XEP-0313: Message Archive Management

Matthew Wild
mailto:mwild1@gmail.com
xmpp:me@matthewwild.co.uk

Kevin Smith
mailto:kevin@kismith.co.uk
xmpp:kevin@doomsong.co.uk

2018-07-16
Version 0.6.3

Status	Type	Short Name
Experimental	Standards Track	mam

This document defines a protocol to query and control an archive of messages stored on a server.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

It is a common desire for users of XMPP to want to store their messages in a central archive on their server. This feature allows them to record conversations that take place on clients that do not support local history storage, to synchronise conversation history seamlessly between multiple clients, to read the history of a MUC room, or to view old items in a pubsub node.

2 Requirements

As this extension aims to make things easy for client developers, some research was made into the way clients handle history today. The resulting protocol was designed to allow for the following primary usage scenarios:

- Automatic history synchronization between multiple clients.
- Calendar-based on-demand display of historic messages in a client that doesn't keep local history.
- So-called 'infinite' scrollback, whereby a client automatically fetches and displays historical messages naturally in the message log as the user scrolls back in time.

Another extension for archiving already exists in XMPP, Message Archiving (XEP-0136) [1]). However implementation experience has shown that the protocol defined therein supports rather more functionality than is typically needed for the above uses, and is significantly more effort to implement.
This specification aims to define a much simpler and modular protocol for working with a server-side message store. Through this it is hoped to boost implementation and deployment of archiving in XMPP. It should be noted that (although not required) a server is free to implement XEP-0136 alongside this protocol if it so chooses, though a mapping between both protocols is beyond the scope of this specification.
Notable functionality in XEP-0136 that is intentionally not defined by this specification for simplicity:

- Support for 'collections'. Few clients even support this concept for local history storage, and it is possible to apply the logic for splitting a stream of messages into conversations on the client-side, thus greatly simplifying the protocol.
- Support for uploading to the archive. On the assumption that a server automatically archives messages to and from the client, it is typically rare for a client to end up with messages that are not already in the archive. Uploading could be useful for bootstrapping an empty archive however, and may be defined in a future specification if there is demand for such functionality.

[1] XEP-0136: Message Archiving <https://xmpp.org/extensions/xep-0136.html>.

- Support for 'off the record' chats (OTR; not to be confused with the encryption algorithm of the same name). This allowed complex negotiation for either the user or contact to command specific conversations to bypass the archive. In reality the archiving behaviour of a contact's server cannot be enforced (they could ignore the OTR request and archive the messages anyway without your consent) so this specification does not try and regulate that. Equivalent functionality can be implemented with server logic for not including encrypted or specially-tagged messages in the archive (out of scope for this specification).
- Support per-session configuration. This feature was deemed unnecessary for the majority of implementations, and hence the configuration protocol in this specification is much simplified, which allows for a simple user interface in clients. Advanced configuration however may be performed through ad-hoc commands depending on the server implementation.

3 Message archives

An archive contains a collection of messages relevant to a particular XMPP address, e.g. a user, MUC, pubsub node, server. Note: while a service might have many "archives" as defined here (one per JID capable of being queried) this is a conceptual distinction, and a server is not bound to any particular implementation or arrangement of data stores.
Exactly which messages a server archives is up to implementation and deployment policy, but it is expected that all messages that hold meaningful content, rather than state changes such as Chat State Notifications, would be archived. Rules are specified later in this document.
A stored message consists of at least the following pieces of information:

- A timestamp of when the message was sent (for an outgoing message) or received (for an incoming message).
- The remote JID that the stanza is to (for an outgoing message) or from (for an incoming message).
- A server-assigned UID that MUST be unpredictable and unique within the archive.
- The message stanza itself. The entire original stanza SHOULD be stored, but at a minimum only the <body/> tag MUST be preserved (ie. the server might, at its discretion, strip certain extensions from messages before storage).

Note that 'incoming' and 'outgoing' messages are viewed within the context of the archived JID, rather than the system as a whole. For example, if romeo@montegue.lit sent a message to juliet@capulet.lit, it would be an outgoing message in the context of archiving for Romeo, and an incoming message in the context of archiving for Juliet.

3.1 Order of messages

Order within the archive MUST be preserved, where the order of messages is the same as the order that the client originally received them (or would have received them if online). Throughout this document the term 'chronological order' refers to this order, however implementors should take care not to rely on timestamps alone for ordering messages, as multiple messages may share the same timestamp.

3.2 Message retention and deletion

A server MAY impose limits on the size of an individual archive. For example a server might begin to discard old messages once the archive reaches a certain size, or only keep messages until they reach a certain age. Any such deleted messages MUST be the oldest in the archive, i.e. it is not permitted to create gaps or "holes" in the archive. The UIDs of deleted messages MUST NOT be reused for new messages.
However a server that wishes to remove messages from the middle of an archive, e.g. to remove accidentally transmitted sensitive information may omit the <message> stanza from inside the <forwarded> element or replace the message with an appropriate placeholder when transmitting the result in response to a query. However servers MUST retain the UID, timestamp and JID of the original message internally to ensure that all queries remain consistent. It should also be understood that clients maintaining their own local copy of the archive may still retain the original message locally in this case, and this protocol provides no mechanism for forcibly removing messages from any local archive or cache that clients may keep.

3.3 Archiving entities

There is no restriction on which services can expose archives, although only user, MUC and pubsub node archives are discussed here.

3.3.1 User archives

The most typical address is that of a user's own bare JID, within which those messages sent to or from that user's account would generally automatically be stored by the server. The collection is ordered chronologically by the time each message was sent/received.
Servers that expose archive messages of sent/received messages on behalf of local users MUST expose these archives to the user on the user's bare JID.

3.3.2 MUC archives

A MUC service allowing MAM queries for a room MUST expose the MAM archive on the room's bare JID

3.3.3 Pubsub node archives

A pubsub service allowing MAM queries for a node's data MUST expose this for queries addressed to the pubsub service

3.4 Querying Entities

While this document talks about 'clients' and 'servers', as these are the common cases, the querying entity (referred to as a 'client') need not be an XMPP client as defined by RFC6120, but could potentially be any type of entity, and the queried entity (referred to as a 'server') need not be an XMPP server as defined by RFC6120, although access controls might prohibit any given entity from being able to access an archive.

3.5 Communicating the archive ID

When a message is archived, the server MUST add an <stanza-id/> element as defined in Unique and Stable Stanza IDs (XEP-0359) [2] to the message, which informs the recipient of where and under what ID the message is stored. When doing this the server MUST follow the business rules defined in XEP-0359. The 'by' attribute MUST be set to the address of the archive. For regular users that's the bare JID of the account and for MUC that's the bare JID of the room.
Servers MUST NOT include the <stanza-id/> element in messages addressed to JIDs that do not have permissions to access the archive, such as a users's outgoing messages to their contacts. However servers SHOULD include the element as a child of the forwarded message when using Message Carbons (XEP-0280) [3]

Listing 1: Client receives a message that has been archived

```
<message to='juliet@capulet.lit/balcony'
         from='romeo@montague.lit/orchard'
         type='chat'>
  <body>Call me but love, and I'll␣be␣new␣baptized;␣Henceforth␣I␣never
     ␣will␣be␣Romeo.</body>
␣␣<stanza-id␣xmlns='urn:xmpp:sid:0'␣by='juliet@capulet.lit'␣id='
    28482-98726-73623'␣/>
</message>
```

[2] XEP-0359: Unique and Stable Stanza IDs <https://xmpp.org/extensions/xep-0359.html>.
[3] XEP-0280: Message Carbons <https://xmpp.org/extensions/xep-0280.html>.

Note: Previous versions of this protocol did not specify any interaction with stanza-id, and clients MUST NOT interpret XEP-0359 IDs in messages as archive IDs unless the server advertises support for 'urn:xmpp:mam:2' specifically.

4 Querying an archive

An entity is able to query (subject to appropriate access rights) an archive for all messages within a certain timespan, optionally restricting results to those to/from a particular JID. To allow limiting the results or paging through them a client may use Result Set Management (XEP-0059) [4], which MUST be supported by both the client and the server.
A query consists of an <iq/> stanza of type 'set' addressed to the account or server entity hosting the archive, with a 'query' payload. On receiving the query, the server pushes to the client a series of messages in chronological order from the archive that match the client's given criteria. After the results it then returns the <iq/> result to indicate that the query is completed.
The final <iq/> result response MUST include an RSM <set/> element, wrapped into a <fin/> element qualified by the 'urn:xmpp:mam:2' namespace, indicating the UID of the first and last message of the (possibly limited) result set. This allows clients to accurately page through messages.

Listing 2: A user queries their archive for messages

```
<iq type='set' id='juliet1'>
  <query xmlns='urn:xmpp:mam:2' queryid='f27' />
</iq>
```

Listing 3: Their server sends the matching messages

```
<message id='aeb213' to='juliet@capulet.lit/chamber'>
  <result xmlns='urn:xmpp:mam:2' queryid='f27' id='28482-98726-73623'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <delay xmlns='urn:xmpp:delay' stamp='2010-07-10T23:08:25Z'/>
      <message xmlns='jabber:client' from="witch@shakespeare.lit" to="
          macbeth@shakespeare.lit">
        <body>Hail to thee</body>
      </message>
    </forwarded>
  </result>
</message>
```

Listing 4: Server returns the result IQ to signal the end

```
<iq type='result' id='juliet1'>
  <fin xmlns='urn:xmpp:mam:2'>
```

[4] XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.

```
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>28482-98726-73623</first>
      <last>09af3-cc343-b409f</last>
    </set>
  </fin>
</iq>
```

To ensure that the client knows when the results are complete, the server MUST send the <iq/> result after last query result has been sent to the client. The client can optionally include a 'queryid' attribute in their query, which allows the client to match results to their initiating query.
When querying a pubsub node's archive, the 'node' attribute is added to the <query> element.

Listing 5: A user queries a pubsub node's archive for messages

```
<iq to='pubsub.shakespeare.lit' type='set' id='juliet1'>
  <query xmlns='urn:xmpp:mam:2' queryid='f28' node='fdp/submitted/
      capulet.lit/sonnets'/>
</iq>
```

4.1 Filtering results

By default all messages match a query, and filters are used to request a subset of the archived messages. Filters are specified in a Data Forms (XEP-0004) [5] data form included with the query. The hidden FORM_TYPE field MUST be set to this protocol's namespace, 'urn:xmpp:mam:2'. Three further fields are defined by this XEP and MUST be supported by servers, though all of them are optional for the client. These fields are:

- start
- end
- with

Other fields may be used, but are not defined in this document - the naming of new fields MUST be consistent with the format defined in Field Standardization for Data Forms (XEP-0068) [6]. Servers MUST NOT mark any fields in the form as being required (i.e. with the data forms <required/> element), regardless of whether they are defined in this document or elsewhere.

[5] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.
[6] XEP-0068: Field Data Standardization for Data Forms <https://xmpp.org/extensions/xep-0068.html>.

4.1.1 Filtering by JID

If a 'with' field is present in the form, it contains a JID against which to match messages. The server MUST only return messages if they match the supplied JID. A message in a user's archive matches if the JID matches either the to or from of the message. An item in a pubsub or MUC archive matches if the publisher of the item matches the JID; note that this should only be available to entities that would already have been allowed to know the publisher of the events (e.g. this could not be used by a visitor to a semi-anonymous MUC).
If the 'with' field's value is the bare JID of the archive, the server must only return results where both the 'to' and 'from' match the bare JID (either as bare or by ignoring the resource), as otherwise every message in the archive would match
If 'with' is omitted, the server MUST match all messages in the selected timespan with the query, regardless of the to/from addresses on each message.

Listing 6: Querying for all messages to/from a particular JID

```
<iq type='set' id='juliet1'>
  <query xmlns='urn:xmpp:mam:2'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mam:2</value>
      </field>
      <field var='with'>
        <value>juliet@capulet.lit</value>
      </field>
    </x>
  </query>
</iq>
```

If (and only if) the supplied JID is a bare JID (i.e. no resource is present), then the server SHOULD return messages if their bare to/from address for a user archive, or from address otherwise, would match it. For example, if the client supplies a 'with' of "juliet@capulet.lit" a query to their own archive would also match messages to or from "juliet@capulet.lit/balcony" and "juliet@capulet.lit/chamber".

4.1.2 Filtering by time received

The 'start' and 'end' fields, if provided, MUST contain timestamps formatted according to the DateTime profile defined in XMPP Date and Time Profiles (XEP-0082) [7]
The 'start' field is used to filter out messages before a certain date/time. If specified, a server MUST only return messages whose timestamp is equal to or later than the given timestamp.
If omitted, the server SHOULD assume the value of 'start' to be equal to the date/time of the earliest message stored in the archive.
Conversely, the 'end' field is used to exclude from the results messages after a certain point in

[7] XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.

time. If specified, a server MUST only return messages whose timestamp is equal to or earlier than the timestamp given in the 'end' field.
If omitted, the server SHOULD assume the value of 'end' to be equal to the date/time of the most recent message stored in the archive.

Listing 7: Querying the archive for all messages in a certain timespan

```
<iq type='set' id='juliet1'>
  <query xmlns='urn:xmpp:mam:2'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mam:2</value>
      </field>
      <field var='start'>
        <value>2010-06-07T00:00:00Z</value>
      </field>
      <field var='end'>
        <value>2010-07-07T13:23:54Z</value>
      </field>
    </x>
  </query>
</iq>
```

Listing 8: Querying the archive for all messages after a certain time

```
<iq type='set' id='juliet1'>
  <query xmlns='urn:xmpp:mam:2'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mam:2</value>
      </field>
      <field var='start'>
        <value>2010-08-07T00:00:00Z</value>
      </field>
    </x>
  </query>
</iq>
```

4.1.3 Limiting results

Finally, in order for the client or server to limit the number of results transmitted at a time a server MUST support Result Set Management (XEP-0059) [8] and MUST support the paging mechanism defined therein. A client MAY include a <set/> element in its query.
For the purposes of this protocol, the UIDs used by RSM correspond with the UIDs of the stanzas stored in the archive.

[8] XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.

Listing 9: A query using Result Set Management

```
<iq type='set' id='q29302'>
  <query xmlns='urn:xmpp:mam:2'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:mam:2</value>
      </field>
      <field var='start'>
        <value>2010-08-07T00:00:00Z</value>
      </field>
    </x>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>10</max>
    </set>
  </query>
</iq>
```

To conserve resources, a server MAY place a reasonable limit on how many stanzas may be pushed to a client in one request. Whether or not the client query included a <set/> element, the server MAY simply return its limited results, modifying the <set/> element it returns appropriately.

Listing 10: Server responds to client with limited results using RSM

```
<!-{}- result messages -{}->
<iq type='result' id='q29302'>
  <fin xmlns='urn:xmpp:mam:2'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>28482-98726-73623</first>
      <last>09af3-cc343-b409f</last>
      <count>20</count>
    </set>
  </fin>
</iq>
```

The <first> and <last> elements specify the UID of the first and last returned results (not necessarily of all the messages that matched the query, if the results have been limited).
The RSM <count> element and the 'index' attribute on the RSM <first> element are optional, but servers SHOULD include them. Please refer to the RSM specification for more information surrounding their meaning and use.

4.1.4 Paging through results

Having previously made a query that returned results limited by the server (as described above), a client can re-send the same request and receive the next 'page' of results. It does this by including a <set> element with its request, containing an <after/> with the UID of the

last message it received from the previous query.

Listing 11: A page query using Result Set Management

```
<iq type='set' id='q29303'>
  <query xmlns='urn:xmpp:mam:2'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'><value>urn:xmpp:mam:2</
            value></field>
        <field var='start'><value>2010-08-07T00:00:00Z</value></field>
      </x>
      <set xmlns='http://jabber.org/protocol/rsm'>
         <max>10</max>
         <after>09af3-cc343-b409f</after>
      </set>
  </query>
</iq>
```

Note: There is no concept of an "open query", and servers MUST be prepared to receive arbitrary page requests at any time.
If the UID contained within an <after> or <before> element is not present in the archive, the server MUST return an item-not-found error in response to the query.
When the results returned by the server are complete (that is: when they have not been limited by the maximum size of the result page (either as specified or enforced by the server)), the server MUST include a 'complete' attribute on the <fin> element, with a value of 'true'; this informs the client that it doesn't need to perform further paging to retreive the requested data. If it is not the last page of the result set, the server MUST either omit the 'complete' attribute, or give it a value of 'false'.

Listing 12: Server completes a result with the last page of messages

```
<!-{}- result messages -{}->
<iq type='result' id='u29303'>
  <fin xmlns='urn:xmpp:mam:2' complete='true'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>23452-4534-1</first>
      <last>390-2342-22</last>
      <count>16</count>
    </set>
  </fin>
</iq>
```

Sometimes (e.g. due to network or storage partitioning, or other transient errors) the server might return results to a client that are unstable (e.g. they might later change in sequence or content). In such a situation the server MUST stamp the <fin> element with a 'stable' attribute with a value of 'false'. If the server knows that the data it's serving are stable it MUST either stamp a 'stable' attribute with a value of 'true', or no such attribute. An example of when

unstable might legitimately be returned is if the MAM service uses a clustered data store and a query covers a time period for which the data store has not yet converged; it the server could return best-guess results and tell the client that they may be unstable. A client SHOULD NOT cache unstable results long-term without later confirming (by reissuing appropriate queries) that they have become stable.

4.1.5 Retrieving form fields

In order for the client find out about additional fields the server might support, it can send an iq stanza of type 'get' addressed to the archive like this:

Listing 13: Client requests supported query fields

```
<iq type='get' id='form1'>
  <query xmlns='urn:xmpp:mam:2'/>
</iq>
```

The server replies with all the form fields it supports in queries, which MUST include the mandatory fields specified in this document.

Listing 14: Server returns supported fields

```
<iq type='result' id='form1'>
  <query xmlns='urn:xmpp:mam:2'>
    <x xmlns='jabber:x:data' type='form'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:mam:2</value>
      </field>
      <field type='jid-single' var='with'/>
      <field type='text-single' var='start'/>
      <field type='text-single' var='end'/>
      <field type='text-single' var='urn:example:xmpp:free-text-search
          '/>
      <field type='text-single' var='urn:example:xmpp:stanza-content'/
          >
    </x>
  </query>
</iq>
```

If the client understands any of the additional fields it MAY proceed to include any of them in subsequent queries. It is not required to include any or all of the supported fields in queries.

Listing 15: Client uses two discovered query fields in a query

```
<iq type='set' id='query4'>
  <query xmlns='urn:xmpp:mam:2'>
```

```
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>urn:xmpp:mam:2</value>
      </field>
      <field type='text-single' var='urn:example:xmpp:free-text-search
          '>
        <value>Where arth thou, my Juliet?</value>
      </field>
      <field type='text-single' var='urn:example:xmpp:stanza-content'>
        <value>{http://jabber.org/protocol/mood}mood/lonely</value>
      </field>
    </x>
  </query>
</iq>
```

Note that as the 'with', 'start' and 'end' fields MUST be implemented by servers, clients are able to submit forms using combinations of only these fields without needing to first fetch the form from the server and the types of these fields MUST be 'jid-single', 'text-single' and 'text-single' respectively. A server MUST NOT rely on a client having first requested the form before submitting queries

4.2 Query results

The server responds to the archive query by transmitting to the client all the messages that match the criteria the client requested, subject to implementation limits. The results are sent as individual stanzas, with the original message encapsulated in a <forwarded/> element as described in Stanza Forwarding (XEP-0297) [9].
The result messages MUST contain a <result/> element with an 'id' attribute that gives the current message's archive UID (archived messages MAY also contain a XEP-0359 <stanza-id> element, but clients MUST NOT depend on it). If the client gave a 'queryid' attribute in its initial query, the server MUST also include that in this result element.
The <result/> element contains a <forwarded/> element which SHOULD contain the original message as it was received, and SHOULD also contain a <delay/> element qualified by the 'urn:xmpp:delay' namespace specified in Delayed Delivery (XEP-0203) [10]. The value of the 'stamp' attribute MUST be the time the message was originally received by the forwarding entity.
The archive results MUST be sorted in chronological order, both within the returned results and within the ordering of RSM such that if a client were to request the first 10 stanzas in an archive, then use RSM to request the next 10 stanzas (by providing the 'after' element with the UID of the 10th stanza in the first results) all 20 result stanzas would be received in chronological order.

[9] XEP-0297: Stanza Forwarding <https://xmpp.org/extensions/xep-0297.html>.
[10] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.

Listing 16: Server returns two matching messages

```
<message id='aeb213' to='juliet@capulet.lit/chamber'>
  <result xmlns='urn:xmpp:mam:2' queryid='f27' id='28482-98726-73623'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <delay xmlns='urn:xmpp:delay' stamp='2010-07-10T23:08:25Z'/>
      <message xmlns='jabber:client'
        to='juliet@capulet.lit/balcony'
        from='romeo@montague.lit/orchard'
        type='chat'>
        <body>Call me but love, and I'll␣be␣new␣baptized;␣Henceforth␣I
           ␣never␣will␣be␣Romeo.</body>
␣␣␣␣␣␣</message>
␣␣␣␣</forwarded>
␣␣</result>
</message>

<message␣id='aeb214'␣to='juliet@capulet.lit/chamber'>
␣␣<result␣xmlns='urn:xmpp:mam:2'␣queryid='f27'␣id='5d398-28273-f7382'>
␣␣␣␣<forwarded␣xmlns='urn:xmpp:forward:0'>
␣␣␣␣␣␣<delay␣xmlns='urn:xmpp:delay'␣stamp='2010-07-10T23:09:32Z'/>
␣␣␣␣␣␣<message␣xmlns='jabber:client'
␣␣␣␣␣␣␣␣␣␣to='romeo@montague.lit/orchard'
␣␣␣␣␣␣␣␣␣␣from='juliet@capulet.lit/balcony'
␣␣␣␣␣␣␣␣␣␣type='chat'␣id='8a54s'>
␣␣␣␣␣␣␣␣<body>What␣man␣art␣thou␣that␣thus␣bescreen'd in night so
    stumblest on my counsel?</body>
      </message>
    </forwarded>
  </result>
</message>
```

5 Business Rules

5.1 Storage and Retrieval Rules

Different entities will have different requirements for which data are stored, as might different deployments. This section provides general rules within which a server will act. While there may be local policy restrictions that prevent archiving of some aspects discussed here, this is a RECOMMENDED baseline. A server MAY implement any subset of possible archives for JIDs it controls (although it MUST advertise support only for those JIDs that support it).
No requirements are placed on how a server implements its storage beyond that it has to store data sufficient to be able to comply with this document. When this document describes storage requirements (e.g. MUST NOT store more than one copy...), it refers to what would appear to have been stored in order to satisfy the query.
If an entity (user's server, MUC room, pubsub node, ...) rejects an incoming message (such as from an occupant not allowed to send messages to the room, a user not authorized to publish

to a pubsub node, a contact blocked by the user etc.) that message should not appear in the archive for the entity that rejected it - the archive should represent what logical entities (MUC occupants, users, pubsub subscribers...) would have received, and so only contain messages accepted for delivery to such entities.

5.1.1 User Archives

A user archive is anticipated to provide the user with the ability to access their prior conversations. To this end, a server SHOULD include in a user archive all of the messages a user sends or receives of type 'normal' or 'chat' that contain a <body> element. A server SHOULD also include messages of type 'groupchat' that have a <body>, but where such history is accessible through another method (e.g. through an archive on the MUC JID), a server MAY exclude these from the archive. A server MAY include additional non-conversation messages. A server MAY include messages of type 'headline', but this is not generally suggested.
At a minimum, the server MUST store the <body> elements of a stanza. It is suggested that other elements that are used in a given deployment to supplement conversations (e.g. XHTML-IM payloads) are also stored. Other elements MAY be stored.
If a server supports mechanisms that multiply copies of a stanza (e.g. Carbons, or forking a stanza to a bare JID), it MUST store such a staza within a given archive only once, irrespective of multiple connected clients receiving copies

5.1.2 MUC Archives

A MUC archives allows a user to view the conversation within a room. All messages sent to the room that contain a <body> element SHOULD be stored, as should subject change stanzas, apart from those messages that the room rejects.
A MUC archive MUST store each message only once (not, for example, every copy sent out to an occupant).
A MUC archive MUST NOT include 'private message' results (those sent directly between occupants, not shared in the room) in the results
A MUC archive MUST check that the user requesting the archive has the right to enter it at the time of the query and only allow access if so. In a members-only chat room, only owners, admins or members can query a room archive. In the case of open MUC rooms, the MUC archives can generally be accessed by any users (including those who have never entered the room) who do not have an affiliation of 'outcast', but a MUC archive MAY further limit access based on other criteria as part of the deployment policy. A MUC archive MAY, if it stores historical data about previous configuration states, limit the results returned to only those that the querying user would have been authorised to see at the time (e.g. it MAY limit the results to not include results while a user was an outcast).
When sending out the archives to a requesting client, the forwarded stanza MUST NOT have a 'to' attribute, and the 'from' MUST be the occupant JID of the sender of the archived message. In the case of non-anonymous rooms or if the recipient of the MUC archive has the right to

access the sender real JID at the time of the query, the archive message will use extended message information in an <x/> element qualified by the 'http://jabber.org/protocol/muc#user' namespace and containing an <item/> child with a 'jid' attribute specifying the occupant's full JID, as defined for non-anonymous room presence in Multi-User Chat (XEP-0045) [11]. The archiving entity MUST strip any pre-existing <x> element from MUC messages (as MUC rooms are not required to do this).

Listing 17: Server returns MUC messages

```
<message id='iasd207' from='coven@chat.shakespeare.lit' to='
    hag66@shakespeare.lit/pda'>
  <result xmlns='urn:xmpp:mam:2' queryid='g27' id='34482-21985-73620'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <delay xmlns='urn:xmpp:delay' stamp='2002-10-13T23:58:37Z'/>
      <message xmlns="jabber:client"
        from='coven@chat.shakespeare.lit/firstwitch'
        id='162BEBB1-F6DB-4D9A-9BD8-CFDCC801A0B2'
        type='groupchat'>
        <body>Thrice the brinded cat hath mew'd.</body>
␣␣␣␣␣␣␣␣<x␣xmlns='http://jabber.org/protocol/muc#user'>
␣␣␣␣␣␣␣␣␣␣<item␣affiliation='none'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣jid='witch1@shakespeare.lit'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣role='participant'␣/>
␣␣␣␣␣␣␣␣</x>
␣␣␣␣␣␣</message>
␣␣␣␣</forwarded>
␣␣</result>
</message>

<message␣id='iasd207'␣from='coven@chat.shakespeare.lit'␣to='
    hag66@shakespeare.lit/pda'>
␣␣<result␣xmlns='urn:xmpp:mam:2'␣queryid='g27'␣id='34482-21985-73620'>
␣␣␣␣<forwarded␣xmlns='urn:xmpp:forward:0'>
␣␣␣␣␣␣<delay␣xmlns='urn:xmpp:delay'␣stamp='2002-10-13T23:58:43Z'/>
␣␣␣␣␣␣<message␣xmlns="jabber:client"
␣␣␣␣␣␣␣␣from='coven@chat.shakespeare.lit/secondwitch'
␣␣␣␣␣␣␣␣id='90057840-30FD-4141-AA44-103EEDF218FC'
␣␣␣␣␣␣␣␣type='groupchat'>
␣␣␣␣␣␣␣␣<body>Thrice␣and␣once␣the␣hedge-pig␣whined.</body>
␣␣␣␣␣␣␣␣<x␣xmlns='http://jabber.org/protocol/muc#user'>
␣␣␣␣␣␣␣␣␣␣<item␣affiliation='none'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣jid='witch2@shakespeare.lit'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣role='participant'␣/>
␣␣␣␣␣␣␣␣</x>
␣␣␣␣␣␣</message>
␣␣␣␣</forwarded>
␣␣</result>
```

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

```
</message>
```

5.1.3 Pubsub Archives

A PubSub service offering MAM SHOULD store each of the items published to each node. When responding to MAM requests it MUST construct the message stanza within the <forwarded> element in the same manner as the notifications sent to subscribers for the item, except that specifying the 'from' 'to' and 'id' attributes are OPTIONAL. Pubsub items must be returned one per message stanza (i.e. there MUST NOT be multiple <item> elements within the <items> element).

Listing 18: Server returns a pubsub messages

```
<message id='iasd208' to='juliet@capulet.lit/chamber'>
  <result xmlns='urn:xmpp:mam:2' queryid='g28' id='28482-20987-73623'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <delay xmlns='urn:xmpp:delay' stamp='2010-07-10T23:08:25Z'/>
      <message xmlns="jabber:client">
                <event xmlns='http://jabber.org/protocol/pubsub#
                    event'>
                  <items node='princely_musings'>
                    <item id='ae890ac52d0df67ed7cfdf51b644e901'>
                      <entry xmlns='http://www.w3.org/2005/Atom'>
                        <title>Soliloquy</title>
                        <summary>
                                      To be, or not to be: that is
                                          the question:
                                      Whether 'tis␣nobler␣in␣the␣
                                          mind␣to␣suffer
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣The␣slings␣and␣arrows␣of␣
    outrageous␣fortune,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Or␣to␣take␣arms␣against␣a␣sea␣
    of␣troubles,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/
    atom03'/>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</items>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</event>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</message>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</forwarded>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</result>
```

```
</message>
```

5.2 IDs

The IDs used within an archive MUST be unique per item stored and MUST NOT be reused, even if the original item with a given ID has since been removed from the archive. If a server provides multiple archives (e.g. many user archives, or many MUC archives), the IDs do not need to be unique across all of these archives unless the server also allows a single query to be run across multiple archives (e.g. searching of all MUC rooms), discussion of which is beyond the scope of this document. These IDs are strings that servers may construct in any manner, and clients must treat as opaque strings (e.g. there is no requirement for them to be numeric, sequenced or GUIDs).

5.3 Client synchronization

In addition to one-off queries, clients may use this protocol to synchronize a local archive with the server's archive. However because this protocol is detached from normal routing of messages, it is possible that a client will receive messages while trying to synchronize, which has the potential to cause duplicated messages. Resolving this is beyond the scope of this specification, but may instead be solved during the initial connection phase by using an alternative connection protocol such as Bind 2.0 (XEP-0386) [12].

6 Archiving Preferences

Depending on implementation and deployment policies, a server MAY allow the user to have control over the server's archiving behaviour. This specification defines a basic protocol for this, and also allows a server to offer more advanced configuration to a user.

6.1 Simple configuration

If the server supports and allows configuration of the preferences described below then it SHOULD implement the protocol defined in this section. This allows the user to retrieve and configure the following preferences:

- A list of JIDs that should always have messages to/from archived in the user's store.
- A list of JIDs that should never have messages to/from archived in the user's store.
- The default archiving behaviour (for JIDs in neither of the above lists).

[12] XEP-0386: Bind 2.0 <https://xmpp.org/extensions/xep-0386.html>.

Listing 19: Retrieving archiving preferences

```
<iq type='get' id='juliet2'>
  <prefs xmlns='urn:xmpp:mam:2'/>
</iq>
```

The server replies with the user's current archiving preferences. The <prefs> element MUST be present and contain the current default archiving policy. The <always> and <never> MUST also be present (even if empty), and contain a list of JIDs enclosed in <jid> elements.

Listing 20: Server responds with current preferences

```
<iq type='result' id='juliet2'>
  <prefs xmlns='urn:xmpp:mam:2' default='roster'>
    <always/>
    <never/>
  </prefs>
</iq>
```

It is also possible that the server may respond with a stanza error, for example the standard 'feature-not-implemented' (server does not support MAM configuration) defined in RFC 6120 [13].

Listing 21: Server does not support archive configuration

```
<iq type='error' id='juliet2'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
  </error>
</iq>
```

To update the preferences, the client can simply send an iq stanza with a type of 'set':

Listing 22: Updating archiving preferences

```
<iq type='set' id='juliet3'>
  <prefs xmlns='urn:xmpp:mam:2' default='roster'>
    <always>
      <jid>romeo@montague.lit</jid>
    </always>
    <never>
      <jid>montague@montague.lit</jid>
    </never>
  </prefs>
</iq>
```

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

The server then replies with the applied preferences (note that due to server policies these MAY be different to the preferences sent by the client):

Listing 23: Server responds with updated preferences

```
<iq type='result' id='juliet3'>
  <prefs xmlns='urn:xmpp:mam:2' default='roster'>
    <always>
      <jid>romeo@montague.lit</jid>
    </always>
    <never>
      <jid>montague@montague.lit</jid>
    </never>
  </prefs>
</iq>
```

It is also possible for the server to respond with an error, for example (but not limited to) the standard 'feature-not-implemented' (the server does not support configuration of preferences), 'forbidden' (the user is not authorized to change their preferences) or 'not-allowed' (the server generally does not allow changing of configuration preferences).

6.1.1 Default behaviour

If a JID is in neither the 'always archive' nor the 'never archive' list then whether it is archived depends on this setting, the default.
The 'default' attribute of the 'prefs' element MUST be one of the following values:

- 'always' - all messages are archived by default.
- 'never' - messages are never archived by default.
- 'roster' - messages are archived only if the contact's bare JID is in the user's roster.

6.1.2 Always archive

The <prefs/> element MAY contain an <always/> child element. If present, it contains a list of <jid/> elements, each containing a single JID. The server SHOULD archive any messages to/from this JID (see 'JID matching').
If missing from the preferences, <always/> SHOULD be assumed by the server to be an empty list.

6.1.3 Never archive

The <prefs/> element MAY contain an <never/> child element. If present, it contains a list of <jid/> elements, each containing a single JID. The server SHOULD NOT archive any messages

to/from this JID (see 'JID matching').
If missing from the preferences, <never/> SHOULD be assumed by the server to be an empty list.

6.2 Advanced configuration

In addition to this protocol, a server MAY offer more advanced configuration to the user through Ad-Hoc Commands (XEP-0050) [14]. Such an interface might, for example, allow the user to configure what types of messages to store, or set a limit on how long messages should remain in the archive.
If supported, such a configuration command SHOULD be presented on the well-defined command node of "urn:xmpp:mam#configure".

6.3 JID matching

6.3.1 General rules

When comparing the message target JID against the user's roster (ie. when the user has set default='roster') the comparison MUST use the bare target JID (that is, stripped of any resource).
For matching against entries in either the 'allow' or 'never' lists, for each listed JID:

- If the listed JID contains a resource, compare against the target JID as-is.
- If the listed JID has no resource (it is a bare JID) then first strip any resource from the target JID prior to comparison.

6.3.2 Outgoing messages

For outgoing messages, the server MUST use the value of the 'to' attribute as the target JID.

6.3.3 Incoming messages

For incoming messages, the server MUST use the value of the 'from' attribute as the target JID.

[14] XEP-0050: Ad-Hoc Commands <https://xmpp.org/extensions/xep-0050.html>.

6.4 Processing Hints

Clients can use Message Processing Hints (XEP-0334) [15] for signaling that they do not wish some messages to be stored in the archive.

```
<message from='romeo@montague.lit/laptop' to='juliet@capulet.lit/
    laptop'>
  <body>V unir avtug'f␣pybnx␣gb␣uvqr␣zr␣sebz␣gurve␣fvtug</body>
␣␣<no-store␣xmlns='urn:xmpp:hints'/>
</message>
```

7 Determining support

If a server or other entity hosts archives and supports MAM queries, it MUST advertise the 'urn:xmpp:mam:2' feature in response to Service Discovery (XEP-0030) [16] requests made to archiving JIDs (i.e. JIDs hosting an archive, such as users' bare JIDs):

Listing 24: Client queries for server features

```
<iq type='get' id='disco1' to='juliet@capulet.lit' from='
    juliet@capulet.lit/balcony'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 25: Server responds with features

```
<iq type='result' id='disco1' from='juliet@capulet.lit' to='
    juliet@capulet.lit/balcony'>
        <query xmlns='http://jabber.org/protocol/disco#info'>
                ...
                <feature var='urn:xmpp:mam:2'/>
                ...
        </query>
</iq>
```

8 Security Considerations

8.1 Data privacy

An archive generally consists of private conversations, and so a server MUST adequately protect an archive from unauthorized third-party access. For example authorized parties for a user's archive would likely include just the user, and a MUC archive for a private room

[15] XEP-0334: Message Processing Hints <https://xmpp.org/extensions/xep-0334.html>.
[16] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

might be restricted to room members. An implementation MAY choose to allow access to any archive by server administrators. If a client requests access to an archive it does not have permissions for the server MUST return an iq with type error, and the error condition SHOULD be 'forbidden'.
A server SHOULD provide a mechanism for a user to disable archiving of messages with all or specific contacts, such as via the configuration protocol described in this document. This allows the user to prevent the archiving of potentially sensitive messages in the first place.
A server MAY automatically prevent certain sensitive messages from being archived. How such messages are identified is beyond the scope of this specification, but technologies such as Security Labels in XMPP (XEP-0258) [17] may be used, for example.

8.2 Stanza IDs

Entities that implement this specification must also adhere to the security requirements of XEP-0359.

8.3 MUC message spoofing

This specification re-uses the <x> element from the 'http://jabber.org/protocol/muc#user' namespace to convey information about the sender of a message in a MUC room. However this element is not sanitized by MUC services, so the archiving entity MUST strip any existing <x> element in the 'http://jabber.org/protocol/muc#user' namespace from messages before archiving them (regardless of whether it adds in its own <x> element).

9 XMPP Registrar Considerations

10 Acknowledgements

Many thanks to Dave Cridland, Kim Alvefur, Yann Leboulanger, Evgeny Khramtsov, Florian Schmaus, Lance Stout, Waqas Hussain and Daniel Gultsch for their input and feedback on this specification.

[17] XEP-0258: Security Labels in XMPP <https://xmpp.org/extensions/xep-0258.html>.