Internet Engineering Task Force
Internet Draft
Category: Standards Track

Brian Stucker
Nortel Networks, Inc.

November 2001
Expires May 2002
<draft-stucker-sipping-publish-00.txt>

# SIP-Specific Network Service Publishing

## Status of this Memo



## Abstract

This document describes an extension to the Session Initiation Protocol (SIP). The purpose of this extension is to create a means for publishing and retrieving information regarding services in the network by using SIP.

The methods described in this document allow a framework by which arbitrary service data can be transported into the network for use by services running in the network. It is not intended to be a general-purpose mechanism for transport of arbitrary data as there are better suited mechanisms for this purpose (ftp, etc.), but is intended to be a simple, light-weight, mechanism that employs SIP in order to support SIP-related services. It is envisioned that each service that employs this method may extend and provide more detail as to how that service interacts

with the mechanisms put forth in this draft.





1. Table of Contents


| | | |
|---|---|---|
| 2 | Introduction | 2 |
| 2.1 | Overview of Operation | 3 |
| 3. | Syntax | 3 |
| 3.1 | New Methods | 3 |
| 3.2 | New Headers | 5 |
| 3.2.1 | "Action" Header | 5 |
| 3.2.2 | "Service" Header | 5 |
| 3.3 | The DATA Method | 6 |
| 3.4 | Usage of the Service Header | 7 |
| 3.5 | Usage of the Action Header | 8 |
| 3.5.1 | The PUT Action | 8 |
| 3.5.2 | The GET Action | 9 |
| 3.5.3 | The DELETE Action | 9 |
| 4. | Security Considerations | 9 |
| 4.1 | Authentication | 10 |
| 4.2 | Authorization | 10 |
| 5. | Statefulness of Service Data | 10 |
| 5.1 | Soft Stated Service Data | 11 |
| 5.2 | Hard Stated Service Data | 11 |
| 6. | Error Cases | 11 |
| 6.1 | Authorization or Authentication Failure | 11 |
| 6.2 | Service Location Failure | 12 |
| 6.3 | Wrong Data Format | 13 |
| 6.4 | Unsupported Action | 14 |
| 7. | Extensibility of Actions | 14 |
| 8. | Proof of Concept Example - Presence | 16 |
| 8.1 | Message Details | 17 |
| 9. | Compatibility with Proxies | 23 |
| 10. | Why use SIP? | 23 |
| 9. | References | 25 |
| 10. | Acknowledgements | 25 |
| 11. | Author's Address | 25 |

## 2. Introduction

The ability for a user to push data into the network has already shown its usefulness. In the case of CPL and presence, both require client devices to publish information regarding how services should behave, based on data pushed into the network.

However, the mechanism that is used to place data into the network has not been well defined up to this point, and problems exist with the current ad-hoc approach of using the REGISTER method to do so.

Therefore, this document describes a new method that creates a framework by which arbitrarty service data can be transported into the network for use by services. An example of this would be CPL scripting documents, or presence documents. It is not intended to be a general-purpose (non-SIP) data transport as there are better suited mechanisms for this purpose (ftp, etc.)

Meeting the data publication requirements for the general problem domain of how to transport any given piece of data between two SIP endpoints is not possible. The goal of this draft is to provide a SIP-specific framework that allows simple pieces of client data to be transported into the network for a specific, SIP-related, purpose. An example of such a purpose would be to transport CPL script data into the network for a CPL call routing service to handle SIP intiated calls appropriately.

SIP is chosen as the transport protocol because the services that this mechanism applies to are services created by way of using SIP. Where SIP is not the mechanism by which a service is provided, the use of the mechanisms described in this draft should be thought out

very carefully.

The framework described in this draft does not outline an extension which may be used directly. It is intended to be extensible, again, because the needs of different services whose data may need to be publish can vary. Each data type that intends to use this mechanism is responsible for defining the complete set of rules as to how this mechanism is to behave. This draft simply sets the framework for this to occur. Guidelines and requirements as to how these extensions must behave and requirements about their definition are described in later sections of this draft.

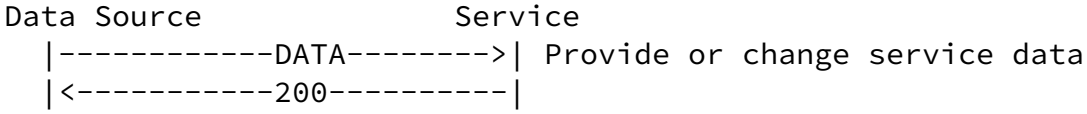




## 2.1. Overview of Operation

The concept of this draft is that entities in the network can notify a resource of changes in their configuration data. Whereas the generalized subscribe-notify framework is used to synchronize two finite state machines, synchronization of data across multiple nodes can be viewed as an event notification. Whenever the data changes at one end, it simply notifies the other side of the change.

A typical flow of messages to push data to a service would be:




Data may be hard-stated, and if so, must be refreshed in exactly the same manner as registrations (see RFC 2543 [1] ).

# 3. Syntax

This section describes the syntax extensions required for data notification in SIP. Semantics are described in section 4.

## 3.1. New Methods

This document describes one new SIP method: "DATA."

This table expands on tables 4 and 5 in RFC 2543 [1] .

| Header | Where | DATA |
|---|---|---|
| Accept | R | o |
| Accept-Encoding | R | o |
| Accept-Language | R | o |
| Action | Rr | m |
| Action | 400 | - |
| Alert-Info | g | - |
| Allow | Rr | o |
| Allow | 405 | m |
| Authentication-Info | 2xx | o |
| Authorization | R | o |
| Call-ID | c | m |
| Call-Info | | o |





| Header | Where | DATA |
|---|---|---|
| Contact | R | m |
| Contact | 1xx, 485 | o |
| Contact | 2xx, 3xx | m |
| Content-Disposition | | o |
| Content-Encoding | | o |
| Content-Length | | o |
| Content-Type | | m |
| CSeq | c | m |
| Date | | o |
| Encryption | g | o |
| Expires | g | o |
| From | c | m |
| In-Reply-To | R | - |
| Max-Forwards | R | o |
| Organization | g | o |
| Priority | R | - |
| Proxy-Authenticate | 407 | m |

| | | | |
|---|---|---|---|
| Proxy-Authorization | | R | o |
| Proxy-Require | | R | o |
| Record-Route | | R | o |
| Record-Route | 2xx, 481, 484 | | o |
| Require | | g | o |
| Retry-After | 404, 413, 480 | | o |
| Retry-After | 486, 500, 503 | | o |
| Retry-After | 600, 603 | | o |
| Route | | R | o |
| Server | | r | o |
| Service | | R | m |
| Service | | r | o |
| Subject | | R | o |
| Suppported | | | o |
| Timestamp | | | o |
| To | | gc(1) | m |
| Unsupported | | 420 | o |
| User-Agent | | | o |
| Via | | c | m |
| Warning | | r | o |
| WWW-Authenticate | | 401 | m |





## 3.2 New Headers

This table expands on tables 4 and 5 in RFC 2543 [1] , as amended by the changes described in section 3.1.

| Header field | where | proxy | ACK | BYE | CAN | INV | OPT | REG | DAT |
|---|---|---|---|---|---|---|---|---|---|
| Action | | | | | R | | | - | - - - |
| Service | | g | - | - | - | - | - | - | m |

### 3.2.1. "Action" header

The following header is defined for the purposes of this specification.

```
Action            =  ( "Action" ) ":" action-type
                     *( "," SP action-type )
action-type       =  ( "Get" | "Put" | "Delete" )
                     *("." action-subtype )
action-subtype    =  token-nodot
token-nodot       =  1*( alphanum | "-"  | "!" | "%" | "*"
                         | "_" | "+" | "`" | "'" | "~" )
```

Action is added to the definition of the element "request-header" in the SIP message grammar.

This document does not define values for action-subtypes. These values will be defined by individual event packages, and MUST be registered with the IANA.

There must be exactly one action type listed per action header. Multiple actions per message are disallowed except in error responses where the supported actions are listed.

### 3.2.2. "Service" header

The following header is defined for the purposes of this specification.

```
Service           =  ( "Service" ) ":" service-tag
                     *( "," SP service-tag )
service-tag       =  token-nodot
token-nodot       =  1*( alphanum | "-"  | "!" | "%" | "*"
                         | "_" | "+" | "`" | "'" | "~" )
```





Service is added to the definition of the element "general-header" in the SIP message grammar.

This document does not define values for service. These values will

be defined by individual event packages, and MUST be registered with the IANA.

There must be exactly one service type listed per action header. Multiple services per message are disallowed except in error responses where the supported services are listed.

## 3.3. The DATA method

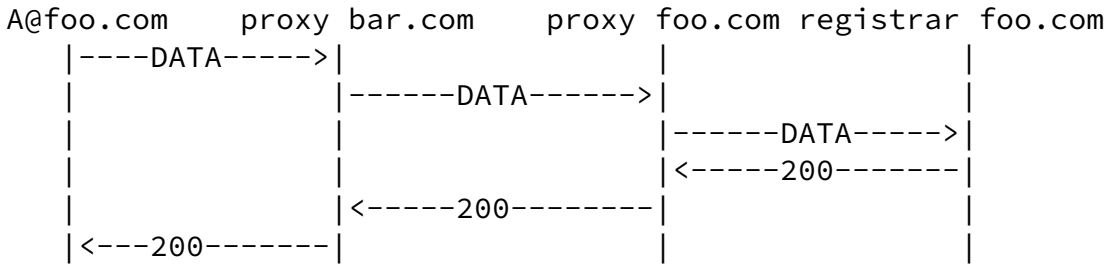
The DATA method is used to provide a mechanism for publishing data into the network. This is done on a domain basis, and follows the formatting, and routing rules used for the REGISTER method.

When the DATA method is routed to the server responsible for the domain listed in the requestURI, that server checks the Service header, and may modify the requestURI so that the message is then forwarded to the correct machine for that service, or may process the message immediately instead.

Example:

```
DATA foo.com SIP/2.0
Via: SIP/2.0/UDP pc.foo.com
To: sip:A@foo.com
From: sip:A@foo.com;tag=123
Service: cpl
Action: put
Call-ID: 9876@pc.foo.com
CSeq: 1288 DATA
Contact: sip:B@pc.foo.com
Content-Type: application/cpl+xml
Content-Length: ...

<cpl>
   <incoming>
          <location url="sip:meadams@47.102.21.11:5060">
             <redirect permanent="no" />
          </location>
   </incoming>
</cpl>
```




In the example shown above, since the "bar.com" proxy was not the definitive controller for "foo.com" the DATA message was forwarded (just like a REGISTER would be). However, once the DATA message hit the proxy at "foo.com" the proxy there was the definitive controller for "foo.com", but forwarded the message anyhow. This is allowed because that proxy may not have had the registrar, where "foo.com" handles cpl information co-located. Therefore the proxy was able to forward the DATA message on to the server that was handling the cpl service for "foo.com". This is allowed.

Once a DATA message gets to a server within the domain listed in the requestURI, one additional forwarding of the message is allowed in order to route to a server handling the service described in the Service header to cover the case that the service is not co-located with the inbound server for that domain. The requestURI MAY be changed in order to facilitate this routing, but only by a server in the domain that was listed in the original DATA message.

## 3.4 Usage of the Service Header

The Service header in the DATA message is used to denote the service for which the data being acted upon resides within. Each service MUST define and register with IANA it's own service tag to be placed in this header. Service tags used in this document are for example only, and have not necessarily been registered with IANA.

Servers within the domain listed in the request URI of the DATA message MUST inspect the contents of this header to see if the service specified is hosted on that machine. If it is not, the DATA message may be forwarded (within the same domain) in order to reach the correct service hosting the service specified.

Authors registering new service tags SHOULD pick service tags that as closely mimic the MIME type value for their data where the MIME type is unambiguous to the service the data is for.

For instance, if your service's data MIME type is application/voicemail+xml, a good basis for a service tag would be "voicemail" since it forms an identifiable portion of the MIME type header. If the service in question reuses a generic MIME type, the service tag should be descriptive of the service, and not the MIME type. You should always be able to look at the service tag and easily figure out what service it refers to.

### 3.5 Usage of the Action Header

The Action header in the DATA message is used to denote what the service should do, at a minimum, upon receiving the DATA message. Three default values are defined, of which the actual operation of each is dependent on the particular service.

Actions may be sub-typed as well, so that a service can define a sub-action called "put.merge" in order to have the data merged via rules specific to that service instead of simply overwritten by use of the "put" action. It is up to each service to decide how the data will be processed according to the action specified, including the operation of the three defaul values specified in this draft.

#### 3.5.1 The PUT Action

The PUT action is used to associate the resource identified within the message body of the DATA message with the TO party and service specified in the DATA message. It is modeled on the HTTP/1.1 PUT method, and acts as such. Services, generally, will take the data in the message body and apply it to the user identified in the TO header (similar to a REGISTER message) for the service specified in the Service header.

The result of this operation is denoted by the response to the DATA message, where 200 OK always denotes that the operation requested succeeeded. Other return values may include a warning header or explaination in the reason code as to why the operation failed. 4xx, 5xx, and 6xx return codes should always denote failure of the operation requested.

Using a DATA (PUT) message that contains no message body MUST NOT

be used in lieu of sending a DATA (DELETE) message, as this can cause information used to filter the data to be deleted to be confused with information to be stored for the resource identified in the DATA message.





### 3.5.2 The GET Action

The GET action is used to retrieve the data for the resource identified within the message body of the DATA message with the TO party and service specified in the DATA message. It is modeled on the HTTP/1.1 GET method, and acts as such. The message body of the DATA (GET) message may contain information used to filter which service data is to be returned. For example, a service may define message body elements to denote which version of a piece of service data is to be returned by sending a last-modified timestamp. Normally, the message body of a DATA (GET) message is expected to be empty, however.

The result of this operation is denoted by the response to the DATA message, where 200 OK always denotes that the operation requested succeeded. The data requested is returned in the message body of the response to the DATA (GET) request, and absence of data does not imply that the request failed. The value of the response code should always be used to determine success or failure of the get action. 4xx, 5xx, and 6xx return codes should always denote failure of the operation requested (even if data is present in the message body of these responses).

### 3.5.3 The DELETE Action

The DELETE action is used to delete the data for the resource identified within the message body of the DATA message with the TO party and service specified in the DATA message. It is modeled on the HTTP/1.1 DELETE method, and acts as such. The message body of the DATA (DELETE) message may contain information used to filter which service data is to be deleted, and MUST NOT be ambiguous. For example, a service may define message body elements to denote which portions of the service data is to be deleted by using a timestamp to delete information before the date noted.

The result of this operation is denoted by the response to the DATA message, where 200 OK always denotes that the operation request succeeded. An indication of what data was deleted MAY be included in the message body of the response, if so desired by the service. The value of the response code should always be used to determine success or failure of the delete action. 4xx, 5xx, and 6xx return codes should always denote failure of the operation requested (even if data is present in the message body of these responses).

## 4. Security Considerations





### 4.1 Authentication

Since the data contained in a DATA message can be potentially sensitive in nature, it is STRONGLY RECOMMENDED that all DATA messages be authenticated as to their source. Additionally, it is RECOMMENDED that they be encrypted. The extent to which SIP is used is up to the implementor as various transport protocols (such as TLS) can mitigate the need to add additional protection using SIP. However, SIP authentication SHOULD be part of any authentication scheme for service data since the service data is part of the SIP service space. Usage of HTTP Basic authentication is NOT RECOMMENDED.

### 4.2 Authorization

In order to support third-party service data (much like third-party registrations), the content of the TO header identifies the resource that the data in the message body (or action) is to be applied. In the case where TO and the FROM headers describe different parties, the FROM party must be authorized prior to the sending of the DATA message to take action on the TO party's service data. The means by which this authorization takes place is outside the scope of this document.

If the network wishes to conceal whether or not an update to the service data has succeeded or not, when a third-party is involved, it may send back a 200 response to the request, even though the request has failed. It should be stressed however, that such

operation should be used sparingly, and only in scenarios where notifying the third party that the request failed would reveal information that could be used in an attack on the network or TO party of the request. This follows the third-party registration model already in SIP [1].

## 5. Statefulness of Service Data

Service data may be soft-stated (expires after a specified period of time) or hard-stated (does not expire until modified explicitly). In order to support both models of statefulness, the presence or absence of an Expires header in the DATA message is utilized.

Mixed statefulness of data for the same service in the same request is undefined. If a service contains information that is both hard-stated and soft-stated, requests and responses must be crafted such that information can be kept separate unless such statefulness is identifiable within the message body itself.





### 5.1 Soft-Stated Service Data

Soft-stated service data is denoted by inclusion of an Expires header specifying the interval of time the data is to be considered valid. If an Expires value is set in the DATA request, it MUST be included in the response. The expires interval in the request may decreased by the service provider. The actual interval that the service data will be valid for MUST BE INCLUDED in the response.

Requests including an Expires header MUST NOT request an interval smaller than 60 seconds for the data to be valid.

An Expires header MUST NOT be included in a request containing a GET or DELETE action (or an sub-typed action derived from these actions). Responses to a GET MAY contain an Expires header to show the remaining amount of time soft-stated service data may be valid for. Responses to a DELETE action MUST NOT contain an Expires header.

### 5.2 Hard-Stated Service Data

Hard-stated service data is denoted by NOT including an Expires header in the request. Service data contained in such requests SHOULD remain valid until replaced or removed. Responses to requests containing hard-stated data should likewise not contain an Expires header.

Statefulness for the DELETE action is defined to be hard-stated since completion of the action removes any material that could be soft-stated. Responses to a DELETE action should therefore always be considered hard-stated, and not contain an Expires header.

Statefulness for the GET action is defined to be both soft-stated and hard-stated. Requests with a GET action are considered hard-stated, but responses MAY or MAY NOT contain an Expires header depending on the statefullness of the data retrieved. If the data retrieved is hard-stated, an Expires header should not be included in the response.

## 6. Error Cases

### 6.1 Authorization or Authentication Failure

Errors arising from authorization or authentication failures should be handled according to the rules for the REGISTER method defined within SIP.





### 6.2 Service Location Failure

In the case that a service is not located, it is possible to return the known set of services that are supported for data publication in the response. This is done by including the list in the Service header returned in the response to the request. The response MUST NOT contain the Action header.

Example:

```
DATA foo.com SIP/2.0
Via: SIP/2.0/UDP pc.foo.com
To: sip:A@foo.com
From: sip:A@foo.com;tag=123
```

```
Service: cpl
Action: put
Call-ID: 9876@pc.foo.com
CSeq: 1288 DATA
Contact: sip:B@pc.foo.com
Content-Type: application/cpl+xml
Content-Length: ...

<cpl>
   <incoming>
          <location url="sip:meadams@47.102.21.11:5060">
             <redirect permanent="no" />
          </location>
   </incoming>
</cpl>


SIP/2.0 400 Service Not Supported
Via: SIP/2.0/UDP pc.foo.com
To: sip:A@foo.com
From: sip:A@foo.com;tag=123
Service: presence, voicemail
Call-ID: 9876@pc.foo.com
CSeq: 1288 DATA
Contact: sip:B@pc.foo.com
Content-Length: 0
```





### 6.3 Wrong Data Format

If the service is known, but the data supplied is not formatted according to the requirements of that service, an error response SHOULD be generated for the request using a 415 Unsupported Media Type, identifying the accepted formats using the Accept, Accept-Encoding, and Accept-Language headers.

Example:

```
   DATA foo.com SIP/2.0
   Via: SIP/2.0/UDP pc.foo.com
   To: sip:A@foo.com
   From: sip:A@foo.com;tag=123
   Service: presence
   Action: put
   Call-ID: 9876@pc.foo.com
   CSeq: 1288 DATA
   Contact: sip:B@pc.foo.com
   Content-Type: application/cpl+xml
   Content-Length: ...

   <cpl>
      <incoming>
             <location url="sip:meadams@47.102.21.11:5060">
                <redirect permanent="no" />
             </location>
      </incoming>
   </cpl>


   SIP/2.0 415 Unsupported Media Type
   Via: SIP/2.0/UDP pc.foo.com
   To: sip:A@foo.com
   From: sip:A@foo.com;tag=123
   Service: presence
   Action: put
   Call-ID: 9876@pc.foo.com
   CSeq: 1288 DATA
   Contact: sip:B@pc.foo.com
   Content-Length: 0
   Accept: application/cpim+pidf
```

## 6.4 Unsupported Action

If the service identified does not support the requested action, an error SHOULD be generated. In this case a 400 response is used to denote a failure in the syntax of the request. Since the Action is the incorrect header, the Service header MUST NOT be included. This is done in order to make it clear which header was in error.

Example:

```
DATA foo.com SIP/2.0
Via: SIP/2.0/UDP pc.foo.com
To: sip:A@foo.com
From: sip:A@foo.com;tag=123
Service: cpl
Action: put.merge
Call-ID: 9876@pc.foo.com
CSeq: 1288 DATA
Contact: sip:B@pc.foo.com
Content-Type: application/cpl+xml
Content-Length: ...

<cpl>
   <incoming>
          <location url="sip:meadams@47.102.21.11:5060">
             <redirect permanent="no" />
          </location>
   </incoming>
</cpl>


SIP/2.0 400 Unsupported Action
Via: SIP/2.0/UDP pc.foo.com
To: sip:A@foo.com
From: sip:A@foo.com;tag=123
Action: put, get, delete
Call-ID: 9876@pc.foo.com
CSeq: 1288 DATA
Contact: sip:B@pc.foo.com
Content-Length: 0
```

# 7. Extensibility of Actions

Services may extend the basic action values of PUT, GET, and DELETE for their purposes. Extending these should be done only when specifying the correct action to take would be difficult using the contents of the message body.

Such extensions are denoted by a ".extension". For instance, if an append action were desired for a service data operation, the extended action would be "put.append", etc.

Extensions should not violate the basic operation of the base action. For instance, "delete.append" would not be considered a valid extension because to append is to add, which is counter to the base action "delete" which is to remove.

Extensions must be defined explicitly for each of the basic actions. Defining a sub-action type for one action does not define it for all other action types. This may result in simply stating that the sub-action type is not defined for a given basic action.

Example:

In order to define the append sub-action, you might state:

put.append - Appends data onto the bottom of the already existing service data.

get.append - Unsupported.

delete.append - Unsupported.

Example:

In orde to define that actions should take into account the Date header of the request, you might state:

put.date - Overwrites service data older than the date specified in the request.

get.date - Retrieves service data newer than the date specified in the request.

delete.date - Deletes service data older than the date specified in the request.

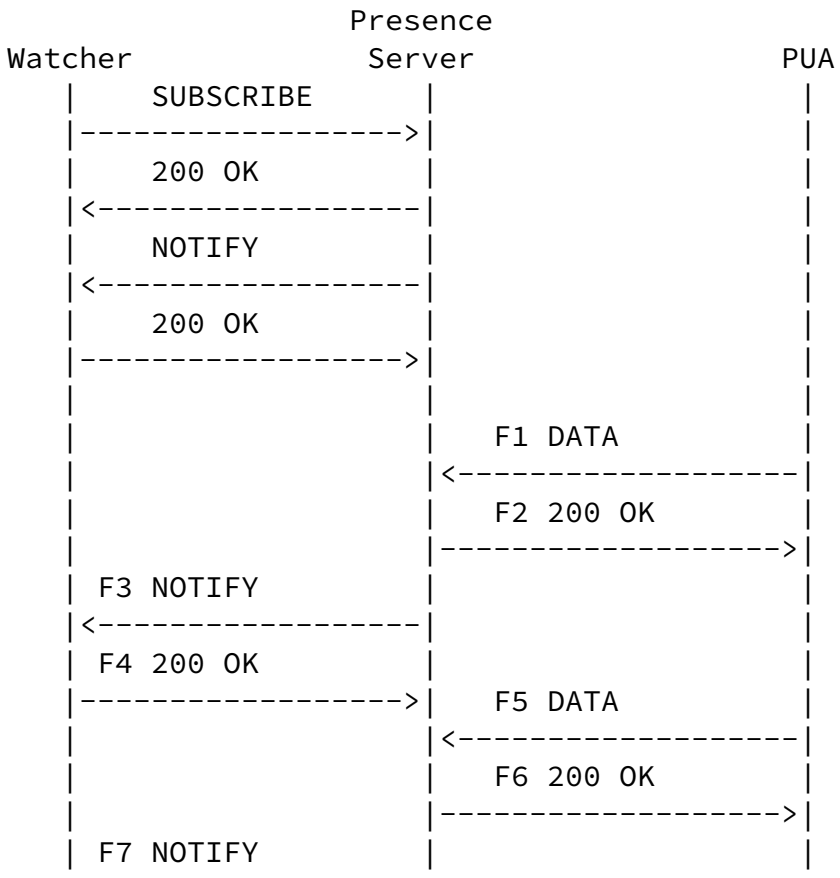
Note, that any such sub-actions would need to be registered with IANA to ensure that the namespace for the action header is coordinated. This would be done as part of defining a service specific publication package based on this draft.

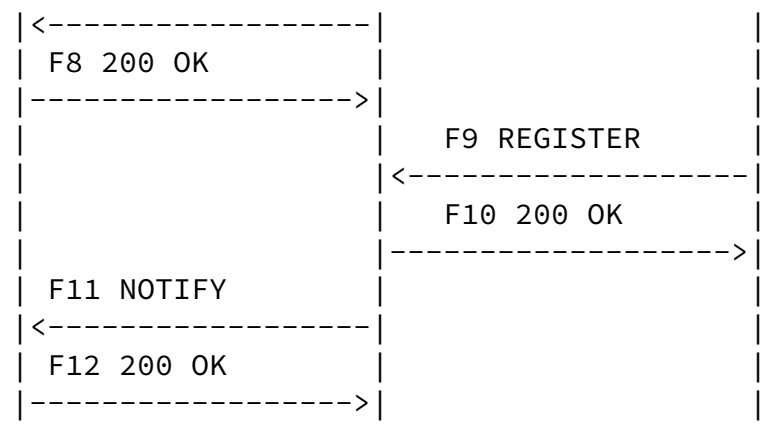




## 8. Proof of Concept Example - Presence

In order to provide a proof of concept example to help illustrate how the DATA method would work with a real-world application, the following section gives an example of how DATA could be used to publish presence documents according to the framework of the SIMPLE draft. The exact rules of how the presence service would handle each of the actions for DATA would need to be further specified in a separate draft. Depending on those rules, and policy uploaded by the user, the actual messaging outcomes may vary from what is put forth here. This section is for demonstration only.










## 8.1 Message Details

F1 DATA PUA->presence server

```
DATA sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:userA@example.com>
From: <sip:userA@example.com>;tag=1234
Call-ID: 9876@pua.example.com
CSeq: 1 DATA
Service: presence
Action: put
Accept: application/cpim-pidf+xml
Contact: <sip:userA@pua.example.com>

<presence xmlns="http://www.ietf.org/ns/cpim-pidf-xml-1.0">
   <tuple name="im-1">
     <status>
       <value>open</value>
       <detail type="im"
        schema="http://www.ietf.org/dtd/im-type-im.dtd">available</detail
     </status>
     <contact priority="2">im:userA@pua.example.com</contact>
   </tuple>
</presence>
```

F2 200 OK  presence server->PUA

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:userA@example.com>;tag=abcd
From: <sip:userA@example.com>;tag=1234
Call-ID: 9876@pua.example.com
CSeq: 1 DATA
Contact: sip:example.com
```





F3 NOTIFY  presence server-> watcher

```
NOTIFY sip:userB@watcherhost.example.com SIP/2.0
Via: SIP/2.0/UDP presence.example.com:5060
From: <sip:userA@example.com>;tag=ffd2
To: <sip:userB@example.com>;tag=xfg9
Call-ID: 192837@presence.example.com
CSeq: 1 NOTIFY
Event: presence
Content-Type: application/cpim-pidf+xml
Content-Length: ..

<presence xmlns="http://www.ietf.org/ns/cpim-pidf-xml-1.0">
   <tuple name="im-1">
     <status>
       <value>open</value>
       <detail type="im"
        schema="http://www.ietf.org/dtd/im-type-im.dtd">available</detail
     </status>
     <contact priority="2">im:userA@pua.example.com</contact>
   </tuple>
</presence>
```

F4 200 OK

```
Via: SIP/2.0/UDP presence.example.com:5060
From: <sip:userA@example.com>;tag=ffd2
To: <sip:userB@example.com>;tag=xfg9
Call-ID: 192837@presence.example.com
CSeq: 1 NOTIFY
```





F5 DATA PUA->presence server

```
DATA sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:userA@example.com>
From: <sip:userA@example.com>;tag=1234
Call-ID: 9876@pua.example.com
CSeq: 2 DATA
Service: presence
Action: delete
Accept: application/cpim-pidf+xml
Contact: <sip:userA@pua.example.com>

<presence xmlns="http://www.ietf.org/ns/cpim-pidf-xml-1.0">
```

```
    <tuple name="im-1">
      <status>
        <value>open</value>
      </status>
    </tuple>
  </presence>
```

F6 200 OK  presence server->PUA

```
 SIP/2.0 200 OK
 Via: SIP/2.0/UDP pua.example.com:5060
 To: <sip:userA@example.com>;tag=abcd
 From: <sip:userA@example.com>;tag=1234
 Call-ID: 9876@pua.example.com
 CSeq: 2 DATA
 Contact: sip:example.com
```





F7 NOTIFY  presence server-> watcher

```
 NOTIFY sip:userB@watcherhost.example.com SIP/2.0
 Via: SIP/2.0/UDP presence.example.com:5060
 From: <sip:userA@example.com>;tag=ffd2
 To: <sip:userB@example.com>;tag=xfg9
 Call-ID: 192837@presence.example.com
 CSeq: 2 NOTIFY
```

```
Event: presence
Content-Type: application/cpim-pidf+xml
Content-Length: ..

<presence xmlns="http://www.ietf.org/ns/cpim-pidf-xml-1.0">
   <tuple name="im-1">
     <status>
       <value>open</value>
     </status>
     <contact priority="2">im:userA@example.com</contact>
   </tuple>
</presence>
```

F8 200 OK

```
Via: SIP/2.0/UDP presence.example.com:5060
From: <sip:userA@example.com>;tag=ffd2
To: <sip:userB@example.com>;tag=xfg9
Call-ID: 192837@presence.example.com
CSeq: 2 NOTIFY
```

F9 REGISTER PUA->registrar/presence server

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP pua.example.com:5060
To: <sip:userA@example.com>
From: <sip:userA@example.com>
Call-ID: 234897@pua.example.com
CSeq: 2 REGISTER
Contact: <sip:userA@pua.example.com>
Expires: 0
```





F10 200 OK registrar/presence server->PUA

```
 SIP/2.0 200 OK
 Via: SIP/2.0/UDP pua.example.com:5060
 To: <sip:userA@example.com>
 From: <sip:userA@example.com>
 Call-ID: 234897@pua.example.com
 CSeq: 2 REGISTER

F11 NOTIFY  presence server-> watcher

 NOTIFY sip:userB@watcherhost.example.com SIP/2.0
 Via: SIP/2.0/UDP presence.example.com:5060
 From: <sip:userA@example.com>;tag=ffd2
 To: <sip:userB@example.com>;tag=xfg9
 Call-ID: 192837@presence.example.com
 CSeq: 3 NOTIFY
 Event: presence
 Content-Type: application/cpim-pidf+xml
 Content-Length: ..

 <presence xmlns="http://www.ietf.org/ns/cpim-pidf-xml-1.0">
    <tuple name="im-1">
      <status>
        <value>closed</value>
      </status>
      <contact priority="2">im:userA@example.com</contact>
    </tuple>
 </presence>


F12 200 OK

 Via: SIP/2.0/UDP presence.example.com:5060
 From: <sip:userA@example.com>;tag=ffd2
 To: <sip:userB@example.com>;tag=xfg9
 Call-ID: 192837@presence.example.com
 CSeq: 3 NOTIFY
```

## 9. Compatibility with Proxies

Because the DATA method shares the same routing rules as a REGISTER method message, in general, the routing of a DATA message is compatible with the current (bis-05) routing rules contained in SIP [1].

In particular, a stateless proxy is allowed to inspect the service header in order to determine the next hop the message should take. This is allowed for in SIP. However, should a stateless proxy be the definitive domain controller for a given domain, the DATA message may be forwarded an additional time (if needed), instead of the one-more-hop rule stated earlier. Should the stateless proxy forward to another stateless proxy, an additional hop is again allowed until the DATA message reaches the correct server to process the request, or reaches a stateful proxy server.

This follows the routing rules set forth in section 16 of (bis-05) SIP.

## 10. Why use SIP?

There are several arguments for using SIP as a mechanism by which to publish generic service information.

First, there are already several such mechanisms defined within SIP. The REGISTER method is used to publish contact information for network routing services. It is possible to use SIP quite well, without the use of the REGISTER message; at the cost that there is no network based routing. Thus, in order for network based routing services to work, a mechanism for publishing information into the SIP network must be supported: hence REGISTER.

Another well-known method of publishing service data is the SUBSCRIBE-NOTIFY framework. There are a number of event packages including presence, watcherinfo, REFER, and voicemail that have defined message bodies to support the transport of service information via SIP.

Finally, SIP itself is designed to be open for service data transport. Nearly any message sent using SIP is allowed to contain a message body. It is the very rare case that the message body MUST be empty, and the content-type header is very rarely excluded from any given method defined for SIP.

The common thread here, though, is that data that is transported is *directly related* to the service space created by SIP. This is done, presumably, to make the protocol simpler to understand by limiting the number of transports, and easier to implement (again, by using a common protocol). By inspection of the SIP RFC, and the many drafts and extensions that have been created since, the conclusion, therefore, can be made that using SIP to carry service specific information, for services defined in the SIP space (not any arbitrary service), is appropriate.

## 11. References

[1] M. Handley/H. Schulzrinne/E. Schooler/J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, IETF; March 1999.

[2] A. Roach, "SIP-Specific Event Notification", <draft-ietf-sip-events-01.txt>, IETF; November 2001. Work in progress.

[3] R. Fielding et. al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC2068, IETF, January 1997.

[4] S. Donovan, "Requirements for Publication of SIP Related Service Data", <draft-donovan-publish-requirements-00.txt>, September 2001. Work in progress.

[5] Rosenberg, J. and H. Schulzrinne, "Guidelines for Authors of SIP Extensions", RFC Guidelines, March 2001.

[6] Rosenberg, J., et. al., "SIP Extensions for Presence", draft-ietf-simple-presence-03.txt, September 2001. Work in pregress.

[7] M. Handley/H. Schulzrinne/E. Schooler/J. Rosenberg/ G. Camarillo/A. Johnston/J. Peterson/R. Sparks/E. Schooler, "SIP: Session Initiation Protocol", <draft-ietf-sip-rfc2543bis-05.txt>, October 2001. Work in Progress.

## 12. Acknowledgements

The author wishes to thank Steve Donovan for a thorough presentation of service data publication requirements, as well

as Jennifer Beckman, Trip Ingle, Alex Nava, Mary Barnes, and Sriram Parameswar for various guidance and support in the creation of this draft.

## 13. Author's Address

Brian Stucker
Nortel Networks, Inc.
2375-B Glenville Rd.
Richardson, TX 75082
USA
Phone: +1 972 685 7724
Fax: +1 972 685 3653
E-Mail: bstucker@nortelnetworks.com