# XEP-0136: Message Archiving

**Abstract:** This document defines mechanisms and preferences for the server-side archiving and retrieval of XMPP messages.

**Authors:** Ian Paterson, Jon Perlow, Peter Saint-Andre, Justin Karneges, Alexander Tsvyashchenko, Yann Leboulanger



**Status:** Draft
**Type:** Standards Track
**Version:** 1.2
**Last Updated:** 2010-06-21

NOTICE: The protocol defined herein is a **Draft Standard** of the XMPP Standards Foundation. Implementations are encouraged and the protocol is appropriate for deployment in production systems, but some changes to the protocol are possible before it becomes a Final Standard.

## Table of Contents

---

# 1. Introduction

> *Notice (2010-06-21): The **XMPP Council** [1] believes that the Message Archiving protocol is unduly complex and will likely request a thorough review and simplification of this specification before advancing it to Final.*

Many XMPP clients implement some form of client-side message archiving. However, it is not always convenient or even possible to archive messages locally, e.g., because it is easier to keep all archives in one universally accessible place (not scattered around on multiple computers or devices) or because the client operates in a web browser or resides on a mobile device that does not have sufficient local storage for message archiving. In addition, server-side archiving makes it possible to offer new services such as integration of instant messaging and email. Therefore it is beneficial to define methods for server-side archiving of XMPP messages.

There are two main approaches to this problem:

1. Enable the client to send individual messages or entire conversations to the server for archiving; we call this MANUAL ARCHIVING.
2. Enable the server (at the user's request) to archive messages as they pass through the server; we call this AUTOMATED ARCHIVING.

So that client and server developers can refer to one specification, both approaches are defined in this document. In addition, this document defines common methods for retrieving and managing archived messages.

Note: Complying with **XMPP Core**, the server MUST respond to all <iq/> element of type 'get' or 'set'. However, most successful responses have been omitted from this document in the interest of conciseness.

# 2. Archiving Preferences

## 2.1 Introduction

Not all users want to archive messages. A client SHOULD save its user's default archiving preference (or "Save Mode") to its own server (i.e., specify whether by default all conversations should be archived or not). In addition, a client MAY save different preferences for particular contacts.

Some users may also prefer that the messages they exchange with contacts are "Off The Record" (OTR). [2] A client SHOULD save its user's default and contact-specific OTR preferences (or "OTR Modes") to its own server.

Whichever archiving method a client uses (e.g., local archiving to files or a database, or server-side archiving that happens either automatically or manually), it SHOULD adhere to its user's archiving preferences as stored on the server. However, a client MAY maintain a set of preferences in a local file which takes precedence over the preferences stored on the server for both local archiving and server-side archiving.

This section addresses the following use cases:

1. A client determines its user's current default Save Mode and OTR Mode, and the Modes for particular contacts.
2. A client sets the default Save Mode and OTR Mode.
3. A client sets the Save Mode and OTR Mode for a particular contact.
4. A client sets the Save Mode for a particular chat session.

## 2.2 Preference Syntax

Archiving preferences are encapsulated in four children of the <pref/> element: <auto/>, <default/>, <item/>, and <method/>. These are defined in the following sections.

In order to capture a complete set of preferences, when the server returns the user's preferences to the client the <pref/> element:

- MUST include an <auto/> element that specifies whether automatic archiving is on or off.
- MUST include a <default/> element that specifies the user's default settings for OTR Mode and Save Mode.
- MAY include one or more <item/> elements that specify preferences related to particular contacts.
- MAY include one or more <session/> elements that specifies whether automatic archiving is on or off for a given chat session.
- MUST include at least three <method/> elements, differentiated by the value of the 'type' attribute (i.e., at least one <method/> element each for "auto", "local", and "manual").

An example follows.

**Example 1. Complete Preferences**

```
<iq type='result' id='pref1' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <auto save='false'/>
    <default expire='31536000' otr='concede' save='body'/>
    <item jid='romeo@montague.net' otr='require' save='false'/>
    <item expire='630720000' jid='benvolio@montague.net' otr='forbid' save='message'/>
    <session thread='ffd7076498744578d10edabfe7f4a866' save='body'/>
    <method type='auto' use='forbid'/>
    <method type='local' use='concede'/>
    <method type='manual' use='prefer'/>
  </pref>
</iq>
```

### 2.2.1 Auto Element

The <auto/> element specifies the current Automatic Archiving settings for *this stream*.

This element MUST be empty and MUST include a boolean 'save' attribute [3] that specifies whether automatic archiving is enabled or disabled for this stream. The element MAY include a 'scope' attribute that specifies how long this setting is true. The allowable values are:

- global -- the setting will remain for next streams.
- stream -- the setting is true only until the end of the stream. For next stream, server default value will be used.

Note: If 'scope' attribute is not set, it SHOULD be considered as 'stream'.

### 2.2.2 Default Element

The <default/> element specifies the default settings for both OTR Mode and Save Mode. The element MUST be empty and MUST include an 'otr' attribute and a 'save' attribute. The element MAY include an 'expire' attribute.

#### 2.2.2.1 expire Attribute

If the 'save' attribute is *not* set to 'false' then is RECOMMENDED to also include an 'expire' attribute, which indicates how many seconds after messages are archived that the server SHOULD delete them.

#### 2.2.2.2 otr Attribute

The 'otr' attribute specifies the user's default setting for OTR Mode. The allowable values are:

- approve -- the user MUST explicitly approve off-the-record communication.
- concede -- communications MAY be off the record if requested by another user.
- forbid -- communications MUST NOT be off the record.
- oppose -- communications SHOULD NOT be off the record even if requested.
- prefer -- communications SHOULD be off the record if possible.
- require -- communications MUST be off the record. *

* Note: If the OTR Mode is 'require' then the Save Mode MUST be 'false'. An 'otr' attribute value of "require" in Message Archiving is equivalent to a 'logging' attribute value of "mustnot" in Stanza Session Negotiation; for details, see the OTR Negotiation section of this document.

#### 2.2.2.3 save Attribute

The 'save' attribute specifies the user's default setting for Save Mode. The allowable values are:

- body -- the saving entity SHOULD save only <body/> elements. *
- false -- the saving entity MUST save nothing.
- message -- the saving entity SHOULD save the full XML content of each <message/> element. **
- stream -- the saving entity SHOULD save every byte that passes over the stream in either direction. ***

* Note: When archiving *locally* a client MAY save the full XML content of each <message/> element even if the Save Mode is 'body'.

** Note: Support for the 'message' value is optional and, to conserve bandwidth and storage space, it is RECOMMENDED that client implementations do not specify the 'message' value. [4]

*** Note: The upload, retrieval and management of 'stream' archives is currently beyond the scope of this document.

### 2.2.3 Item Element

The <item/> element specifies the settings for both the OTR Mode and Save Mode with regard to a particular entity. The element MUST be empty and MUST include a 'jid' attribute, an 'otr' attribute, and a 'save' attribute. The element MAY include an 'expire' attribute.

#### 2.2.3.1 expire Attribute

If the 'save' attribute is *not* set to 'false' then is RECOMMENDED to also include an 'expire' attribute, which indicates how many seconds after messages are archived that the server SHOULD delete them.

#### 2.2.3.2 jid Attribute

The 'jid' attribute specifies the JabberID of the XMPP entities to which the preferences specified in this <item/> element apply, see the JID Matching section of this document.

#### 2.2.3.3 otr Attribute

The 'otr' attribute specifies the user's setting for OTR Mode with regard to the specified JID. The allowable values are:

- approve -- the user MUST explicitly approve off-the-record communication.
- concede -- communications MAY be off the record if requested by another user.
- forbid -- communications MUST NOT be off the record.
- oppose -- communications SHOULD NOT be off the record even if requested.
- prefer -- communications SHOULD be off the record if possible.
- require -- communications MUST be off the record. *

* Note: If the OTR Mode is 'require' then the Save Mode MUST be 'false'. An 'otr' attribute value of "require" in Message Archiving is equivalent to a 'logging' attribute value of "mustnot" in Stanza Session Negotiation; for details, see the OTR Negotiation section of this document.

#### 2.2.3.4 save Attribute

The 'save' attribute specifies the user's setting for Save Mode with regard to the specified JID. The allowable values are:

- body -- the saving entity SHOULD save only <body/> elements.
- false -- the saving entity MUST save nothing.
- message -- the saving entity SHOULD save the full XML content of each <message/> element.
- stream -- the saving entity SHOULD save every byte that passes over the stream in either direction. *

* Note: The upload, retrieval and management of 'stream' archives is *currently* beyond the scope of this document.

### 2.2.4 Session Element

The <session/> element specifies the settings for Save Mode with regard to a particular chat session. The element MUST be empty and MUST include a 'thread' attribute, and a 'save' attribute. The element MAY include a 'timeout' attribute.

Server implementations SHOULD remove all <session/> elements when stream is closed.

#### 2.2.4.1 timeout Attribute

The 'timeout' attribute indicates how long this rule will stay in server after the latest message in this thread is exchanged. Server MUST NOT forget this rule before 'timeout' seconds after latest message in this thread is exchanged but MAY keep this rule longer than 'timeout' value specifies.

Client MUST NOT set this attribute, but wait for server's answer to know this value.

If the client wants to keep this rule longer, it must send a new <session/> element to the server before this timeout expires.

#### 2.2.4.2 thread Attribute

The 'thread' attribute specifies the ThreadID of the chat session (in the sense of **Best Practices for Message Threads** [5]) to which the preferences specified in this <session/> element apply.

#### 2.2.4.3 save Attribute

The 'save' attribute specifies the user's setting for Save Mode with regard to the specified chat session. The allowable values are:

- body -- the saving entity SHOULD save only <body/> elements.
- false -- the saving entity MUST save nothing.
- message -- the saving entity SHOULD save the full XML content of each <message/> element.
- stream -- the saving entity SHOULD save every byte that passes over the stream in either direction. *

* Note: The upload, retrieval and management of 'stream' archives is *currently* beyond the scope of this document.

### 2.2.5 Method Element

Each <method/> element specifies the the user's preferences for one available archiving method. The <method/> element MUST be empty and MUST include both the 'type' and 'use' attributes.

#### 2.2.5.1 type Attribute

The allowable values of the 'type' attribute are:

- auto -- preferences for use of automatic archiving on the user's server.
- local -- preferences for use of local archiving to a file or database on the user's machine or device.
- manual -- preferences for use of manual archiving by the user's client to the user's server.

#### 2.2.5.2 use Attribute

The allowable values of the 'use' attribute are:

- concede -- this method MAY be used if no other methods are available.
- forbid -- this method MUST NOT be used.
- prefer -- this method SHOULD be used if available.

## 2.3 Determining Preferences

In order to determine its user's current Save Mode(s) and OTR Mode(s), a client sends to its server an IQ-get containing an empty <pref/> element qualified by the 'urn:xmpp:archive' namespace.

**Example 2. Client Requests Archiving Preferences**

```
<iq type='get' id='pref1'>
  <pref xmlns='urn:xmpp:archive'/>
</iq>
```

The server responds with the default Save Mode and OTR Mode (a single <default/> element) and any specific Save Modes and OTR Modes for individual contacts (zero or more <item/> elements).

**Example 3. Server Returns Preferences**

```
<iq type='result' id='pref1' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <auto save='false'/>
    <default expire='31536000' otr='concede' save='body'/>
    <item jid='romeo@montague.net' otr='require' save='false'/>
    <item expire='630720000' jid='benvolio@montague.net' otr='forbid' save='message'/>
    <session thread='ffd7076498744578d10edabfe7f4a866' save='body'/>
    <method type='auto' use='forbid'/>
    <method type='local' use='concede'/>
    <method type='manual' use='prefer'/>
  </pref>
</iq>
```

The foregoing preferences can be explained as follows:

1. By default, message bodies should be saved (according the preferred method specified later), communications may be off the record if requested, and any saved messages should be expired after 1 year.
2. When communicating with romeo@montague.net, both entities must not save messages and all communications must be off the record.
3. When communicating with benvolio@montague.net, both entities should save full messages, communications must not be off the record, and any saved messages should be expired after 20 years.
4. Message bodies in thread ffd7076498744578d10edabfe7f4a866 should be saved.
5. Server-side archiving must not occur automatically.
6. Local archiving may be used if requested.
7. Manual server-side archiving is preferred.

If the user has never set the default Modes, the 'save' and 'otr' attributes SHOULD specify the server's default settings, and the 'unset' attribute SHOULD be set to 'true'. Note: The 'unset' attribute defaults to 'false'.

**Example 4. Server Returns Service Default Preferences**

```
<iq type='result' id='pref1' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <default otr='concede' save='false' unset='true'/>
    <method type='auto' use='concede'/>
    <method type='local' use='concede'/>
    <method type='manual' use='concede'/>
    <auto save='false'/>
  </pref>
</iq>
```

Once it has received a request for archiving preferences from the client, the server MUST send any subsequent changes to any of the user's archiving preferences to the client until the stream is closed (see below). Note: changes to the <auto/> element MUST NOT be replicated in this way.

## 2.4 Setting Default Modes

A client may set the default Modes:

**Example 5. Client Sets Default Modes**

```
<iq type='set' id='pref2'>
  <pref xmlns='urn:xmpp:archive'>
    <default otr='prefer' save='false'/>
  </pref>
</iq>
```

If the server can process the request, it acknowledges the change:

**Example 6. Server Acknowledges Change**

```
<iq type='result' id='pref2' to='juliet@capulet.com/chamber'/>
```

The server then MUST inform all of the user's connected resources that have previously requested the user's archiving preferences:

**Example 7. Server Pushes New Modes**

```
<iq type='set' id='push1' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <default otr='prefer' save='false'/>
  </pref>
</iq>

<iq type='set' id='push2' to='juliet@capulet.com/pda'>
  <pref xmlns='urn:xmpp:archive'>
    <default otr='prefer' save='false'/>
  </pref>
</iq>
```

If the server does not allow the saving of full message stanza content, the client set the value of the 'save' attribute to 'message' or 'stream', and any of the user's connected resources have Automatic Archiving enabled, then the server SHOULD return a <feature-not-implemented/> error.

If administrator policies require that at least the <body/> elements (or the full content) of every message must be logged automatically and the client attempts to set the value of the 'save' attribute to 'false' or 'body', then the server SHOULD return a <not-acceptable/> error.

## 2.5 Setting Modes for a Contact

A client may use a similar protocol to set the Modes for a particular contact or domain of contacts (bare JID, full JID or domain). Note: It is STRONGLY RECOMMENDED for the value of the 'jid' attribute to be a bare JID <localpart@domain.tld> rather than a full JID <localpart@domain.tld/resource>.

**Example 8. Client Sets Modes for a Contact**

```
<iq type='set' id='pref3'>
  <pref xmlns='urn:xmpp:archive'>
    <item jid='romeo@montague.net' save='body' expire='604800' otr='concede'/>
  </pref>
</iq>
```

**Example 9. Server Acknowleges Change**

```
<iq type='result' id='pref3' to='juliet@capulet.com/chamber'/>
```

**Example 10. Server Pushes New Modes**

```
<iq type='set' id='push3' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <item jid='romeo@montague.net' save='body' expire='604800' otr='concede'/>
  </pref>
</iq>

<iq type='set' id='push4' to='juliet@capulet.com/pda'>
```

```
  <pref xmlns='urn:xmpp:archive'>
    <item jid='romeo@montague.net' save='body' expire='604800' otr='concede'/>
  </pref>
</iq>
```

The same error cases apply as when Setting Default Modes.

In order to remove all preferences for a contact, the client shall send an <itemremove/> element to the server.

**Example 11. Client Removes Preferences for a Contact**

```
<iq type='set' id='remove1'>
  <itemremove xmlns='urn:xmpp:archive'>
    <item jid='benvolio@montague.net'/>
  </itemremove>
</iq>
```

**Example 12. Server Acknowleges Change**

```
<iq type='result' id='remove1' to='juliet@capulet.com/chamber'/>
```

## 2.6 Setting Modes for a Chat Session

A client may use a similar protocol to set the Modes for a particular chat session. A chat session is identified by its unique 'thread' attributes in <message> stanza (see **Stanza Session Negotiation** [6]).

**Example 13. Client Sets Modes for a Chat Session**

```
<iq type='set' id='pref4'>
  <pref xmlns='urn:xmpp:archive'>
    <session thread='ffd7076498744578d10edabfe7f4a866' save='body' otr='concede'/>
  </pref>
</iq>
```

**Example 14. Server Acknowleges Change**

```
<iq type='result' id='pref4' to='juliet@capulet.com/chamber'/>
```

**Example 15. Server Pushes New Modes**

```
<iq type='set' id='push5' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <session thread='ffd7076498744578d10edabfe7f4a866' save='body' timeout='3600' otr='concede'/
  </pref>
</iq>

<iq type='set' id='push6' to='juliet@capulet.com/pda'>
  <pref xmlns='urn:xmpp:archive'>
    <session thread='ffd7076498744578d10edabfe7f4a866' save='body' timeout='3600' otr='concede'/
  </pref>
</iq>
```

The same error cases apply as when Setting Default Modes.

In order to remove a preference for a chat session, the client shall send an <sessionremove/> element to the server.

**Example 16. Client Removes Preferences for a Contact**

```
<iq type='set' id='remove2'>
  <sessionremove xmlns='urn:xmpp:archive'>
    <session thread='ffd7076498744578d10edabfe7f4a866'/>
  </sessionremove>
</iq>
```

**Example 17. Server Acknowleges Change**

```
<iq type='result' id='remove2' to='juliet@capulet.com/chamber'/>
```

## 2.7 Setting Archiving Method Preferences

The client can set one or more method preferences by sending an IQ-set containing a <pref/> element that in turn contains one or more <method/> elements.

**Example 18. Client Sets Method Preferences**

```
<iq type='set' id='pref5'>
  <pref xmlns='urn:xmpp:archive'>
    <method type='auto' use='concede'/>
    <method type='local' use='forbid'/>
    <method type='manual' use='prefer'/>
  </pref>
</iq>
```

**Example 19. Server Acknowleges Change**

```
<iq type='result' id='pref5' to='juliet@capulet.com/chamber'/>
```

If the client includes less than three <method/> elements, the server MUST NOT modify the unspecified methods and MUST leave them as currently stored on the server. However, when the server pushes the method preferences it MUST include all of the preferences, not only those that were set by the client.

**Example 20. Server Pushes New Method Preferences**

```
<iq type='set' id='push7' to='juliet@capulet.com/chamber'>
  <pref xmlns='urn:xmpp:archive'>
    <method type='auto' use='concede'/>
    <method type='local' use='forbid'/>
    <method type='manual' use='prefer'/>
  </pref>
</iq>

<iq type='set' id='push8' to='juliet@capulet.com/pda'>
  <pref xmlns='urn:xmpp:archive'>
    <method type='auto' use='concede'/>
    <method type='local' use='forbid'/>
    <method type='manual' use='prefer'/>
  </pref>
</iq>
```

## 2.8 Server Archiving Preferences Interpretation

Most archiving preferences are designed for interpretation only by the client. The server MUST NOT take into account any of the archiving preferences when server administration policies *require* that every message is to be logged automatically. Otherwise, the server MUST interpret the following archiving preferences (and SHOULD NOT interpret any other ones):

1. The <auto/> element.
2. When performing automatic archiving: the 'save' attribute of the <default> element, the 'jid' and 'save' attributes of the <item> element, and the 'thread' and 'save' attributes of the <session> element. See Preferences precedence rules for details.
3. When performing expiration of old messages: the 'jid' and 'expire' attributes of the <item> element.
4. When performing expiration of old rules: the 'thread' and 'timeout' attributes of the <session> element.

## 2.9 Preferences precedence rules

When determining archiving preferences for a given message, the following rules shall apply:

1. 'save' value is taken from the <session> element that matches the conversation, if present, else from the <item> element that matches the contact (see JID Matching), if present, else from default element.
2. 'otr' and 'expire' value are taken from the <item> element that matches the contact (see JID

Matching), if present, else from default element.

# 3. Off The Record

A user will sometimes exchange messages with contacts who prefer that their conversations are not archived by either party.

## 3.1 OTR Negotiation

Any client that archives messages SHOULD support **Stanza Session Negotiation** and its 'logging' field both to give other contacts the opportunity to indicate this preference, and to negotiate an "Off The Record" (OTR) policy that complies with its user's own Archiving Preferences.

Note: A client MUST NOT propose or agree to enable OTR (i.e., disallow message logging) unless it has confirmed that its server will allow it to switch off Automatic Archiving. The client can do so based on the rules in the Automatic Archiving section of this document. If the client logs in and does not receive a warning message, it can assume that automatic archiving is not on by default. If the client does receive a warning message because automatic archiving is on by default, the client can determine if auto-archiving can be turned off by trying to do so; if the client receives an error, it knows that automatic archiving cannot be turned off.

Both parties to a stanza session negotiation may have OTR preferences (i.e, the initiating party or "user" and the responding party or "contact"). These preferences will interact in the ways specified below, resulting either in a successful negotiation or an unsuccessful negotiation (naturally, an unsuccessful negotiation can lead to a subsequent negotiation attempt by the user or the contact).

The following table shows how to instantiate the user's OTR preference in a stanza session negotiation (SSN) offer. The various OTR preferences map to particular values of the SSN 'logging' field, including the order of values for that field. In particular, an SSN logging value of 'may' means that the receiving party MAY enable message logging and an SSN logging value of 'mustnot' means that the receiving party MUST NOT enable message logging.

**Table 1: Stanza Session Negotiation logging options offered by initiating party (user)**

| User's OTR Preference | Offering 'logging' Negotiation Option(s)* |
|---|---|
| require | mustnot** |
| prefer | mustnot,may |
| approve | mustnot,may |
| concede | may,mustnot*** |
| oppose | may,mustnot*** |
| forbid | may*** |

* In order of preference, the first value is the default.

** If the user receives no response it MUST NOT send any messages to the contact.

*** Alternatively, the user MAY decide not to *initiate* an OTR negotiation and to save messages (until the contact initiates a negotiation).

Note: When negotiating a stanza session, the user MUST include the <required/> element inside the 'logging' <field/> element. If the user does not receive a successful response to its chat negotiation request (and if the OTR Mode is not 'require'), then it SHOULD proceed as if the contact had responded with the value of the 'logging' <field/> element set to 'may'.

The following table shows what stanza session negotiation values the responding party (i.e., "contact") should send for the 'logging' field in its response to a stanza session negotiation request from the user.

**Table 2: Stanza Session Negotiation logging value selected by responding party (contact)**

| Contact's OTR Preference | Responding 'logging' Negotiation Values* | | | |
|---|---|---|---|---|
| **Initiator Options -->** | **mustnot** | **mustnot,may*** | **may,mustnot*** | **may** |
| require OTR mode | mustnot | mustnot | mustnot | fail** |
| prefer OTR mode | mustnot | mustnot | mustnot | may |
| approve OTR mode | mustnot | mustnot | may | may |
| concede OTR mode | mustnot | mustnot | may | may |
| oppose OTR mode | mustnot | may | may | may |

| | | | | |
|---|---|---|---|---|
| forbid OTR mode | fail** | may | may | may |

* The first value is the default.

** The negotiation fails and the parties MUST NOT exchange any messages; however, the recipient MAY attempt to initiate a stanza session negotiation with the other party.

Note: If a contact does not include a 'logging' field in its initial Stanza Session Negotiation request, and a user's Archiving Preferences indicate that OTR is *required*, then the client MUST refuse the request. It MAY then send its own Stanza Session Negotiation request with a 'logging' field.

If a user's OTR preference for a contact changes during a Chat Session that has been negotiated with the contact, and if the new preference would affect the value of the 'logging' field that was previously negotiated, then the client MUST immediately renegotiate the 'logging' field according to the user's new OTR preference (or terminate the Chat Session).

### 3.2 Notes

If a Stanza Session Negotiation result differ from current preferences of archiving for the contact (negotiation agreed to enable OTR and current 'save' value for this contact is *not* 'false', or negotiation agreed to disable OTR and current 'save' value for this contact is 'false'), the client MUST send a new <session/> element with the corresponding 'thread' attribute to the server to inform it to save or not the session.

If a Stanza Session Negotiation agreed to enable OTR then the clients MUST NOT allow messages sent in *either* direction to be archived in any way (including Manual Archiving and Automatic Archiving). [7]

If a Stanza Session Negotiation agreed to enable OTR then both clients MUST ensure that the Stanza Session Negotiation messages themselves are not archived. For example, if Automatic Archiving was enabled when the client received the initial Stanza Session Negotiation request, then the client MUST immediately ask its server to delete its copy of the request (see Removing a Collection for a description of how to remove the messages currently being recorded by the server).

## 4. Collections: The Chat Element

Whether manual archiving or automatic archiving is used, messages are archived in the form of "collections". A collection is a set of messages to/from the same user that are received near each other in time or as part of the same conversation thread. A collection is intended to mimic the natural flow of human conversations, which in instant messaging (IM) systems tend to occur in bursts (e.g., a five-minute conversation one day, followed by a ten-minute conversation the next).

Each collection of messages and notes is encapsulated in a <chat/> element and is uniquely identified by the combination of the 'start' and 'with' attributes as defined below. The syntax of the <chat/> element is specified in this section.

### 4.1 start Attribute

The 'start' attribute specifies the start time of the conversation thread, which MUST be UTC and adhere to the DateTime format specified in **XMPP Date and Time Profiles** [8].

### 4.2 subject Attribute

The 'subject' attribute specifies a friendly name for the collection (note the Security Considerations regarding the subject attribute). Inclusion of the 'subject' attribute is OPTIONAL.

### 4.3 thread Attribute

A client SHOULD include a thread ID in each <message/> element it sends that is part of a conversation it expects will be archived (as explained in **Best Practices for Message Threads** [9], a thread ID is captured in the <thread/> child of the <message/> element).

If the messages contained in a conversation contain a thread ID, then the server MUST map that thread ID to the 'thread' attribute of the <chat/> element. There MUST be a one-to-one mapping between the <thread/> element and the 'thread' attribute.

If a thread ID is not included, the server may use its own implementation-specific methods for mapping messages and conversations to collections.

The content of <message/> elements that have different thread IDs SHOULD be archived in separate collections and the content of <message/> elements that have the same thread IDs SHOULD be archived in the same collection; this is the responsibility of the client if manual archiving is used and the responsibility of the server if automatic archiving is used. The thread attribute SHOULD NOT be set to any value other than the exact content of the <thread/> elements. If no <thread/> elements appeared in the conversation then the <chat/> element SHOULD have no thread attribute. Implementations SHOULD use the thread attribute for cross-referencing purposes only, within the

archive each collection MUST be uniquely identified by the combination of its 'with' and 'start' attributes.

Inclusion of the 'thread' attribute is RECOMMENDED.

### 4.4 version Attribute

Upon receiving a manually-uploaded collection or creating a collection as a result of auto-archiving, the server MUST assign a version number to the collection, which MUST start at zero (0). Whenever the collection is modified, the server MUST increment the version number by one (1). The server MUST include the version number attribute whenever it sends the collection or information about the collection to the client, by including a 'version' attribute in the <chat/>, <changed/>, or <removed/> element. If the client includes a 'version' attribute in an IQ-set, the server MUST ignore it.

Inclusion of the 'version' attribute is REQUIRED of servers.

### 4.5 with Attribute

The 'with' attribute specifies the JID with which the messages were exchanged.

Inclusion of the 'with' attribute is REQUIRED.

### 4.6 from and to Elements

The content of each individual message MUST be encapsulated in a <to/> or <from/> element. The time in whole seconds of the message relative to the previous message in the collection (or, for the first message, relative to the start of the collection) SHOULD be specified with a 'secs' attribute. Note: When deciding whether to round up or down to a number of whole seconds, entities MUST ensure that the sum of the 'secs' attribute and the 'secs' attributes of the preceeding messages will accurately reflect the absolute time of the message. (e.g., if a sequence of messages occur at exactly 0.51-second intervals then the 'secs' attributes should generally alternate between '0' or '1'.)

The content of each <to/> or <from/> element SHOULD depend on the user's Archiving Preferences. <to/> or <from/> elements MUST NOT be empty. Note: A server MAY be configured to return a <feature-not-implemented/> error if any <to/> or <from/> element contains anything other than <body/> elements.

### 4.7 note Element

The <note/> element specifies a private note about the conversation. The absolute time the note was created SHOULD be specified with a 'utc' attribute (which MUST be UTC and adhere to the DateTime format specified in **XEP-0082**). Inclusion of the <note/> element is OPTIONAL.

## 5. Manual Archiving

### 5.1 Introduction

While automatic archiving is easy for the client and server to implement, there are many contexts in which manual archiving is required. For examples, when:

- Messages are encrypted using evanescent keys, as in **Encrypted Session Negotiation** [10]
- A client's own server does not support automatic archiving but it (or another server) does support manual archiving
- A server does not support encryption of auto-archived collections (see **Encryption of Archived Messages** [11])
- A client wants to maintain a unified archive for messages that were transmitted both in and out-of-band (e.g. SMS or email)
- A client wants to append private notes to a conversation

Therefore, often a client will want to send or receive a sequence of messages, optionally add private notes to the sequence, optionally encrypt the sequence (see **XEP-0241**), and then ask the server to archive it. Such messages and notes SHOULD be stored on the server in the form of a "collection".

### 5.2 Uploading Messages to a Collection

A collection of messages and notes is uploaded to the server encapsulated in a <save/> element.

**Example 21. Storing messages in a collection**

```
<iq type='set' id='up1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'
          thread='damduoeg08'
          subject='She speaks!'>
      <from secs='0'><body>Art thou not Romeo, and a Montague?</body></from>
```

```
      <to secs='11'><body>Neither, fair saint, if either thee dislike.</body></to>
      <from secs='7'><body>How cam'st thou hither, tell me, and wherefore?</body></from>
      <note utc='1469-07-21T03:04:35Z'>I think she might fancy me.</note>
    </chat>
  </save>
</iq>
```

If the collection does not exist then the server MUST create a new collection and inform the client that the collection version number is zero.

**Example 22. Collection created**

```
<iq type='result' id='up1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'
          thread='damduoeg08'
          subject='She speaks!'
          version='0'/>
  </save>
</iq>
```

If the collection already exists then the server MUST append the messages to the existing collection (which MAY involve adding messages that appear to be duplicates, i.e., messages that have identical <from/> elements, <to/> elements, and dateTimes).

**Example 23. Messages appended to collection**

```
<iq type='result' id='up1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'
          thread='damduoeg08'
          subject='She speaks!'
          version='1'/>
  </save>
</iq>
```

Note: Clients MUST take care to append each sequence of messages to the collection before the sequence becomes so large that uploading it may violate common rate limiting restrictions (in Jabber systems, often called "karma").

If the server cannot service an upload request because the collection is too large then it MUST return a <not-acceptable/> error:

**Example 24. Unsuccessful reply**

```
<iq type='error' to='romeo@montague.net/orchard' id='up1'>
  <error code='406' type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

## 5.3 Changing the Subject of a Collection

If the client specifies a new value for the 'subject' attribute of any existing collection then the server MUST update the existing value and increment the collection version. Note: The client cannot specify new values for the 'with' or 'start' attributes. The only way to change these values is to delete the collection (see Removing a Collection) and then create a new one.

**Example 25. Changing the subject of a collection without appending messages**

```
<iq type='set' id='subject1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'
          subject='She speaks twice!'/>
  </save>
</iq>
```

**Example 26. Collection subject updated**

```
<iq type='result' id='subject1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'
          subject='She speaks twice!'
          version='1'/>
  </save>
</iq>
```

## 5.4 Offline Messages

The client MAY specify an absolute time for any message by providing a 'utc' attribute (which MUST be UTC and adhere to the DateTime format specified in **XEP-0082**) instead of a 'secs' attribute. The absolute time MAY be earlier than the start time of the collection:

**Example 27. Storing offline messages in a collection**

```
<iq type='set' id='up2'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'
          subject='She speaks!'>
      <from utc='1469-07-21T00:32:29Z'><body>Art thou not Romeo, and a Montague?</body></from>
      <to secs='11'><body>Neither, fair saint, if either thee dislike.</body></to>
      <from secs='7'><body>How cam'st thou hither, tell me, and wherefore?</body></from>
    </chat>
  </save>
</iq>
```

## 5.5 Groupchat Messages

A client MAY archive messages that it receives from **Multi-User Chat** [12] rooms. The 'with' attribute MUST be the bare JID of the room. The client MUST include a 'name' attribute for each <from/> element to specify the room nickname of the message sender and MAY include a 'jid' attribute to specify the full or bare JID of the sender (if available).

**Example 28. Storing groupchat messages in a collection**

```
<iq type='set' id='up3'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='balcony@house.capulet.com'
          start='1469-07-21T03:16:37Z'>
      <from secs='0' name='benvolio'><body>She will invite him to some supper.</body></from>
      <from secs='6' name='mercutio'><body>A bawd, a bawd, a bawd! So ho!</body></from>
      <from secs='3' name='romeo' jid='romeo@montague.net'><body>What hast thou found?</body></f
    </chat>
  </save>
</iq>
```

## 5.6 Linking Collections

Collections MAY be linked together by including a <previous/> and/or <next/> element. Each such element MUST include both a 'with' and a 'start' element to identify the other collection to which the collection is linked. For example, the <previous/> and <next/> elements in the two examples below are being used to link a groupchat between Romeo, Benvolio and Mercutio to a private chat that Romeo was having with Benvolio before they invited Mercutio to join them. Note: Collections MAY be linked in only one direction, they are not required to be double-linked in the way the examples below are.

**Example 29. Private chat linked to later groupchat**

```
<iq type='set' id='link1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='benvolio@montague.net'
          start='1469-07-21T03:01:54Z'>
      <next with='balcony@house.capulet.com' start='1469-07-21T03:16:37Z'/>
      <to secs='0'><body>O, I am fortune's fool!</body></to>
```

```
      <from secs='4'><body>Why dost thou stay?</body></from>
    </chat>
  </save>
</iq>
```

**Example 30. Groupchat linked to earlier private chat**

```
<iq type='set' id='link2'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='balcony@house.capulet.com'
          start='1469-07-21T03:16:37Z'>
      <previous with='benvolio@montague.net' start='1469-07-21T03:01:54Z'/>
      <from secs='0' name='benvolio'><body>She will invite him to some supper.</body></from>
      <from secs='6' name='mercutio'><body>A bawd, a bawd, a bawd! So ho!</body></from>
      <from secs='3' name='romeo'><body>What hast thou found?</body></from>
    </chat>
  </save>
</iq>
```

A collection MUST NOT contain more than one <previous/> and one <next/> element. If a <previous/> element is uploaded to a collection that already contains one then the older <previous/> element MUST be discarded. The same requirement applies for <next/> elements.

When a collection is retrieved (see Retrieving a Collection) the <previous/> and <next/> elements MUST appear as the first elements in the collection, whatever order they were uploaded in.

<previous/> and <next/> elements MAY be removed from a collection simply by uploading a <previous/> and/or <next/> element without any 'with' or 'start' attributes. Note: The server SHOULD NOT return an error if it finds that a link to be deleted does not exist.

**Example 31. Deleting any links to other collections**

```
<iq type='set' id='link3'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='balcony@house.capulet.com'
          start='1469-07-21T03:16:37Z'>
      <previous/>
      <next/>
    </chat>
  </save>
</iq>
```

## 5.7 Associating Attributes with a Collection

A client MAY append attributes to a collection by including an x:data form of type 'submit' (see **Data Forms** [13]) when it uploads to a collection.

A collection MUST NOT contain more than one x:data form. If a form is uploaded to a collection that already contains one then the older form element MUST be discarded. When a collection is retrieved (see Retrieving a Collection) the x:data form MUST appear as the first element in the collection after any <previous/> or <next/> elements, whatever order it was uploaded in. Upon retrieval the 'type' attribute of the form MAY be 'submit' or 'form'.

Any data forms for associating attributes are application-specific and are to be defined outside this specification. The following example shows attributes generated by a fictional application.

**Example 32. Private chat with attributes form**

```
<iq type='set' id='form1'>
  <save xmlns='urn:xmpp:archive'>
    <chat with='benvolio@montague.net'
          start='1469-07-21T03:01:54Z'>
      <to secs='0'><body>O, I am fortune's fool!</body></to>
      <from secs='4'><body>Why dost thou stay?</body></from>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE'><value>http://example.com/archiving</value></field>
        <field var='task'><value>1</value></field>
        <field var='important'><value>1</value></field>
        <field var='action_before'><value>1469-07-29T12:00:00Z</value></field>
      </x>
```

```
    </chat>
  </save>
</iq>
```

As described in **XEP-0241**, the content of the uploaded x:data form MAY be encrypted.

## 6. Automatic Archiving

If server administration policies *require* that every message is logged automatically (see Security Considerations) then:

- The server MUST enable automatic archiving when each stream is opened.
- Clients MUST NOT be allowed to disable automatic archiving.
- If the server has not received a request from a client for its user's archiving preferences (see Determining Preferences) within a few seconds of authenticating the client then the server MUST send a warning message to the client:

**Example 33. Server warns user of a legacy client about compulsory archiving**

```
<message from='capulet.com' to='juliet@capulet.com/chamber'>
  <body>WARNING: All messages that you send or
        receive will be recorded by the server.</body>
</message>
```

Otherwise:

- Automatic archiving MUST default to enabled or disabled when each stream is opened according to the last value of <auto/> element if 'scope' was set to 'global' (see Auto element), else Automatic archiving MUST default disabled.
- A client MAY enable or disable automatic archiving for messages sent over its stream at any time. Note: If the client switches off all auto-archiving then the server MUST close and archive all active collections.
- Once automatic archiving is switched on then the server MUST automatically archive messages only according to the user's Archiving Preferences.
- Note: Both parties to an ESession (see **Encrypted Session Negotiation** [14]) SHOULD either disable archiving or use an archiving method other than automatic, since ESession decryption keys are short-lived -- making it impossible to decrypt automatically archived messages.

The client can enable auto-archiving by setting the 'save' attribute to "true" or "1".

**Example 34. Client enables auto archiving**

```
<iq type='set' id='auto1'>
  <auto save='true' xmlns='urn:xmpp:archive'/>
</iq>
```

If the server does not support the saving of full message stanza or stream content and the user has specified the 'message' or 'stream' Save Mode in one of its Archiving Preferences, the server MUST return a <feature-not-implemented/> error.

**Example 35. Server Does Not Support Full Message or Stream Content**

```
<iq type='error' id='auto1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

The client can disable auto-archiving by setting the 'save' attribute to "false" or "0".

**Example 36. Client disables auto archiving**

```
<iq type='set' id='auto3'>
  <auto save='false' xmlns='urn:xmpp:archive'/>
</iq>
```

If service policies require that every message is logged automatically, the server MUST return a <not-allowed/> error.

**Example 37. Automatic Archiving is Compulsory**

```
<iq type='error' id='auto3'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

# 7. Archive Management

Manually uploaded and automatically saved collections are managed in the same way. There are three main areas of functionality related to archive management:

1. Retrieving a list of collections
2. Retrieving a collection
3. Removing a collection

Requirements and protocol flows for each of these use cases are defined below. The protocols to retrieve a list of collections and an indivdual collection both make extensive use of **Result Set Management** [15]. Clients and servers SHOULD support all the features defined in that protocol.

## 7.1 Retrieving a List of Collections

To request a list of collections, the client sends a <list/> element. The 'start' and 'end' attributes MAY be specified to indicate a date range (the values of these attributes MUST be UTC and adhere to the DateTime format specified in **XEP-0082**). The 'with' attribute MAY specify the JIDs of XMPP entities (see the JID Matching section of this document).

If the 'with' attribute is omitted then collections with any JID are returned. If only 'start' is specified then all collections on or after that date should be returned. If only 'end' is specified then all collections prior to that date should be returned.

The client SHOULD use **Result Set Management** to limit the number of collections returned by the server in a single stanza, taking care not to request a page of collections that is so big it might exceed rate limiting restrictions.

**Example 38. Requesting the first page of a list with same JID**

```
<iq type='get' id='juliet1'>
  <list xmlns='urn:xmpp:archive'
        with='juliet@capulet.com'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>30</max>
    </set>
  </list>
</iq>
```

**Example 39. Requesting the first page of a list with same JID between two times**

```
<iq type='get' id='period1'>
  <list xmlns='urn:xmpp:archive'
        with='juliet@capulet.com'
        start='1469-07-21T02:00:00Z'
        end='1479-07-21T04:00:00Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>30</max>
    </set>
  </list>
</iq>
```

**Example 40. Requesting the first page of a list after a time**

```
<iq type='get' id='list1'>
  <list xmlns='urn:xmpp:archive'
        start='1469-07-21T02:00:00Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>30</max>
    </set>
  </list>
</iq>
```

The server MUST list the collections (empty <chat/> elements including all attributes) in chronological order when responding to any request.

Note: In accordance with **Result Set Management**, the client MUST assume that the unique IDs it receives in the <first/> and <last/> elements are opaque. Servers MAY adopt a unique ID format other than the one suggested in the example above.

If no collections correspond to the request the server MUST return an empty <list/> element:

**Example 41. Receiving an empty list**

```
<iq type='result' to='romeo@montague.net/orchard' id='list1'>
  <list xmlns='urn:xmpp:archive'/>
</iq>
```

**Example 42. Requesting the second page of a list**

```
<iq type='get' id='list2'>
  <list xmlns='urn:xmpp:archive'
        start='1469-07-21T02:00:00Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>30</max>
      <after>1469-07-21T03:16:37Zbalcony@house.capulet.com</after>
    </set>
  </list>
</iq>
```

Refer to **Result Set Management** to learn more about the various ways that the pages of the list may be accessed.

## 7.2 Retrieving a Collection

To request a page of messages from a collection the client sends a <retrieve/> element. The 'with' and 'start' attributes specify the participating JID and the start time (see **XEP-0082**). Both attributes MUST be included to uniquely identify a collection.

> Note: the <retrieve/> SHALL NOT possess the 'exactmatch' attribute, because exact JID matching is always implied for this command (see the JID Matching section of this document). This is done to prevent the return of multiple collections in response to the <retrieve/> command.

The client SHOULD use **Result Set Management** to limit the number of messages returned by the server in a single stanza, taking care not to request a page of messages that is so big it might exceed rate limiting restrictions.

**Example 43. Requesting the first page of a collection**

```
<iq type='get' id='page1'>
  <retrieve xmlns='urn:xmpp:archive'
            with='juliet@capulet.com/chamber'
            start='1469-07-21T02:56:15Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>100</max>
    </set>
  </retrieve>
</iq>
```

**Example 44. Receiving the first page of a collection**

```
<iq type='result' to='romeo@montague.net/orchard' id='page1'>
  <chat xmlns='urn:xmpp:archive'
        with='juliet@capulet.com/chamber'
        start='1469-07-21T02:56:15Z'
        subject='She speaks!'
        version='4'>
    <from secs='0'><body>Art thou not Romeo, and a Montague?</body></from>
```

```
    <to secs='11'><body>Neither, fair saint, if either thee dislike.</body></to>
    .
    [97 more messages]
    .
    <from secs='9'><body>How cam'st thou hither, tell me, and wherefore?</body></from>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>0</first>
      <last>99</last>
      <count>217</count>
    </set>
  </chat>
</iq>
```

Note: In accordance with **Result Set Management**, the client MUST assume the unique IDs it receives in the <first/> and <last/> elements are opaque. Servers MAY adopt a unique ID format other than the one suggested in the example above.

If the specified collection does not exist then the server MUST return an <item-not-found/> error:

**Example 45. Unsuccessful reply**

```
<iq type='error' to='romeo@montague.net/orchard' id='page1'>
  <retrieve xmlns='urn:xmpp:archive'
            with='juliet@capulet.com/chamber'
            start='1469-07-21T02:56:15Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>100</max>
    </set>
  </retrieve>
  <error code='404' type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requested collection is empty the server MUST return an empty <chat/> element:

**Example 46. Receiving an empty collection**

```
<iq type='result' to='romeo@montague.net/orchard' id='page1'>
  <chat xmlns='urn:xmpp:archive'
        with='juliet@capulet.com/chamber'
        start='1469-07-21T02:56:15Z'
        subject='She speaks!'
        version='5'/>
</iq>
```

**Example 47. Requesting the second page of a collection**

```
<iq type='get' id='page2'>
  <retrieve xmlns='urn:xmpp:archive'
            with='juliet@capulet.com/chamber'
            start='1469-07-21T02:56:15Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>100</max>
      <after>99</after>
    </set>
  </retrieve>
</iq>
```

Refer to **Result Set Management** to learn more about the various ways that the pages of a collection may be accessed.

## 7.3 Removing a Collection

To request the removal of a single collection the client sends an empty <remove/> element. The 'with' and 'start' attributes MUST be included to uniquely identify the collection.

**Example 48. Removing a single collection**

```
<iq type='set' id='remove1'>
  <remove xmlns='urn:xmpp:archive'
          with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'/>
</iq>
```

The client MAY remove several collections at once. The 'start' and 'end' elements MAY be specified to indicate a date range. The 'with' attribute MAY specify JID of XMPP entities, see the JID Matching section of this document.

**Example 49. Removing all collections with a specified bare JID between two times**

```
<iq type='set' id='remove3'>
  <remove xmlns='urn:xmpp:archive'
          with='juliet@capulet.com'
          start='1469-07-21T02:00:00Z'
          end='1469-07-21T04:00:00Z'/>
</iq>
```

If the 'with' attribute is omitted then collections with any JID are removed.

If the end date is in the future then all collections on or after the start date are removed.

**Example 50. Removing all collections after a date**

```
<iq type='set' id='remove4'>
  <remove xmlns='urn:xmpp:archive'
          start='1469-07-21T02:00:00Z'
          end='2038-01-01T00:00:00Z'/>
</iq>
```

If the start date is before all the collections in the archive then all collections prior to the end date are removed.

**Example 51. Removing all collections before a date**

```
<iq type='set' id='remove5'>
  <remove xmlns='urn:xmpp:archive'
          start='0000-01-01T00:00:00Z'
          end='1469-07-21T04:00:00Z'/>
</iq>
```

**Example 52. Removing all collections**

```
<iq type='set' id='remove6'>
  <remove xmlns='urn:xmpp:archive'/>
</iq>
```

If the value of the optional 'open' attribute is set to 'true' then only collections that are currently being recorded automatically by the server (see Automatic Archiving) are removed.

**Example 53. Removing a collection being recorded by the server**

```
<iq type='set' id='remove7'>
  <remove xmlns='urn:xmpp:archive'
          with='juliet@capulet.com/chamber'
          open='true'/>
</iq>
```

**Example 54. Removing all collections being recorded by the server**

```
<iq type='set' id='remove8'>
  <remove xmlns='urn:xmpp:archive'
          open='true'/>
</iq>
```

If the specified collection (or collections) do not exist then the server MUST return an <item-not-found/> error:

**Example 55. Unsuccessful reply**

```
<iq type='error' to='romeo@montague.net/orchard' id='remove1'>
  <remove xmlns='urn:xmpp:archive'
          with='juliet@capulet.com/chamber'
          start='1469-07-21T02:56:15Z'/>
  <error code='404' type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

## 8. Replication

This section describes how a client can replicate an archive locally. [16] The existence of a local copy of the archive enables clients to search the content of all messages (including collections saved by another client machine). [17]

The client can "synchronize" its local copy of the archive with the "master" archive on the server at any time. The first step is to request the list of collections that the server has modified (created, changed, or removed) in its master archive since the last update to the client's copy of the archive.

Replication uses the <modified/> element. The list of collections that have been modified since a given time is requested by sending a <modified/> element to the server. The server then returns the list of collections that have been created, changed, or removed. A collection that has been created or changed is specified with a <changed/> element (with version zero for created collections), and a collection that has been removed is specified with a <removed/> element.

When requesting the list of modified collections, the client MUST embed appropriate **Result Set Management** data in the <modified/> element. The <modified/> element MUST include a 'start' attribute that specifies a UTC datetime (see **XEP-0082**) that it has previously received from the server or that it has determined locally (e.g., when synchronizing for the first time the client SHOULD choose a suitable time for the first page request, such as 1970-01-01T00:00:00Z).

**Example 56. Requesting a page of modifications**

```
<iq type='get' id='sync1'>
  <modified xmlns='urn:xmpp:archive'
            start='1469-07-21T01:14:47Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <max>50</max>
    </set>
  </modified>
</iq>
```

The server MUST return the changed collections in the chronological order that they were changed (most recent last). If a collection has been modified, created, or removed *after* the time specified by the 'start' attribute, then the server MUST include it in the returned result set page of collections (unless the specified maximum page size would be exceeded). Each <changed/> or <removed/> collection element (for modified/created, or removed collections respectively) in the returned list MUST include the 'with' and 'start' attribues. The XML character data of the <last/> element is a unique, persistent identifier created by the server, which MUST be treated as opaque by the client.

**Example 57. Receiving a page of modifications**

```
<iq type='result' to='romeo@montague.net/orchard' id='sync1'>
  <modified xmlns='urn:xmpp:archive'>
    <changed with='juliet@capulet.com/chamber'
             start='1469-07-21T02:56:15Z'
             version='0'/>
    [ ... up to 48 more collections ... ]
    <removed with='balcony@house.capulet.com'
             start='1469-07-21T03:16:37Z'
             version='3'/>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <last>ja923ljasnvla09woei777</last>
      <count>1372</count>
    </set>
  </modified>
```

```
</iq>
```

Note: The server should remember the 'with' and 'start' attribues and the time of removal of all deleted collections. If this "state" cannot be maintained indefinitely, then unless all the user's clients replicate before the server deletes its memory of a removal it will not be reflected in all the local copies of the archive.

Note: Along with its copy of the archive the client SHOULD save the most recent <last/> identifier that it received from the server. The next time it synchronizes with the server it SHOULD specify that identifier when requesting the first result set page by including it as the XML character data of the <after/> element in Result Set Management.

**Example 58. Requesting the next page of modifications**

```
<iq type='get' id='sync2'>
  <modified xmlns='urn:xmpp:archive'
            start='1469-07-21T01:14:47Z'>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <after>ja923ljasnvla09woei777</after>
      <max>50</max>
    </set>
  </modified>
</iq>
```

After receiving each result set page the client SHOULD delete from its local archive any collections that have been removed from the master archive. The client should also retrieve from the server the content of each collection that has been modified (see Retrieving a Collection) and add it to its local copy of the archive (deleting any older version of the same collection that it may already have).

# 9. Determining Server Support

A client discovers whether its server supports this protocol using **Service Discovery** [18].

**Example 59. Client Service Discovery request**

```
<iq from='romeo@montague.net/orchard'
    id='disco1'
    to='montague.net'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

For each feature defined herein, if the server supports that feature it MUST return a <feature/> element with the 'var' attribute set to 'urn:xmpp:archive:[name]', where '[name]' is 'auto' for the Automatic Archiving feature, 'manage' for the Archive Management feature, 'manual' for the Manual Archiving feature, and 'pref' for the Archiving Preferences feature.

**Example 60. Server Service Discovery response**

```
<iq from='montague.net'
    id='disco1'
    to='romeo@montague.net/orchard'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='urn:xmpp:archive'/>
    <feature var='urn:xmpp:archive:auto'/>
    <feature var='urn:xmpp:archive:manage'/>
    <feature var='urn:xmpp:archive:manual'/>
    <feature var='urn:xmpp:archive:pref'/>
  </query>
</iq>
```

# 10. Implementation Notes

## 10.1 JID Matching

<list/>, <remove/> commands and <item/> element in archiving preferences require the ability to match multiple collections by given JID. Therefore, the following rules apply.

1. If the JID is of the form <localpart@domain.tld/resource>, only this particular JID matches.
2. If the JID is of the form <localpart@domain.tld>, any resource matches.
3. If the JID is of the form <domain.tld>, any node matches.

However, having these rules only would make impossible a match like "all collections having JID exactly equal to bare JID/domain JID". Therefore, when the 'exactmatch' attribute is set to "true" or "1" [19] on the <list/>, <remove/> or <item/> element, a JID value such as "example.com" matches that exact JID only rather than <*@example.com>, <*@example.com/*>, or <example.com/*>, and a JID value such as "localpart@example.com" matches that exact JID only rather than <localpart@example.com/*>.

### 10.2 Time Synchronization

When creating a new collection, it is RECOMMENDED that the client synchronizes the collection start time that it sends to the server with server time. This is important since the user may subsequently retrieve the archived collection using client machines whose UTC clocks are not synchronized with the client machine that uploaded the collection. (i.e. Either or both of the clients' UTC clocks may be wrong.) The client can achieve this synchronization with server time by using **Entity Time** [20] to estimate the difference between the server and client UTC clocks.

When retrieving collections, it is RECOMMENDED that the client adjusts the start times of the collections it receives from server to be synchronized with the clock of the client machine.

### 10.3 Bandwidth Considerations

When uploading messages using manual archiving, a client SHOULD NOT upload one message at a time to the server since this increases both bandwidth consumption and the total number of transactions. It is instead RECOMMENDED that clients upload messages only when the conversation thread *appears* to be terminated, e.g. when the user closes the chat window. If the user reopens the window and the thread continues then the client should append the new messages to the collection when the user closes the window again.

### 10.4 Storage Considerations

Server implementations SHOULD give system administrators the option to disable support for both automatic and manual archiving, since archived conversations can consume significant storage space.

### 10.5 Conversations Tracking In Automatic Archiving

When starting automatic archiving for a new conversation, it is possible that the initial message might not be enough to determine the full JID of the recipient. For example, if the conversation is initiated by a client whose server performs automatic archiving and that client does not know which full JID it ought to use for the recipient, it will send the initial message to bare JID. As a result, automatic archiving will create a collection that is identified by the bare JID of the recipient.

However, once the client receives a reply from the contact, it knows the full JID of the recipient. At that point, the initial collection can be adjusted, changing the bare JID to the full JID of the recipient.

The server SHOULD attempt to perform conversation tracking and JID adjustment to ensure that the identifying JID for the collection reflects the actual full JID used during conversation rather than initial bare JID used when the conversation started. However, the server MAY use the bare JID for the collection if there is evidence that the conversation involves several different full JIDs, such as receiving messages from different full JIDs with the same <thread/> element.

## 11. Stream Feature

Although message archiving is not negotiated between a client and its server as part of stream negotiation, a server MAY advertise a stream feature of "urn:xmpp:archive" during stream setup (via the <feature/> element, which MUST contain an empty <optional/> child), and MUST do so if automatic archiving is on by default (if so, the <feature/> element MUST include an empty <default/> child).

**Example 61. Stream Feature**

```
<feature xmlns='urn:xmpp:archive'>
  <optional/>
</feature>
```

**Example 62. Stream Feature (Automatic Archiving on By Default)**

```
<feature xmlns='urn:xmpp:archive'>
  <optional/>
  <default/>
</feature>
```

# 12. Security Considerations

## 12.1 Automatic Archiving Defaulting to On

If automatic archiving defaults to enabled then that creates serious privacy issues for users of legacy clients that do not support this protocol, and (more seriously) for those contacts who they unwittingly mislead by agreeing to disable logging (via the 'logging' field defined in **XEP-0155**).

If a server deployment enables automatic archiving by default, then it MUST return a stream feature containing an empty <default/> element (see the Stream Feature section of this document).

## 12.2 Store Headers

The client that originates a message MAY specify a 'false' value for the 'store' header (see **Stanza Headers and Internet Metadata** [21]). The recipient MUST NOT archive such a message or any of the information it contains.

If the sender plans to use 'store' headers it MUST use Service Discovery to determine whether or not the recipient supports them. Note: Since servers are not required to check the content of message stanzas for headers, if the recipient is using automatic archiving then it MUST indicate that it does not support 'store' headers.

If the recipient does not support 'store' headers, then the sender MUST confirm with its human user (if any) before sending such a message.

# 13. IANA Considerations

No interaction with the **Internet Assigned Numbers Authority (IANA)** [22] is required as a result of this document.

# 14. XMPP Registrar Considerations

## 14.1 Protocol Namespace

The **XMPP Registrar** [23] includes "urn:xmpp:archive:data" in its registry of protocol namespaces (see <http://xmpp.org/registrar/namespaces.html>).

## 14.2 Service Discovery Features

The XMPP Registrar includes the following features in its registry of service discovery features (see <http://xmpp.org/registrar/disco-features.html>):

- urn:xmpp:archive:auto
- urn:xmpp:archive:manage
- urn:xmpp:archive:manual
- urn:xmpp:archive:pref

# 15. XML Schema

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0136: http://www.xmpp.org/extensions/xep-0136.html
    </xs:documentation>
  </xs:annotation>

  <xs:annotation>
    <xs:documentation>
      The allowable root elements for the namespace defined
      herein are:
        - auto
        - chat
        - itemremove
```

```
      - list
      - modified
      - pref
      - remove
      - retrieve
      - save
  </xs:documentation>
</xs:annotation>

<xs:element name='auto'>
  <xs:complexType>
    <xs:sequence>
      <xs:any processContents='lax' namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='save' type='xs:boolean' use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='changed'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='start' type='xs:dateTime' use='required'/>
        <xs:attribute name='with' type='xs:string' use='required'/>
        <xs:attribute name='version' type='xs:nonNegativeInteger' use='required'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='chat'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='from' type='messageType'/>
      <xs:element name='next' type='linkType'/>
      <xs:element ref='note'/>
      <xs:element name='previous' type='linkType'/>
      <xs:element name='to' type='messageType'/>
      <xs:any processContents='lax' namespace='##other'/>
    </xs:choice>
    <xs:attribute name='start' type='xs:dateTime' use='required'/>
    <xs:attribute name='subject' type='xs:string' use='optional'/>
    <xs:attribute name='thread' use='optional' type='xs:string'/>
    <xs:attribute name='version' use='optional' type='xs:nonNegativeInteger'/>
    <xs:attribute name='with' type='xs:string' use='required'/>
  </xs:complexType>
</xs:element>

<xs:complexType name='messageType'>
  <xs:sequence>
    <xs:element name='body' type='xs:string' minOccurs='0' maxOccurs='unbounded'/>
    <xs:any processContents='lax' namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='jid' type='xs:string' use='optional'/>
  <xs:attribute name='name' type='xs:string' use='optional'/>
  <xs:attribute name='secs' type='xs:nonNegativeInteger' use='optional'/>
  <xs:attribute name='utc' type='xs:dateTime' use='optional'/>
</xs:complexType>

<xs:complexType name='linkType'>
  <xs:simpleContent>
    <xs:extension base='empty'>
      <xs:attribute name='start' type='xs:dateTime' use='optional'/>
      <xs:attribute name='with' type='xs:string' use='optional'/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name='default'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
```

```
        <xs:attribute name='expire' type='xs:nonNegativeInteger' use='optional'/>
        <xs:attribute name='otr' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='approve'/>
              <xs:enumeration value='concede'/>
              <xs:enumeration value='forbid'/>
              <xs:enumeration value='oppose'/>
              <xs:enumeration value='prefer'/>
              <xs:enumeration value='require'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='save' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='body'/>
              <xs:enumeration value='false'/>
              <xs:enumeration value='message'/>
              <xs:enumeration value='stream'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='unset' use='optional' type='xs:boolean'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='feature'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='optional' minOccurs='1' maxOccurs='1'/>
      <xs:element ref='default' minOccurs='0' maxOccurs='1'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name='item'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='exactmatch' type='xs:boolean' use='optional'/>
        <xs:attribute name='expire' type='xs:nonNegativeInteger' use='optional'/>
        <xs:attribute name='jid' use='required' type='xs:string'/>
        <xs:attribute name='otr' use='optional'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='approve'/>
              <xs:enumeration value='concede'/>
              <xs:enumeration value='forbid'/>
              <xs:enumeration value='oppose'/>
              <xs:enumeration value='prefer'/>
              <xs:enumeration value='require'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='save' use='optional'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='body'/>
              <xs:enumeration value='false'/>
              <xs:enumeration value='message'/>
              <xs:enumeration value='stream'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<xs:element name='list'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='chat' minOccurs='0' maxOccurs='unbounded'/>
      <xs:any processContents='lax' namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='end' type='xs:dateTime' use='optional'/>
    <xs:attribute name='exactmatch' type='xs:boolean' use='optional'/>
    <xs:attribute name='start' type='xs:dateTime' use='optional'/>
    <xs:attribute name='with' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='method'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='type' type='xs:string' use='required'/>
        <xs:attribute name='use' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='concede'/>
              <xs:enumeration value='forbid'/>
              <xs:enumeration value='prefer'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='modified'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='changed' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element ref='removed' minOccurs='0' maxOccurs='unbounded'/>
      <xs:any processContents='lax' namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='start' type='xs:dateTime' use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='note'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute name='utc' type='xs:dateTime' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='pref'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='auto' minOccurs='0' maxOccurs='1'/>
      <xs:element ref='default' minOccurs='0' maxOccurs='1'/>
      <xs:element ref='item' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element ref='method' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name='itemremove'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='item' minOccurs='1' maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name='remove'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='end' type='xs:dateTime' use='optional'/>
          <xs:attribute name='exactmatch' type='xs:boolean' use='optional'/>
          <xs:attribute name='open' use='optional' type='xs:boolean'/>
          <xs:attribute name='start' type='xs:dateTime' use='required'/>
          <xs:attribute name='with' type='xs:string' use='required'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='removed'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='start' type='xs:dateTime' use='required'/>
          <xs:attribute name='with' type='xs:string' use='required'/>
          <xs:attribute name='version' type='xs:nonNegativeInteger' use='required'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='retrieve'>
    <xs:complexType>
      <xs:sequence>
        <xs:any processContents='lax' namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
      </xs:sequence>
      <xs:attribute name='start' type='xs:dateTime' use='required'/>
      <xs:attribute name='with' type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='save'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='chat' minOccurs='1' maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

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

</xs:schema>
```

## 16. Acknowledgements

Thanks to Alexey Melnikov and Alexander Tsvyashchenko for their comments and suggestions.

---

# Appendices

---

### Appendix A: Document Information

Series: XEP
Number: 0136
Publisher: XMPP Standards Foundation
Status: Draft
Type: Standards Track
Version: 1.2
Last Updated: 2010-06-21
Approving Body: XMPP Council
Dependencies: XMPP Core, XEP-0004, XEP-0030, XEP-0059, XEP-0060

Supersedes: None
Superseded By: None
Short Name: archive
Schema: <http://www.xmpp.org/schemas/archive.xsd>
Source Control: HTML
This document in other formats: XML PDF

---

## Appendix B: Author Information


### Ian Paterson

Email: ian.paterson@clientside.co.uk
JabberID: ian@zoofy.com

### Jon Perlow

Email: jonp@google.com
JabberID: jonp@google.com

### Peter Saint-Andre

Email: stpeter@jabber.org
JabberID: stpeter@jabber.org
URI: https://stpeter.im/

### Justin Karneges

Email: justin@affinix.com
JabberID: justin@andbit.net

### Alexander Tsvyashchenko

Email: lists@ndl.kiev.ua

### Yann Leboulanger

Email: asterix@lagaule.org


---

## Appendix C: Legal Notices

Foundation or such author has been advised of the possibility of such damages.

#### IPR 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 <http://xmpp.org/about-xmpp/xsf/xsf-ipr-policy/> or obtained by writing to XMPP Standards Foundation, 1899 Wynkoop Street, Suite 600, Denver, CO 80202 USA).

---

## Appendix D: Relation to XMPP

The Extensible Messaging and Presence Protocol (XMPP) is defined in the XMPP Core (RFC 6120) and XMPP IM (RFC 6121) specifications contributed by the XMPP Standards Foundation to the Internet Standards Process, which is managed by the Internet Engineering Task Force in accordance with RFC 2026. Any protocol defined in this document has been developed outside the Internet Standards Process and is to be understood as an extension to XMPP rather than as an evolution, development, or modification of XMPP itself.

---

## Appendix E: Discussion Venue

The primary venue for discussion of XMPP Extension Protocols is the <standards@xmpp.org> discussion list.

Discussion on other xmpp.org discussion lists might also be appropriate; see <http://xmpp.org/about/discuss.shtml> for a complete list.

Errata can be sent to <editor@xmpp.org>.

---

## Appendix F: Requirements Conformance

The following requirements keywords as used in this document are to be interpreted as described in RFC 2119: "MUST", "SHALL", "REQUIRED"; "MUST NOT", "SHALL NOT"; "SHOULD", "RECOMMENDED"; "SHOULD NOT", "NOT RECOMMENDED"; "MAY", "OPTIONAL".

---

## Appendix G: Notes

1. The XMPP Council is a technical steering committee, authorized by the XSF Board of Directors and elected by XSF members, that approves of new XMPP Extensions Protocols and oversees the XSF's standards process. For further information, see <http://xmpp.org/council/>.

2. The "OTR" mode for message archiving is not to be confused with the "OTR" technology for "off-the-record communications" described at <http://www.cypherpunks.ca/otr/>.

3. In accordance with Section 3.2.2.1 of **XML Schema Part 2: Datatypes**, the allowable lexical representations for the xs:boolean datatype are the strings "0" and "false" for the concept 'false' and the strings "1" and "true" for the concept 'true'; implementations MUST support both styles of lexical representation.

4. Stream compression typically does not mitigate bandwidth and storage issues since clients running in constrained runtime environments typically cannot take advantage of stream compression (no binary data, only XML, may be transfered).

5. XEP-0201: Best Practices for Message Threads <http://xmpp.org/extensions/xep-0201.html>.

6. XEP-0155: Stanza Session Negotiation <http://xmpp.org/extensions/xep-0155.html>.

7. If a client (or user) acts in bad faith then its contacts cannot prevent it from archiving conversations.

8. XEP-0082: XMPP Date and Time Profiles <http://xmpp.org/extensions/xep-0082.html>.

9. XEP-0201: Best Practices for Message Threads <http://xmpp.org/extensions/xep-0201.html>.

10. XEP-0116: Encrypted Session Negotiation <http://xmpp.org/extensions/xep-0116.html>.

11. XEP-0241: Encryption of Archived Messages <http://xmpp.org/extensions/xep-0241.html>.

12. XEP-0045: Multi-User Chat <http://xmpp.org/extensions/xep-0045.html>.

13. XEP-0004: Data Forms <http://xmpp.org/extensions/xep-0004.html>.

14. XEP-0116: Encrypted Session Negotiation <http://xmpp.org/extensions/xep-0116.html>.

15. XEP-0059: Result Set Management <http://xmpp.org/extensions/xep-0059.html>.

16. Clients that run in constrained environments may not be able to implement replication if they are prevented from accessing (sufficient) local storage.

17. Since collections SHOULD be stored on the server in a form that it cannot decrypt, server-side searching of the content of messages is beyond the scope of this protocol.

18. XEP-0030: Service Discovery <http://xmpp.org/extensions/xep-0030.html>.

19. In accordance with Section 3.2.2.1 of **XML Schema Part 2: Datatypes**, the allowable lexical representations for the xs:boolean datatype are the strings "0" and "false" for the concept 'false' and the strings "1" and "true" for the concept 'true'; implementations MUST support both styles of lexical representation.

20. XEP-0202: Entity Time <http://xmpp.org/extensions/xep-0202.html>.

21. XEP-0131: Stanza Headers and Internet Metadata <http://xmpp.org/extensions/xep-0131.html>.

22. 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/>.

23. 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 <http://xmpp.org/registrar/>.

---

## Appendix H: Revision History

Note: Older versions of this specification might be available at http://xmpp.org/extensions/attic/

### Version 1.2 (2010-06-21)

Added persistent auto save setting and ability to control settings per chat session; XMPP Council added a statement regarding a future review and simplification of this protocol.

(at/yl)

### Version 1.1 (2009-09-23)

Moved JID matching text to a dedicated section and clarified matching rules; described implementation notes regarding server interpretation of archiving preferences and conversation tracking.

(at/psa)

### Version 1.0 (2008-07-16)

Per a vote of the XMPP Council, advanced status to Draft; concurrently, the XMPP Registrar issued the urn:xmpp:archive namespace.

(psa)

### Version 0.18 (2008-05-19)

Corrected and clarified usage of exactmatch attribute; removed out-of-date archive, delete, and keys elements from schema.

(psa)

### Version 0.17 (2008-04-30)

Split encryption content off into new specification.

(psa)

### Version 0.16 (2008-04-15)

Addressed last call comments; clarified syntax of the pref and chat elements; deferred definition of data forms for attribute association to future specifications; removed registration of field standardization fields.

(psa)

### Version 0.15 (2008-03-03)

Changed temporary namespace per XEP-0053 procedures; moved all encrypted-related

material to dedicated section; removed text about file format; modified replication to use start attribute, not after element; added version attribute for tracking collection changes; added itemremove element to delete all preferences for a contact.

(psa)

**Version 0.14 (2007-08-16)**

Clarified text regarding preference syntax; completed copy edit.

(psa)

**Version 0.13 (2007-01-08)**

Harmonized stanza session negotiation of message logging settings with XEP-0155; defined stream feature.

(psa/ip)

**Version 0.12 (2006-11-23)**

All modes allow multiple body children of to and from elements; changed namespace and collection element name to chat; renamed all value of save attribute to message; added stream value of the save attribute, thread attribute, save wrapper element, and Linking Collections and Associating Attributes sections

(ip)

**Version 0.11 (2006-11-06)**

Added more otr attribute values and clarified their meanings, changed the names of the use attribute values

(ip)

**Version 0.10 (2006-10-11)**

Added auto-archiving warning for legacy clients; corrected examples

(ip)

**Version 0.9 (2006-10-02)**

Added method child elements and expire attribute to pref element

(ip)

**Version 0.8 (2006-09-29)**

Server generates encryption secrets for auto-archiving; specified use of W3C XML Encryption standard; enabled replacement of keys encrypted with an obsolete public key; enabled removal of open collections

(ip)

**Version 0.7 (2006-09-08)**

Added preferences, results set management and notes; reinstated encryption and replication; simplified auto-archiving and off-the-record (with XEP-0155); many minor changes

(ip)

**Version 0.6 (2006-08-18)**

Added unset value for save attribute and added service attribute on default element; added source attribute on record element; specified that services should (not must) support save mode for particular contacts.

(jp/psa)

**Version 0.5 (2006-05-03)**

Integrated text from server-side archiving proposal; added partial support to collection retrieval; harmonized XML formats and namespaces; defined XMPP Registrar considerations and XML schema.

(psa/jp/jk)

**Version 0.4 (2005-12-21)**

Added Replication and Searching section, partial attribute; minor improvements

(ip)

**Version 0.3 (2005-10-21)**

Added more examples to Removing Collections

(ip)

**Version 0.2 (2005-04-18)**

Complete rewrite.

(ip)

**Version 0.1 (2004-06-04)**

Initial version.

(jk)

---

END