XEP-0033: Extended Stanza Addressing

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

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

2017-01-11
Version 1.2.1

Status	Type	Short Name
Draft	Standards Track	address

This specification defines an XMPP protocol extension that enables entities to include RFC822-style address headers within XMPP stanzas in order to specify multiple recipients or sub-addresses.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

On the existing Jabber network, there are many opportunities to optimize stanza traffic. For example, clients that want to send the same stanza to multiple recipients currently must send multiple stanzas. Similarly, when a user comes online the server sends many nearly-identical presence stanzas to remote servers.
The 'http://jabber.org/protocol/address' specification provides a method for both clients and servers to send a single stanza and have it be delivered to multiple recipients, similar to that found in RFC 822 [1]. As a side-effect, it also provides all of the functionality specified by the old 'jabber:x:envelope' [2] proposal, which this XEP can supersede.

2 Discovering Server Support

Support for Extended Stanza Addressing in a given server instance SHOULD be determined using Service Discovery (XEP-0030) [3]. A conforming server MUST respond to disco#info requests.

2.1 Disco to determine support

To determine if a server or service supports Extended Stanza Addressing, the requesting entity SHOULD send a disco#info request to it.

Listing 1: Disco request for address header support

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='multicast.montague.net'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

If the server supports Extended Stanza Addressing, it MUST include a "http://jabber.org/protocol/address" feature in the response.

Listing 2: Disco response for server supporting address headers

```
<iq type='result'
    from='multicast.montague.net'
    to='romeo@montague.net/orchard'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
```

[1] RFC 822: Standard for the Format of ARPA Internet Text Messages <http://tools.ietf.org/html/rfc0822>.
[2] jabber:x:envelope - Message Envelope Information Extension
[3] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

```
    ...
    <feature var='http://jabber.org/protocol/address'/>
    ...
  </query>
</iq>
```

2.2 Multicast service

The IM service MAY implement multicast directly, or it MAY delegate that chore to a separate service. A client can use the following approach to find a multicast-capable service hosted by its domain:

1. Send a disco#info request to the IM server; if its reply includes the 'http://jabber.org/protocol/address' feature, then it is a multicast-capable service.
2. If the IM server is not a multicast-capable service, send a disco#items request to the IM server; the IM server will then return a list of associated services.
3. Send a disco#info request to each of the services associated with the IM server; if one of the replies includes the 'http://jabber.org/protocol/address' feature, then that service is a multicast-capable service.

The multicast service MAY choose to limit which local users can use the service. The server MAY choose to limit whether non-local servers can send address headers that require the local server to send to third parties (relaying). In either case, if the server chooses to disallow the request, the server MUST return a Forbidden error (see the Error Conditions section below). In the relaying case, the server SHOULD NOT deliver to *any* of the addresses (even the local ones) if the sender is disallowed.

2.3 Caching

Implementations MAY choose to cache the disco response. Positive responses MAY be cached differently than negative responses. The result SHOULD NOT be cached for more than 24 hours, unless some sort of time-to-live information is added to the Service Discovery protocol in the future.

3 Outer Stanza

For multicast processing, the stanza containing an address header (the 'outer stanza') MUST be addressed to the multicast service, with no username or resource in the 'to' attribute.
When used for additional information in a one-to-one stanza (e.g. using the 'node' attribute), the outer stanza SHOULD be addressed directly to the recipient, not to the multicast service.

A multicast service MUST NOT change the 'from' address on the outer stanza. Note that this will limit third-party relaying across server-to-server connections as a side-effect.
Address headers MAY be included in message or presence stanzas. They MUST NOT be included as the direct child of an IQ stanza.

4 Addresses

Address values are packaged together into an <addresses/> element.

Listing 3: Message with an extended address

```
<message to='multicast.jabber.org'>
   <addresses xmlns='http://jabber.org/protocol/address'>
       <address type='to' jid='hildjj@jabber.org/Work' desc='Joe␣
           Hildebrand'/>
       <address type='cc' jid='jer@jabber.org/Home' desc='Jeremie␣
           Miller'/>
   </addresses>
   <body>Hello, world!</body>
</message>
```

Listing 4: Presence with an extended address

```
<presence from='hildjj@jabber.com' to='multicast.jabber.org' type='
    unavailable'>
    <addresses xmlns='http://jabber.org/protocol/address'>
        <address type='bcc' jid='temas@jabber.org'/>
        <address type='bcc' jid='jer@jabber.org'/>
    </addresses>
</presence>
```

Each address to which the sender wants the stanza to be re-sent will show up as an <address/> in the <addresses/> element. There are several different types of address, shown below.
An <address/> element MUST possess a 'type' attribute, and MUST possess at least one of the 'jid', 'uri', 'node', and 'desc' attributes. An <address/> element MUST NOT possess both a 'jid' attribute and a 'uri' attribute. If sending through a multicast service, an address MUST include a 'jid' or a 'uri' attribute, unless it is of type 'noreply'.

4.1 'jid' attribute

The 'jid' attribute is used to specify a simple Jabber ID associated with this address. If the 'jid' attribute is specified, the 'uri' attribute MUST NOT be specified. Support for the 'jid' attribute is REQUIRED.

4.2 'uri' attribute

The 'uri' attribute is used to specify an external system address, such as a sip:, sips:, or im: URI. If the 'uri' attribute is specified, the 'jid' and 'node' attributes MUST NOT be specified. These URIs MUST be formatted as specified in their respective RFCs, however with the characters & < > ' " replaced by their equivalent XML escapes, & < > ' ". If a receiving entity does not understand the given URI scheme, or if the URI is not formatted correctly, a "JID Malformed" error SHOULD be returned. Support for the 'uri' attribute is OPTIONAL.

4.3 'node' attribute

The 'node' attribute is used to specify a sub-addressable unit at a particular JID, corresponding to a Service Discovery node. A node attribute MAY be included if a 'jid' attribute is specified. If a 'uri' attribute is specified, a 'node' attribute MUST NOT be specified. Support for the 'node' attribute is RECOMMENDED.

4.4 'desc' attribute

The 'desc' attribute is used to specify human-readable information for this address. This data may be used by clients to provide richer address-book integration. This information is in the language of the sender, which MAY be identified using the standard xml:lang rules from XMPP Core [4]. Support for the 'desc' attribute is RECOMMENDED.

4.5 'delivered' attribute

When a multicast service delivers the stanza to a non-bcc address, it MUST add a delivered='true' attribute to the address element. A multicast service MUST NOT deliver to an address that was marked with a delivered='true' attribute when the service received the stanza. A multicast service SHOULD attempt to deliver to all addresses that are not marked with a delivered='true' attribute. The delivered attribute is used to prevent loops. See the Multicast Usage section below for more details. Support for the 'delivered' attribute is REQUIRED.

4.6 'type' attribute

The 'type' attribute is used to specify the semantics of a particular address. Support for the 'type' attribute is REQUIRED.

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

4.6.1 Address type='to'

These addressees are the primary recipients of the stanza.

4.6.2 Address type='cc'

These addressees are the secondary recipients of the stanza.

4.6.3 Address type='bcc'

These addressees should receive 'blind carbon copies' of the stanza. This means that the server MUST remove these addresses before the stanza is delivered to anyone other than the given bcc addressee or the multicast service of the bcc addressee.

4.6.4 Address type='replyto'

This is the address to which all replies are requested to be sent. Clients SHOULD respect this request unless an explicit override occurs. There MAY be more than one replyto or replyroom on a stanza, in which case the reply stanza MUST be routed to all of the addresses.

4.6.5 Address type='replyroom'

This is the JID of a Multi-User Chat (XEP-0045) [5] room to which responses should be sent. When a user wants to reply to this stanza, the client SHOULD join this room first. Clients SHOULD respect this request unless an explicit override occurs. There MAY be more than one replyto or replyroom on a stanza, in which case the reply stanza MUST be routed to all of the addresses.

4.6.6 Address type='noreply'

This address type contains no actual address information. Instead, it means that the receiver SHOULD NOT reply to the message. This is useful when broadcasting messages to many receivers.

4.6.7 Address type='ofrom'

This address type is used by Multi-User Chat (XEP-0045) [6] services. If a room is non-anonymous, the service MAY include an Extended Stanza Addressing (XEP-0033) [16] element

[5] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[6] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

that notes the original full JID of the sender by means of the ”ofrom” address type

4.7 Extensibility

As specified herein, the <address/> element is empty. Implementations or future protocols MAY extend the <address/> element for additional functionality, but any extensions are out of scope for this XEP. Such extensions SHOULD be appropriately qualified with a new namespace, and any extensions that are not understood by an implementation MUST be ignored.

Listing 5: Possible future extension

```
<message to='groups.jabber.org'>
  <addresses xmlns='http://jabber.org/protocol/address'>
    <address type='to' desc='Foo Group'>
      <group xmlns='some:funny:group:ns'>foo</group>
    </address>
    <address type='replyroom' jid='jdev@conference.jabber.org'/>
  </addresses>
</message>
```

5 Business Rules

5.1 Directed Presence

This specification can be used to, in effect, send directed presence (see Section 4.6 of RFC 6121 [7]). In order to ensure that entities that have received directed available presence through the service also are informed when the originating entity sends unavailable presence, the multicast service MUST do the following:

1. Keep track of the entities to which it sends available presence (i.e., presence stanzas with no 'type' attribute) along with the originating entity of that presence.
2. Upon receiving a presence stanza of type ”unavailable” from an originating entity, broadcast that unavailable presence to all entities to which it has send available presence.

In this way, the multicast service ensures that all entities which have received available presence through the service also receive the associated unavailable presence.

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

6 Multicast Usage

The following usage scenario shows how messages flow through both address-enabled and non-address-enabled portions of the Jabber network.
Note: the logic associated with *how* to perform the following tasks is purely informational. A conforming service MUST generate output as if these rules had been followed, but need not (and probably *will not*) use this algorithm.

1. User desires to send a stanza to more than one recipient.
2. Client determines the JID of a multicast service, using Service Discovery.
3. If no multicast service is found, the client MAY choose to deliver each stanza individually, or it MAY query each of the servers associated with desired recipients, and batch stanzas to those servers itself.
4. If a multicast service is found, the client constructs a stanza with an address block, and sends it to the multicast service. (Note: For the following rules, any address that was marked on the incoming address header with delivered='true' is never re-delivered.)
5. The server checks to see if it can deliver to all of the specified addresses. If not, the stanza is returned with a "Forbidden" error, and processing stops.
6. The server adds a delivered='true' attribute to all addresses.
7. The server removes all type='bcc' attributes.
8. The server delivers the stanza to all of the 'to', 'cc', and 'bcc' addresses from the original address header that are being handled locally. The server replaces the 'to' attribute on the outer stanza with the JID of each addressee. Each 'bcc' recipient MUST receive only the <address type='bcc'/> associated with that addressee.
9. For each non-local server (the 'target server') that has addresses specified in 'to', 'cc', or 'bcc' addresses in the original address header, the local server determines whether the target server supports multicast, using Service Discovery.
10. If the target server does not support address headers, the local server sends a copy of the stanza to each address, with the 'to' attribute on the outer stanza set to the JID of the given addressee.
11. If the target server does support address headers, the server removes the delivered='true' attributes on each of the addresses bound for that server, and replaces the 'to' attribute on the outer stanza with the adress of the multicast service for the target server. The 'bcc' addresses for the target server from the original address header are added back to the address header. A single stanza is sent to the target server.

7 Example Flow

Assume for these examples that header1.org and header2.org support address headers, and noheader.org does not.

Listing 6: Client checks local server for address header support

```
<iq type='get' to='header1.org' from='a@header1.org/work' id='id_1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 7: Client receives positive results from server

```
<iq type='result' from='header1.org' to='a@header1.org/work' id='id_1'
    >
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='http://jabber.org/protocol/address'/>
  </query>
</iq>
```

Listing 8: Client sends message to local server with address header

```
<message to='header1.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
     <address type='to'  jid='to@header1.org'/>
     <address type='cc'  jid='cc@header1.org'/>
     <address type='bcc' jid='bcc@header1.org'/>
     <address type='to'  jid='to@header2.org'/>
     <address type='cc'  jid='cc@header2.org'/>
     <address type='bcc' jid='bcc@header2.org'/>
     <address type='to'  jid='to@noheader.org'/>
     <address type='cc'  jid='cc@noheader.org'/>
     <address type='bcc' jid='bcc@noheader.org'/>
   </addresses>
   <body>Hello, World!</body>
</message>
```

Listing 9: Server delivers to local addresses

```
<message to='to@header1.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to' jid='to@header1.org' delivered='true'/>
      <address type='cc' jid='cc@header1.org' delivered='true'/>
      <address type='to' jid='to@header2.org' delivered='true'/>
      <address type='cc' jid='cc@header2.org' delivered='true'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
```

```
</message>
<message to='cc@header1.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
     <address type='to' jid='to@header1.org' delivered='true'/>
     <address type='cc' jid='cc@header1.org' delivered='true'/>
     <address type='to' jid='to@header2.org' delivered='true'/>
     <address type='cc' jid='cc@header2.org' delivered='true'/>
     <address type='to' jid='to@noheader.org' delivered='true'/>
     <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
<message to='bcc@header1.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
     <address type='to' jid='to@header1.org' delivered='true'/>
     <address type='cc' jid='cc@header1.org' delivered='true'/>
     <address type='bcc' jid='bcc@header1.org'/>
     <address type='to' jid='to@header2.org' delivered='true'/>
     <address type='cc' jid='cc@header2.org' delivered='true'/>
     <address type='to' jid='to@noheader.org' delivered='true'/>
     <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
```

Listing 10: Local server checks target server for address support

```
<iq type='get' to='header2.org' from='header1.org' id='id_2'>
   <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 11: Local server receives initial negative results from target server

```
<iq type='result' from='header2.org' to='header1.org' id='id_2'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
... no address feature ...
  </query>
</iq>
```

Listing 12: Local server checks target server for another component having address support

```
<iq type='get' to='header2.org' from='header1.org' id='id_3'>
   <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

Listing 13: Local server receives associated services from target server

```
<iq type='result' from='header2.org' to='header1.org' id='id_3'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='multicast.header2.org'
```

```
          name='Multicast_Service'/>
  </query>
</iq>
```

Listing 14: Local server checks associated services for address support

```
<iq type='get' to='multicast.header2.org' from='header1.org' id='id_4'
    >
   <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 15: Local server receives positive results from target server

```
<iq type='result' from='multicast.header2.org' to='header1.org/work'
    id='id_4'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='http://jabber.org/protocol/address'/>
  </query>
</iq>
```

Listing 16: Local server delivers to target server supporting address headers

```
<message to='multicast.header2.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to'  jid='to@header1.org' delivered='true'/>
      <address type='cc'  jid='cc@header1.org' delivered='true'/>
      <address type='to'  jid='to@header2.org'/>
      <address type='cc'  jid='cc@header2.org'/>
      <address type='bcc' jid='bcc@header2.org'/>
      <address type='to'  jid='to@noheader.org' delivered='true'/>
      <address type='cc'  jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
```

Listing 17: Target server delivers to its recipients

```
<message to='to@header2.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to' jid='to@header1.org'  delivered='true'/>
      <address type='cc' jid='cc@header1.org'  delivered='true'/>
      <address type='to' jid='to@header2.org'  delivered='true'/>
      <address type='cc' jid='cc@header2.org'  delivered='true'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
<message to='cc@header2.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
```

```
      <address type='to' jid='to@header1.org'  delivered='true'/>
      <address type='cc' jid='cc@header1.org'  delivered='true'/>
      <address type='to' jid='to@header2.org'  delivered='true'/>
      <address type='cc' jid='cc@header2.org'  delivered='true'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
<message to='bcc@header2.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to' jid='to@header1.org'  delivered='true'/>
      <address type='cc' jid='cc@header1.org'  delivered='true'/>
      <address type='to' jid='to@header2.org'  delivered='true'/>
      <address type='cc' jid='cc@header2.org'  delivered='true'/>
      <address type='bcc' jid='bcc@header2.org'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
```

Listing 18: Local server checks target server for address header support

```
<iq type='get' to='noheader.org' from='header1.org' id='id_5'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 19: Local server receives negative results from target server (assume no associated services found)

```
<iq type='result' from='noheader.org' to='header1.org' id='id_5'>
   <query xmlns='http://jabber.org/protocol/disco#info'>
... no address feature ...
   </query>
</iq>
```

Listing 20: Server delivers to each address on the target server not supporting address headers

```
<message to='to@noheader.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to' jid='to@header1.org'  delivered='true'/>
      <address type='cc' jid='cc@header1.org'  delivered='true'/>
      <address type='to' jid='to@header2.org'  delivered='true'/>
      <address type='cc' jid='cc@header2.org'  delivered='true'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
```

```
<message to='cc@noheader.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to' jid='to@header1.org'  delivered='true'/>
      <address type='cc' jid='cc@header1.org'  delivered='true'/>
      <address type='to' jid='to@header2.org'  delivered='true'/>
      <address type='cc' jid='cc@header2.org'  delivered='true'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
   </addresses>
   <body>Hello, World!</body>
</message>
<message to='bcc@noheader.org' from='a@header1.org/work'>
   <addresses xmlns='http://jabber.org/protocol/address'>
      <address type='to' jid='to@header1.org'  delivered='true'/>
      <address type='cc' jid='cc@header1.org'  delivered='true'/>
      <address type='to' jid='to@header2.org'  delivered='true'/>
      <address type='cc' jid='cc@header2.org'  delivered='true'/>
      <address type='to' jid='to@noheader.org' delivered='true'/>
      <address type='cc' jid='cc@noheader.org' delivered='true'/>
      <address type='bcc' jid='bcc@noheader.org'/>
   </addresses>
   <body>Hello, World!</body>
</message>
```

8 Reply Handling

When replying to a message stanza that contains an extended address, the following rules apply:

1. If a noreply address is specified, a reply SHOULD NOT be generated.
2. If one or more replyroom addresses are specified, the client SHOULD join the specified chat rooms instead of replying directly to the specified users. No further extended address processing is required.
3. If one or more replyto address are specified, the reply SHOULD go to the specified addresses. No further extended address processing is required. Any <thread/> element from the initial message MUST be copied into the replies.
4. Otherwise, an extended-address aware client MUST copy the address header from the original message into the reply, removing any delivered attributes. If the original sender is not in the copied list, the original sender MUST be added as a 'to' address. The recipient's address SHOULD be removed from the list. The client then proceeds with the rules from the Multicast Usage section above for delivery of the message.

9 Error Conditions

The following error conditions are to be used by implementations (for further information regarding error syntax, see Error Condition Mappings (XEP-0086) [8]):

XMPP Condition	Purpose
<forbidden/>	The sending user does not have permission to use this multicast service.
<jid-malformed/>	A URI attribute was invalid or not understood (note that support for the 'uri' attribute is optional).
<not-acceptable/>	Too many receiver fields were specified. Servers SHOULD have a configurable upper limit for the number of addresses. The limit SHOULD be more than 20 and less than 100.

10 Security Considerations

A recipient SHOULD trust a stanza's extended addressing headers only as much as it trusts the sender of the stanza.
Furthermore, there exists the potential for abuse related to the 'replyto' and 'replyroom' features (e.g., an entity could send messages with 'replyroom' set to the address of a room that hosts salacious content or with 'replyto' set to the address of a spambot that harvests Jabber addresses). Therefore if a human user's receiving application receives a message with extended stanza addressing that specifies a 'replyto' or 'replyroom' address other than that of the sender, it SHOULD inform the user of that fact. (Naturally, the receiving application MAY also limit the entities to which the recipient can reply using privacy lists as specified in XMPP IM [9].)

11 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [10].

[8] XEP-0086: Error Condition Mappings <https://xmpp.org/extensions/xep-0086.html>.
[9] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[10] 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/>.

12 XMPP Registrar Considerations

The XMPP Registrar [11] shall include 'http://jabber.org/protocol/address' in its registry of protocol namespaces.

13 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/address'
    xmlns='http://jabber.org/protocol/address'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0033: http://www.xmpp.org/extensions/xep-0033.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='addresses'>
     <xs:complexType>
        <xs:sequence>
           <xs:element ref='address'
                       minOccurs='1'
                       maxOccurs='unbounded'/>
        </xs:sequence>
     </xs:complexType>
  </xs:element>

  <xs:element name='address'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='delivered' use='optional' fixed='true'/>
          <xs:attribute name='desc' use='optional' type='xs:string'/>
          <xs:attribute name='jid' use='optional' type='xs:string'/>
          <xs:attribute name='node' use='optional' type='xs:string'/>
          <xs:attribute name='type' use='required'>
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
```

[11] 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/>.

```
                <xs:enumeration value='bcc'/>
                <xs:enumeration value='cc'/>
                <xs:enumeration value='noreply'/>
                <xs:enumeration value='replyroom'/>
                <xs:enumeration value='replyto'/>
                <xs:enumeration value='to'/>
                <xs:enumeration value='ofrom'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name='uri' use='optional' type='xs:string'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

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

</xs:schema>
```

14 Acknowledgements

Sections of this document were inspired by RFC 822.