or whose name begins with a plus (+) [3]. The proxy converts all of those parameters to the syntax of RFC 2533, based on the rules in Section 8.

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 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:

(sip.methods=[method of request])

7.2.2.2. Event Packages

For requests that establish a subscription [5], 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 that supports the given event package. To support this implicit preference, the proxy adds a term to the conjunction of the following form:

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

7.2.3. 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 [3] and then 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 "audio", "automata", "class", "duplex", "data", "control", "mobility", "description", "events", "priority", "methods", "schemes", "application", "video", "actor", "language", "isfocus", "type", "extensions", or "text" 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 [3], 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 8.

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";
    +sip.message="TRUE";other-param=66372;
    methods="INVITE,OPTIONS,BYE,CANCEL,ACK";schemes="sip,http"
```

This would be converted into the following predicate:

```
(& (sip.audio=TRUE)
   (sip.video=TRUE)
   (sip.mobility=fixed)
   (sip.message=TRUE)
   (| (sip.methods=INVITE) (sip.methods=OPTIONS) (sip.methods=BYE)
      (sip.methods=CANCEL) (sip.methods=ACK))
   (| (sip.schemes=sip) (sip.schemes=http)))
```

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

7.2.4. 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. These two cases are differentiated by the presence of angle brackets around the feature tag value. When these brackets are present (i.e., ;+sip.foo="<value>"), it implies case

sensitive string comparison. When they are not present, (i.e., (;+sip.bar="val"), it implies case insensitivity. 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 with 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 with 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 target 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 target 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 in 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.

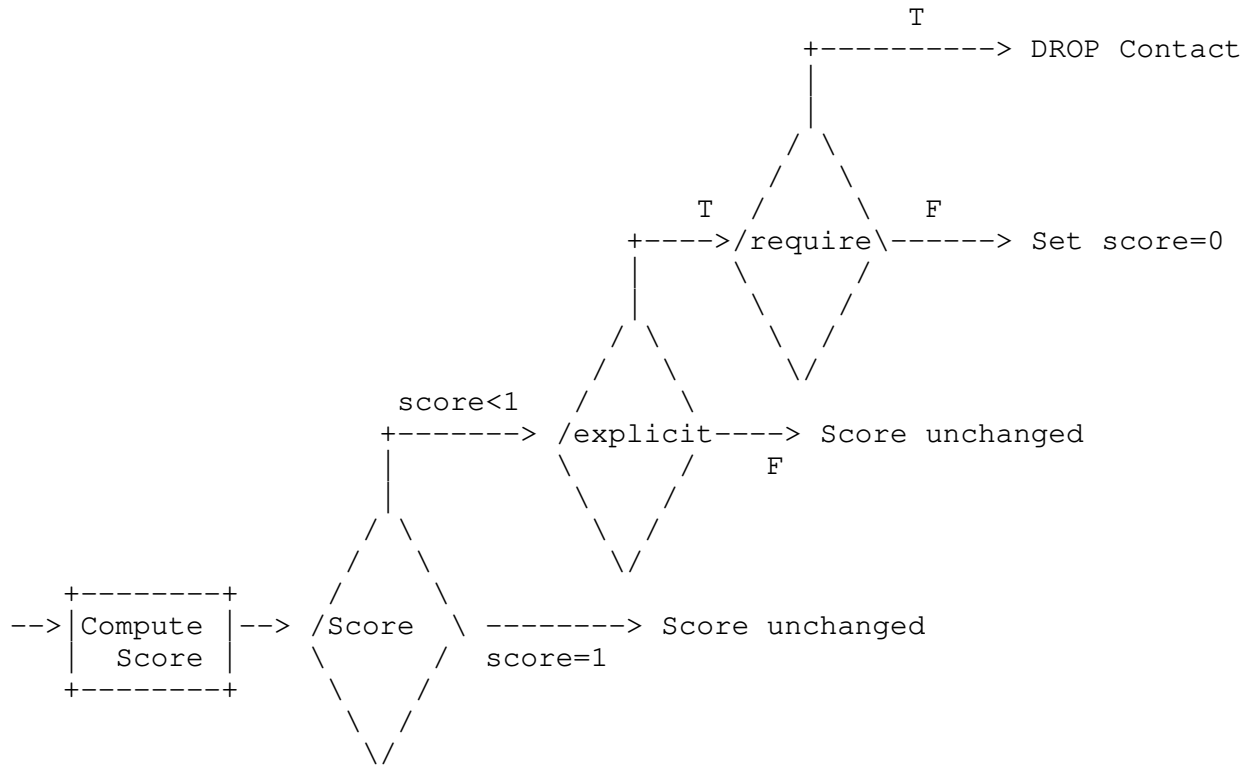

Figure 1: Applying the Score

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, the Accept-Contact predicate had an explicit tag, and 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. The overall caller preference, Qa, is the arithmetic average of S1 through SM.

At this point, any URIs that were removed from the target set because they were immune from caller preferences are added back in, and Qa for that URI is set to 1.0.

The purpose of the caller preference Qa is to provide an ordering for any contacts remaining in the target set, if the callee has not provided an ordering. To do this, the contacts remaining in the target set are sorted by the q-value provided by the callee. Once sorted, they are grouped into equivalence classes, such that all contacts with the same q-value are in the same equivalence class. Within each equivalence class, the contacts are then ordered based on their values of Qa. The result is an ordered list of contacts that is used by the proxy.

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, ordered by 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 response. 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, 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.

If a proxy server is recursing, it adds the Contact header fields returned in the redirect responses to the target set, and re-applies the caller preferences algorithm.

If the server is redirecting, it returns all entries in the target set. It assigns q-values to those entries so that the ordering is identical to the ordering determined by the processing above. 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.

7.2.5. 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=0.2
Contact: sip:u2@h.example.com;audio="FALSE";
  methods="INVITE";actor="msg-taker";q=0.2
Contact: sip:u3@h.example.com;audio;actor="msg-taker";
  methods="INVITE";video;q=0.3
Contact: sip:u4@h.example.com;audio;methods="INVITE,OPTIONS";q=0.2
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: *;actor="msg-taker";video
Accept-Contact: *;audio;require
Accept-Contact: *;video;explicit
Accept-Contact: *;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. That is because u3 is the only one that explicitly indicated support for video, and explicitly indicated it is a message taker. 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 matches 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 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 0.83 and Qa for u4 is 0.5. u5 is added back in with a Qa of 1.0.

Next, the remaining contacts in the target set are sorted by q-value. u5 has a value of 0.5, u1 has a q-value of 0.2 and so does u4. There are two equivalence classes. The first has a q-value of 0.5, and consists of just u5. Since there is only one member of the class, sorting within the class has no impact. The second equivalence class has a q-value of 0.2. Within that class, the two contacts, u1 and u4, are ordered based on their values of Qa. u1 has a Qa of 0.83, and u4, a Qa of 0.5. Thus, u1 comes first, followed by u4. The resulting overall ordered set of contacts in the target set is u5, u1, and then u4.

8. Mapping Feature Parameters to a Predicate

Mapping between feature parameters and a feature set predicate, formatted according to the syntax of RFC 2533 [2], is trivial. It is just the opposite of the process described in Section 5 of [3].

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 an exclamation mark (!), 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 exclamation mark 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 the 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.

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 are compared using case insensitive rules.

Feature tags, as specified in RFC 2506 [13], 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 RFC 2506 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 (/). If there was no leading plus sign, and the remainder of the encoded name was "audio", "automata", "class", "duplex", "data", "control", "mobility", "description", "events", "priority", "methods", "schemes", "application", "video", "actor", "isfocus", "extensions" or "text", the prefix "sip." is added to the remainder of the encoded name to compute the feature tag name.

As an example, the Accept-Contact header field:

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

would be converted to the following feature predicate:

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

9. Header Field Definitions

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

Figure 2 and Figure 3 are an extension of Tables 2 and 3 in RFC 3261 [1] for the Accept-Contact, Reject-Contact, and Request-Disposition header fields. The column "INF" is for the INFO method [6], "PRA" is for the PRACK method [7], "UPD" is for the UPDATE method [8], "SUB" is for the SUBSCRIBE method [5], "NOT" is for the NOTIFY method [5], "MSG" is for the MESSAGE method [9], and "REF" is for the REFER method [10].

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

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

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

Figure 3: Accept-Contact, Reject-Contact, and Request-Disposition header fields

9.1. 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 (e.g., you cannot have both "proxy" and "redirect" in the same Request-Disposition header 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 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 upon receipt of a 2xx. However, it would still send CANCEL on any outstanding branches upon 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). If there are multiple addresses with the highest q-value, the server chooses one based on its local policy. 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 not extensible on purpose. This is to avoid a proliferation of new extensions to SIP that are "tunneled" through this header field.

9.2. 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.

10. Augmented BNF

The BNF for the Request-Disposition header field is:

```
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"
```

The BNF for the Accept-Contact and Reject-Contact header fields is:

```
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 / req-param
                      / explicit-param / generic-param
                    ;;feature param from RFC 3840
                    ;;generic-param from RFC 3261
rc-params       =  feature-param / generic-param
req-param       =  "require"
explicit-param  =  "explicit"
```

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

11. 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, requests with caller preferences SHOULD be integrity-protected with the sips mechanism specified in RFC 3261, Section 26.

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.

12. IANA Considerations

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 3841

Header Field Name: Accept-Contact

Compact Form: a

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

RFC Number: RFC 3841

Header Field Name: Reject-Contact

Compact Form: j

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

RFC Number: RFC 3841

Header Field Name: Request-Disposition

Compact Form: d

13. Acknowledgments

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

14. References

14.1. Normative References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[2] Klyne, G., "A Syntax for Describing Media Feature Sets", RFC 2533, March 1999.

[3] Rosenberg, J., Schulzrinne, J., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Roach, A.B., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[6] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[7] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[8] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[9] Campbell, B., Ed., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[10] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

14.2. Informative References

[11] Lennox, J. and H. Schulzrinne, "Call Processing Language Framework and Requirements", RFC 2824, May 2000.

[12] Rosenberg, J., "Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)", Work in Progress, November 2002.

[13] Holtman, K., Muntz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.

15. Authors' Addresses

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

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

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

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

Paul Kyzivat
Cisco Systems
1414 Massachusetts Avenue
BXB500 C2-2
Boxboro, MA 01719
US

EMail: pkyzivat@cisco.com

16. Full Copyright Statement

Intellectual Property

Acknowledgement

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