XEP-0273: Stanza Interception and Filtering Technology (SIFT)

Joe Hildebrand
mailto:jhildebr@cisco.com
xmpp:hildjj@jabber.org

Jack Moffitt
mailto:jack@chesspark.com
xmpp:jack@chesspark.com

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

2011-06-27
Version 0.4

Status	Type	Short Name
Deferred	Standards Track	sift

This specification defines an XMPP protocol extension that enables a client to exercise control over the XML stanzas it will receive from the server by instructing the server to intercept and filter inbound stanzas.

Legal

Copyright

Permissions

Warranty

NOTE WELL: This Specification is provided on an ”AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF’s Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction

In some scenarios a client might want to control the XML stanzas it will receive over its stream with the server. Some potential use cases include:

- A mobile client might want to receive messages but not presence notifications, since the latter are quite "chatty" and can run down the battery.
- A softphone might want to receive IQ stanzas only if the payload is qualified by an XML namespace related to the use of Jingle (XEP-0166) [1] for Internet telephony.
- A presence compositor might want to receive presence updates but not message stanzas or IQ stanzas, and only from the user's own resources (i.e., not from other entities).

Although XMPP IM [2] specifies the use of a negative presence priority to block inbound message delivery, it does not enable the client to block inbound presence notifications, filter inbound IQ stanzas, or otherwise exercise fine-grained control over the delivery of inbound stanzas. While it would be possible to define particular values of negative presence priorities for some delivery control methods (e.g., <priority>-2<priority> could be hardcoded to mean "don't send me messages or presence"), that would be an ugly hack and thus inconsistent with XMPP Design Guidelines (XEP-0134) [3]. Therefore, this specification defines a stanza interception and filtering technology (a.k.a. "SIFT") that is more consistent with the underlying design of XMPP.

The following taxonomy of client types is not exhaustive but might assist developers in understanding the scenarios in which SIFT might be useful.

Type	Sends Presence *	Receives Presence **	Receives Messages
Normal User	Yes	Yes	Yes
Invisible User	No	Yes	Yes
Large-Scale Bot	Yes	No	Yes
Presentity	Yes	Yes	No
Presence Watcher	No	Yes	No
Presence Publisher	Yes	No	No
Message Subscriber	No	No	Yes
Message Publisher	No	No	No

[1] XEP-0166: Jingle <https://xmpp.org/extensions/xep-0166.html>.

[2] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

[3] XEP-0134: XMPP Design Guidelines <https://xmpp.org/extensions/xep-0134.html>.

* Note: SIFT is not used to control outbound presence, since that use case is already covered by Privacy Lists (XEP-0016) [4]; this table includes information about outbound presence only to motivate various scenarios in which SIFT can be used.
* Note: For purposes of this taxonomy, we refer only to presence notifications (available and unavailable), not also to subscription-related presence stanzas (subscribe, unsubscribe, etc.).

2 Requirements

The SIFT protocol is designed to meet the following requirements.

1. Make it possible for a client to disable receipt of various inbound stanzas (presence notifications, subscription-related presence stanzas, message stanzas, IQ stanzas) while still receiving other kinds of stanzas.
2. Make it possible for a client to "sift" based on all senders, local vs. remote senders, or other senders vs. oneself.
3. Make it possible for a client to "sift" based on whether the recipient is the user's bare JID or the particular client's full JID.
4. Enable future extensibility based on regular expressions, XPath expressions, etc.

3 Protocol

The SIFT protocol is used to *intercept* or *filter* inbound stanzas only, not outbound stanzas sent by the client to the server or other entities. By "intercept" is meant that the server will not deliver any such stanza kind (message, presence notification, presence subscription, or IQ) to the client, and by "filter" is meant that the server will apply a rule to determine if the specific stanza will be delivered to the client (e.g., matching against a payload namespace); in general we refer to these actions as "sifting". The SIFT protocol enables the server to support only basic interception (even here to support interception only for particular kinds of stanzas), basic filtering as defined by the rules described in this specification, or advanced filtering using extensions to SIFT defined in other specifications. Each of the features supported by the server can be discovered by the client for maximum interoperability. The features, the process for discovering them, and the process for enabling them are described in the following sections.

[4] XEP-0016: Privacy Lists <https://xmpp.org/extensions/xep-0016.html>.

3.1 Features

SIFT supports the features defined below. Each feature is identified by a separate value for 'var' attribute qualified by the 'http://jabber.org/protocol/disco#info' namespace as specified in Service Discovery (XEP-0030) [5].

3.1.1 Stanza Kinds

A server MAY support any combination of sifting IQ stanzas, message stanzas, presence notifications, and presence subscriptions, as advertised by the following service discovery features.

urn:xmpp:sift:stanzas:iq The server enables the client to sift all <iq/> stanzas or ones that match the specified criteria.

urn:xmpp:sift:stanzas:message The server enables the client to sift all <message/> stanzas or ones that match the specified criteria.

urn:xmpp:sift:stanzas:presence The server enables the client to sift all <presence/> notifications (i.e., presence stanzas with no 'type' or with a type of "unavailable") or ones that match the specified criteria.

urn:xmpp:sift:stanzas:sub The server enables the client to sift all subscription-related <presence/> stanzas (i.e., presence stanzas with a type of "subscribe", "subscribed", "unsubscribe", or "unsubscribed") or ones that match the specified criteria.

3.1.2 Sender

A server MAY enable the client to sift based on sender. The following features are supported.

urn:xmpp:sift:senders:all The server shall sift this kind of stanza no matter who the sender is. This is the default.

urn:xmpp:sift:senders:local The server shall sift this kind of stanza only from entities associated with the same local domain as the user itself (not from remote domains).

urn:xmpp:sift:senders:others The server shall sift this kind of stanza only from other entities (not from the user itself).

urn:xmpp:sift:senders:remote The server shall sift this kind of stanza only from entities associated with remote domains (not from the same local domain as the user itself).

urn:xmpp:sift:senders:self The server shall sift this kind of stanza only from the user itself (not from other entities).

[5] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

These values are child elements of the <iq/>, <message/>, <presence/>, and <sub/> elements when the server returns a features discovery result, whereas they are values of the ’sender’ attribute when the client enables sift support.

3.1.3 Recipient

A server MAY enable the client to filter based on recipient. The following features are supported.

urn:xmpp:sift:recipients:all The server shall sift this kind of stanza if the recipient is the bare JID <localpart@domain.tld> of the user or the full JID <localpart@domain.tld/resource> of the particular resource. This is the default.

urn:xmpp:sift:recipients:bare The server shall sift this kind of stanza only if the recipient is the bare JID <localpart@domain.tld> of the user.

urn:xmpp:sift:recipients:full The server shall sift this kind of stanza only if the recipient is the full JID <localpart@domain.tld/resource> of the particular resource.

These values are child elements of the <iq/>, <message/>, <presence/>, and <sub/> elements when the server returns a features discovery result, whereas they are values of the ’recipient’ attribute when the client enables sift support.

3.1.4 Payload

A server MAY enable the client to sift based on the XML namespace and element name of the payload(s) that the client allows for delivery. If so, the server shall advertise a feature of **urn:xmpp:sift:payloads:qname**.
By ”payload” is meant a first-level child element of an <iq/>, <message/>, or <presence/> stanza. **This includes elements defined in RFC 6120 [6] and RFC 6121 [7].** As a result, if the server supports payload matching then a client can even sift out elements allowed by the ’jabber:client’ namespace, such as message <subject/> and presence <status/>.

3.1.5 Advanced Matching

A server could match based on more complex criteria, e.g. Regular Expressions or XPath Expressions; such functionality is implicitly allowed because the XML schema specifies the <xs:any/> notation, but any such advanced matching shall be defined in separate specifications.

[6] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

[7] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

3.2 Discovering Supported Features

A client can discover if its server supports SIFT by sending a disco#info request.

Listing 1: A disco#info query

```
<iq type=’get’
    from=’romeo@montague.lit/pda’
    to=’montague.lit’
    id=’bf4vb167’>
  <query xmlns=’http://jabber.org/protocol/disco#info’/>
</iq>
```

If a server supports the SIFT protocol, it MUST advertise that fact in its responses to ”disco#info” requests by returning a feature of ”urn:xmpp:sift:2” (see Namespace Versioning regarding the possibility of incrementing the version number). The server MUST also specify which features it supports.
In the following reply, the server indicates that it supports a minimal subset of SIFT features merely for the sake of presence blocking.

Listing 2: Minimal server support

```
<iq type=’result’
    from=’montague.lit’
    to=’romeo@montague.lit/pda’
    id=’bf4vb167’>
  <query xmlns=’http://jabber.org/protocol/disco#info’>
    <feature var=’urn:xmpp:sift:2’/>
    <feature var=’urn:xmpp:sift:stanzas:presence’/>
  </query>
</iq>
```

In the following reply, the server indicates that it supports a wider range of SIFT features.

Listing 3: More extensive server support

```
<iq type=’result’
    from=’montague.lit’
    to=’romeo@montague.lit/pda’
    id=’bf4vb167’>
  <query xmlns=’http://jabber.org/protocol/disco#info’>
    <feature var=’urn:xmpp:sift:2’/>
    <feature var=’urn:xmpp:sift:recipients:all’/>
    <feature var=’urn:xmpp:sift:senders:all’/>
    <feature var=’urn:xmpp:sift:senders:others’/>
    <feature var=’urn:xmpp:sift:stanzas:iq’/>
    <feature var=’urn:xmpp:sift:stanzas:message’/>
    <feature var=’urn:xmpp:sift:stanzas:presence’/>
    <feature var=’urn:xmpp:sift:stanzas:sub’/>
  </query>
</iq>
```

To enable clients to cache information about supported features, a server SHOULD return Entity Capabilities (XEP-0115) [8] data via stream features as described in Entity Capabilities (XEP-0115) [9].

3.3 Enabling SIFT

To enable sifting of stanzas, the client sends an IQ-set to the server containing a <sift/> child element that in turn contains an <iq/> element, a <message/> element, a <presence/> element, or some combination of those elements. Each of these elements MAY include a ’recipient’ attribute whose value is ”all”, ”bare”, or ”full” (defaulting to ”all”). Each of these elements MAY also include a ’sender’ attribute whose value is ”all”, ”local”, ”others”, ”remote”, or ”self” (defaulting to ”all”).
Note: The last SIFT request sent from the client to the server overrides all previous SIFT requests; SIFT requests are not cumulative. Therefore, each SIFT request needs to contain all the SIFT rules that the client wishes the server to enforce, not a delta from the previous request.

Listing 4: Sifting of message and presence stanzas

```
<iq from=’romeo@montague.lit/pda’
    id=’rv491g37’
    to=’romeo@montague.lit’
    type=’set’>
  <sift xmlns=’urn:xmpp:sift:2’>
    <message sender=’others’/>
    <presence/>
  </sift>
</iq>
```

The foregoing IQ-set means ”sift messages from others and presence from all senders, no matter if the recipient is my bare JID or my full JID”.
Each of the child elements <iq/>, <message/>, <presence/>, and <sub/> MAY also contain one or more <allow/> children whose ’name’ attribute specifies the element name and whose ’ns’ attribute specifies the XML namespace of stanza payloads the client would like to allow. If no <allow/> elements are included, then sifting of that kind of stanza is completed without reference to the payload.

Listing 5: Sifting for particular IQ payloads

[8] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.
[9] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

```
<iq from='romeo@montague.lit/pda'
    id='bs01jg75'
    to='romeo@montague.lit'
    type='set'>
  <sift xmlns='urn:xmpp:sift:2'>
    <iq>
      <allow name='jingle' ns='urn:xmpp:jingle:1'/>
      <allow name='query' ns='http://jabber.org/protocol/disco#info'/>
    </iq>
    <message/>
  </sift>
</iq>
```

The foregoing IQ-set means ”filter out inbound IQ stanzas except if the payload matches <jingle xmlns='urn:xmpp:jingle:1'/> or <query xmlns='http://jabber.org/protocol/disco#info'/>”. In XMPP, an IQ stanza can contain only one payload element, so the filtering logic is straightforward. However, a message or presence stanza can contain multiple payload elements (cf. Message Stanza Profiles (XEP-0226) [10]). Therefore, filtering for message and presence stanzas means that if the stanza contains the defined payload or payloads (perhaps in addition to other payloads), the server shall deliver it to the client.
For instance, the following example shows how a client would filter inbound messages and IQs to only receive SOAP payloads as specified in SOAP over XMPP (XEP-0072) [11].

Listing 6: Sifting for SOAP

```
<iq from='romeo@montague.lit/pda'
    id='cid143n9'
    to='romeo@montague.lit'
    type='set'>
  <sift xmlns='urn:xmpp:sift:2'>
    <iq>
      <allow name='Envelope' ns='http://www.w3.org/2003/05/soap-
         envelope'/>
    </iq>
    <message>
      <allow name='Envelope' ns='http://www.w3.org/2003/05/soap-
         envelope'/>
    </message>
  </sift>
</iq>
```

Similarly, the following example shows how a client would filter inbound presence notifications to only receive notifications that contain entity capabilities data as specified in Entity Capabilities (XEP-0115) [12].

[10] XEP-0226: Message Stanza Profiles <https://xmpp.org/extensions/xep-0226.html>.
[11] XEP-0072: SOAP over XMPP <https://xmpp.org/extensions/xep-0072.html>.
[12] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

Listing 7: Sifting for entity capabilities

```
<iq from=’romeo@montague.lit/pda’
    id=’zl2f36d8’
    to=’romeo@montague.lit’
    type=’set’>
  <sift xmlns=’urn:xmpp:sift:2’>
    <presence>
      <allow name=’c’ ns=’http://jabber.org/protocol/caps’/>
    </presence>
  </sift>
</iq>
```

An even more strict example is filtering out all children of the XMPP <message/> stanza except <body/>.

Listing 8: Sifting Out All But Message Bodies

```
<iq from=’romeo@montague.lit/pda’
    id=’b8dv163d’
    to=’romeo@montague.lit’
    type=’set’>
  <sift xmlns=’urn:xmpp:sift:2’>
    <message>
      <allow name=’body’ ns=’jabber:client’/>
    </message>
  </sift>
</iq>
```

Naturally, the server could return the typical XMPP error conditions, such as <service-unavailable/> if the server does not support the SIFT protocol or the version specified by the client, <feature-not-implemented/> if the server does not support a particular feature (e.g., <iq/> sifting) requested by the client, <bad-request/> if the request is malformed, <internal-server-error/> if the server experiences a malfunction while attempting to process the request, and so on.

3.4 Disabling SIFT

To completely disable all SIFT processing, the client sends an empty <sift/> element.

Listing 9: Disabling all SIFT processing

```
<iq from=’romeo@montague.lit/pda’
    id=’mxi371g9’
    to=’romeo@montague.lit’
    type=’set’>
```

```
  <sift xmlns='urn:xmpp:sift:2'/>
</iq>
```

4 Business Rules

4.1 Handling IQ Stanzas

If the client does not request filtering of inbound IQ stanzas, the server MUST pass through to the client all IQ stanzas that are addressed to the full JID of the client (subject to appropriate security controls as defined in the relevant RFCs and XEPs).
If the client requests filtering of inbound IQ stanzas, for unfiltered payload name+namespace combinations the server MUST pass through to the client all IQ stanzas that are addressed to the full JID of the client (subject to appropriate security controls as defined in the relevant RFCs and XEPs), whereas for filtered payload name+namespace combinations the server MUST respond to all IQ stanzas in a way consistent with the specification for the given payload namespace (if defined) or as specified in XMPP-CORE and XMPP-IM for IQs where no full JID <localpart@domain.tld/resource> matches; typically that means returning a <service-unavailable/> error.

4.2 Handling Message Stanzas

When a client indicates that it wishes to receive messages, the server SHOULD deliver to the client all messages in the offline message queue and MUST deliver to the client any subsequent messages that would normally be delivered to the client in accordance with the rules defined in XMPP Core [13] and XMPP-IM.
If the client subsequently indicates that it wants the server to intercept inbound messages (and there are no other connected or available resources that have expressed interest in receiving inbound messages), the server SHOULD treat messages as if there were no connected or available resources (e.g., storing them offline for later delivery); if the client then indicates again that it wishes to receive inbound messages, the server SHOULD send those queued messages to the client so that it can get back in sync regarding messages received from its contacts.

4.3 Handling Presence Notifications

When the client indicates that it wishes to receive inbound presence notifications, the server SHOULD send outbound presence probes on the client's behalf. Responses to these presence probes are addressed to the bare JID of the account and then broadcasted to all of the resources that have expressed interest in receiving inbound presence notifications.
If the client subsequently indicates that it wants the server to intercept inbound presence notifications, the server MUST NOT deliver to the client presence notifications that are

[13] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

addressed to the bare JID or full JID as defined by the ’recipient’ attribute.
If the client then indicates again that it wishes to receive inbound presence notifications, the server shall resynchronize the client regarding the presence states of its contacts (how it does so is implementation-specific, e.g. whether it queues received presence notifications or re-probes the user’s contacts).

4.4 Handling Subscriptions

When the client indicates that it wishes to receive inbound subscription-related presence stanzas, it MUST deliver presence stanzas of type ”subscribe”, ”subscribed”, ”unsubscribe”, and ”unsubscribed” to the client in accordane with the rules in XMPP IM [14].
If the client subsequently indicates that it wants the server to intercept inbound subscription-related presence stanzas, the server MUST NOT deliver to the client such stanzas that are addressed to the bare JID or full JID as defined by the ’recipient’ attribute.

4.5 Lack of Sifting

Naturally, if the server advertises support for the SIFT protocol but the client does not send any IQ-set stanzas containing SIFT payloads, the server MUST proceed as it normally would in accordance with the core XMPP specifications.

5 Use Cases

5.1 Negative Presence Priority

XMPP-IM defines the concept of negative values for the presence <priority/> element, where a negative value instructs the server to not deliver to the client any messages that are directed to the bare JID of the user. This behavior can be emulated using SIFT by asking the server to intercept inbound message stanzas for the bare JID, but not presence notifications or IQ stanzas.

Listing 10: Emulating negative presence priority

```
<iq from='romeo@montague.lit/pda'
    id='zkd71d37'
    to='romeo@montague.lit'
    type='set'>
  <sift xmlns='urn:xmpp:sift:2'>
    <message recipient='bare'/>
  </sift>
```

[14] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

```
</iq>
```

If a client requests message sifting, but sends presence, it SHOULD specify a negative priority as a hint to contacts.

5.2 Presence Hush

Because inbound presence notifications can be ”chatty”, mobile clients and other entities with limited battery life might want to ”hush” the presence session by asking the server to intercept inbound presence notifications but not message stanzas.

Listing 11: Hushing the presence session

```
<iq from='romeo@montague.lit/pda'
    id='uh2s64g9'
    to='romeo@montague.lit'
    type='set'>
  <sift xmlns='urn:xmpp:sift:2'>
    <presence/>
  </sift>
</iq>
```

5.3 Removing Extraneous Message Extensions

Some XMPP-based services include a large number of extensions in messages (e.g., microblogging extensions). A client might want to filter out those extensions and allow only the bare minimum elements allowed by the base XMPP specifications. It can do so by specifying that the only payloads it wants to receive are the <body/>, <subject/>, and <thread/> elements qualified by the ’jabber:client’ namespace.

Listing 12: Removing extraneous message extensions

```
<iq from='romeo@montague.lit/pda'
    id='nj3fac27'
    to='romeo@montague.lit'
    type='set'>
  <sift xmlns='urn:xmpp:sift:2'>
    <message>
      <allow name='body' ns='jabber:client'/>
      <allow name='subject' ns='jabber:client'/>
      <allow name='thread' ns='jabber:client'/>
    </message>
  </sift>
</iq>
```

6 Security Considerations

To follow.

7 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [15].

8 XMPP Registrar Considerations

8.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:sift:2

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [16] shall add the foregoing namespace to the registry located at <https://xmpp.org/registrar/namespaces.html>, as described in Section 4 of XMPP Registrar Function (XEP-0053) [17].

8.2 Protocol Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of XEP-0053.

9 XML Schema

[15] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[16] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

[17] XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>.

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:sift:2'
    xmlns='urn:xmpp:sift:2'
    elementFormDefault='qualified'>

  <xs:element name='sift'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='iq'
                    type='siftElementType'
                    minOccurs='0'
                    maxOccurs='1'/>
        <xs:element name='message'
                    type='siftElementType'
                    minOccurs='0'
                    maxOccurs='1'/>
        <xs:element name='presence'
                    type='siftElementType'
                    minOccurs='0'
                    maxOccurs='1'/>
        <xs:element name='sub'
                    type='siftElementType'
                    minOccurs='0'
                    maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='siftElementType'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='allow'
                    type='allowElementType'
                    minOccurs='0'
                    maxOccurs='unbounded'/>
        <xs:any     namespace='##other'
                    minOccurs='0'
                    maxOccurs='unbounded'/>
      </xs:sequence>
      <xs:attribute name='recipient'
                    use='optional'
                    default='all'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='all'/>
            <xs:enumeration value='bare'/>
            <xs:enumeration value='full'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='sender'
                    use='optional'
                    default='all'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='all'/>
            <xs:enumeration value='local'/>
            <xs:enumeration value='others'/>
            <xs:enumeration value='remote'/>
            <xs:enumeration value='self'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

10 Acknowledgements

The authors wish to acknowledge feedback received from Dave Cridland, Jack Erwin, Fabio Forno, Waqas Hussain, Craig Kaes, Dirk Meyer, Chris Newton, Christopher Orr, Robert Quattlebaum, Travis Shirk, Mike Taylor, Matthew Wild, and Jiří Zárevúcký, as well as from participants at XMPP Summit #7 in July 2009 and XMPP Summit #8 in February 2010.