



XMPP

XEP-0198: Stream Management

Justin Karneges

<mailto:justin@karneges.com>
<xmpp:justin@andbit.net>

Peter Saint-Andre

<mailto:stpeter@stpeter.im>
<xmpp:stpeter@jabber.org>
<https://stpeter.im/>

Joe Hildebrand

<mailto:jhildebr@cisco.com>
<xmpp:hildjj@jabber.org>

Fabio Forno

<mailto:fabio.forno@gmail.com>
<xmpp:ff@jabber.blundo.com>

Dave Cridland

<mailto:dave@helloworld.com>
<xmpp:dwd@dave.cridland.net>

Matthew Wild

<mailto:mwild1@gmail.com>
<xmpp:me@matthewwild.co.uk>

Thilo Molitor

<mailto:thilo+xmpp@eightysoft.de>
<xmpp:thilo.molitor@juforum.de>

2022-10-05

Version 1.6.1

Status	Type	Short Name
Draft	Standards Track	sm

This specification defines an XMPP protocol extension for active management of an XML stream between two XMPP entities, including features for stanza acknowledgements and stream resumption.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the [XMPP Standards Foundation](#) (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

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

Liability

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

Conformance

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

Contents

1	Introduction	1
2	Stream Feature	1
3	Enabling Stream Management	2
4	Acks	3
5	Resumption	6
6	Error Handling	10
7	Stream Closure	11
8	Scenarios	11
8.1	Basic Acking Scenario	11
8.2	Efficient Acking Scenario	13
9	SASL2 And BIND2 Interaction	14
9.1	Enable Stream Management Inline	15
9.1.1	Failed To Enable Stream Management	15
9.2	Inline Stream Resumption	15
9.2.1	Failed Resumption	15
9.3	Inlining Examples	16
10	Security Considerations	17
11	IANA Considerations	18
12	XMPP Registrar Considerations	18
12.1	Protocol Namespaces	18
12.2	Protocol Versioning	18
12.3	Stream Features	18
13	XML Schemas	18
14	Acknowledgements	21

1 Introduction

[XMPP Core](#) ¹ defines the fundamental streaming XML technology used by XMPP (i.e., stream establishment and termination including authentication and encryption). However, the core XMPP specification does not provide tools for actively managing a live XML stream.

The basic concept behind stream management is that the initiating entity (either a client or a server) and the receiving entity (a server) can exchange "commands" for active management of the stream. The following stream management features are of particular interest because they are expected to improve network reliability and the end-user experience:

- Stanza Acknowledgements -- the ability to know if a stanza or series of stanzas has been received by one's peer.
- Stream Resumption -- the ability to quickly resume a stream that has been terminated.

Stream management implements these features using short XML elements at the root stream level. These elements are not "stanzas" in the XMPP sense (i.e., not <iq/>, <message/>, or <presence/> stanzas as defined in RFC 6120) and are not counted or acked in stream management, since they exist for the purpose of managing stanzas themselves.

Stream management is used at the level of an XML stream. To check TCP connectivity underneath a given stream, it is RECOMMENDED to use whitespace keepalives (see RFC 6120), [XMPP Ping \(XEP-0199\)](#) ², or TCP keepalives. By contrast with stream management, [Advanced Message Processing \(XEP-0079\)](#) ³ and [Message Delivery Receipts \(XEP-0184\)](#) ⁴ define acks that are sent end-to-end over multiple streams; these facilities are useful in special scenarios but are unnecessary for checking of a direct stream between two XMPP entities.

Note: Stream Management can be used for server-to-server streams as well as for client-to-server streams. However, for convenience this specification discusses client-to-server streams only. The same principles apply to server-to-server streams.

2 Stream Feature

The server returns a stream header to the client along with stream features, where the features include an <sm/> element qualified by the 'urn:xmpp:sm:3' namespace (see [Namespace Versioning](#) regarding the possibility of incrementing the version number).

Note: The client cannot negotiate stream management until it has authenticated with the server and has bound a resource; see below for specific restrictions.

Listing 1: Server sends new stream header along with stream features

¹RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <<http://tools.ietf.org/html/rfc6120>>.

²XEP-0199: XMPP Ping <<https://xmpp.org/extensions/xep-0199.html>>.

³XEP-0079: Advanced Message Processing <<https://xmpp.org/extensions/xep-0079.html>>.

⁴XEP-0184: Message Delivery Receipts <<https://xmpp.org/extensions/xep-0184.html>>.

```
<stream:stream
  from='example.com'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'
  version='1.0'>

<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind' />
  <sm xmlns='urn:xmpp:sm:3' />
</stream:features>
```

3 Enabling Stream Management

To enable use of stream management, the client sends an `<enable/>` command to the server.

Listing 2: Client enables stream management

```
<enable xmlns='urn:xmpp:sm:3' />
```

If the client wants to be allowed to resume the stream, it includes a boolean 'resume' attribute, which defaults to false⁵. For information about resuming a previous session, see the [Resumption](#) section of this document.

The `<enable/>` element MAY include a 'max' attribute to specify the client's preferred maximum resumption time in seconds.

Upon receiving the enable request, the server MUST reply with an `<enabled/>` element or a `<failed/>` element qualified by the 'urn:xmpp:sm:3' namespace. The `<failed/>` element indicates that there was a problem establishing the stream management "session". The `<enabled/>` element indicates successful establishment of the stream management session.

Listing 3: Server enables stream management

```
<enabled xmlns='urn:xmpp:sm:3' />
```

The parties can then use stream management features defined below.

If the server allows session resumption, it MUST include a 'resume' attribute set to a value of "true" or "1"⁶.

Listing 4: Server enables stream management with session resumption

```
<enabled xmlns='urn:xmpp:sm:3' id='some-long-sm-id' resume='true' />
```

⁵In 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.

⁶In 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.

The <enabled/> element MAY include a 'max' attribute to specify the server's preferred maximum resumption time.

The <enabled/> element MAY include a 'location' attribute to specify the server's preferred IP address or hostname (optionally with a port) for reconnection, in the form specified in Section 4.9.3.19 of RFC 6120 (i.e., "domainpart:port", where IPv6 addresses are enclosed in square brackets "[...]" as described in RFC 5952⁷); if reconnection to that location fails, the standard XMPP connection algorithm specified in RFC 6120 applies.

The client MUST NOT attempt to negotiate stream management until it is authenticated; i.e., it MUST NOT send an <enable/> element until after authentication (such as SASL, [Non-SASL Authentication \(XEP-0078\)](#)⁸ or [Server Dialback \(XEP-0220\)](#)⁹) has been completed successfully. For client-to-server connections, the client MUST NOT attempt to enable stream management until after it has completed Resource Binding *unless it is resuming a previous session* (see [Resumption](#)).

The server SHALL enforce this order and return a <failed/> element in response if the order is violated (see [Error Handling](#)).

Listing 5: Server returns error if client attempts to enable stream management before resource binding

```
<failed xmlns='urn:xmpp:sm:3'>
  <unexpected-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
</failed>
```

Note that a client SHALL only make at most one attempt to enable stream management. If a server receives a second <enable/> element it SHOULD respond with a stream error, thus terminating the client connection.

Listing 6: Server returns error if client attempts to enable stream management more than once

```
<failed xmlns='urn:xmpp:sm:3'>
  <unexpected-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
</failed>
```

4 Acks

After enabling stream management, the client or server can send ack elements at any time over the stream. An ack element is one of the following:

- The <a/> element is used to **answer** a request for acknowledgement or to send an unrequested ack.

⁷RFC 5952: A Recommendation for IPv6 Address Text Representation <<http://tools.ietf.org/html/rfc5952>>.

⁸XEP-0078: Non-SASL Authentication <<https://xmpp.org/extensions/xep-0078.html>>.

⁹XEP-0220: Server Dialback <<https://xmpp.org/extensions/xep-0220.html>>.

- The `<r/>` element is used to **request** acknowledgement of received stanzas.

The following attribute is defined:

- The 'h' attribute identifies the last **handled** stanza (i.e., the last stanza that the server will acknowledge as having received).

An `<a/>` element **MUST** possess an 'h' attribute.

The `<r/>` element has no defined attributes.

Definition: Acknowledging a previously-received ack element indicates that the stanza(s) sent since then have been "handled" by the server. By "handled" we mean that the server has accepted responsibility for a stanza or stanzas (e.g., to process the stanza(s) directly, deliver the stanza(s) to a local entity such as another connected client on the same server, or route the stanza(s) to a remote entity at a different server); until a stanza has been affirmed as handled by the server, that stanza is the responsibility of the sender (e.g., to resend it or generate an error if it is never affirmed as handled by the server).

Receipt of an `<r/>` element does not imply that new stanzas have been transmitted by the peer; receipt of an `<a/>` element only indicates that new stanzas have been processed if the 'h' attribute has been incremented.

The value of 'h' starts at zero at the point stream management is enabled or requested to be enabled (see note below). The value of 'h' is then incremented to one for the first stanza handled and incremented by one again with each subsequent stanza handled. In the unlikely case that the number of stanzas handled during a stream management session exceeds the number of digits that can be represented by the unsignedInt datatype as specified in [XML Schema Part 2](#)¹⁰ (i.e., 2^{32}), the value of 'h' SHALL be reset from $2^{32}-1$ back to zero (rather than being incremented to 2^{32}).

Note: Each entity maintains two counters for any given stream: a counter of stanzas it has sent, and a counter of stanzas it has received and handled ('h'). The counter for an entity's own sent stanzas is set to zero and started after sending either `<enable/>` or `<enabled/>`. The counter for the received stanzas ('h') is set to zero and started after receiving either `<enable/>` or `<enabled/>`.

The following annotated example shows a message sent by the client, a request for acknowledgement, and an ack of the stanza.

Listing 7: Simple stanza acking

```

<!--{}- Client -{}->
<enable xmlns='urn:xmpp:sm:3' />

<!--{}- Client sets outbound count to zero. -{}->

<message from='laurence@example.net/churchyard'
         to='juliet@example.com'

```

¹⁰XML Schema Part 2: Datatypes [<http://www.w3.org/TR/xmlschema11-2/>](http://www.w3.org/TR/xmlschema11-2/).

```

        xml:lang='en'>
    <body>
        I'll send a friar with speed, to Mantua,
        with my letters to thy lord.
    </body>
</message>

<!-- Note that client need not wait for a response. -->

<!-- Server -->
<enabled xmlns='urn:xmpp:sm:3' />

<!--
    Server receives enable, and responds,
    setting both inbound and outbound counts
    to zero.

    In addition, client sets inbound count to zero.
-->

<!-- Client -->
<r xmlns='urn:xmpp:sm:3' />

<!-- Server -->
<a xmlns='urn:xmpp:sm:3' h='1' />

```

When an `<r/>` element ("request") is received, the recipient MUST acknowledge it by sending an `<a/>` element to the sender containing a value of 'h' that is equal to the number of stanzas handled by the recipient of the `<r/>` element. The response SHOULD be sent as soon as possible after receiving the `<r/>` element, and MUST NOT be withheld for any condition other than a timeout. For example, a client with a slow connection might want to collect many stanzas over a period of time before acking, and a server might want to throttle incoming stanzas. The sender does not need to wait for an ack to continue sending stanzas.

Either party MAY send an `<a/>` element at any time (e.g., after it has received a certain number of stanzas, or after a certain period of time), even if it has not received an `<r/>` element from the other party. It is RECOMMENDED that initiating entities (usually clients) send an `<a/>` element right before they gracefully close the stream, in order to inform the peer about received stanzas. Otherwise it can happen that stanzas are re-sent (usually by the server) although they were actually received.

When a party receives an `<a/>` element, it SHOULD keep a record of the 'h' value returned as the sequence number of the last handled outbound stanza for the current stream (and discard the previous value).

If a stream ends and it is not resumed within the time specified in the original `<enabled/>` element, the sequence number and any associated state MAY be discarded by both parties. Before the session state is discarded, implementations SHOULD take alternative action regarding any unhandled stanzas (i.e., stanzas sent after the most recent 'h' value received):

- A server SHOULD treat unacknowledged stanzas in the same way that it would treat a stanza sent to an unavailable resource, by either returning an error to the sender, delivery to an alternate resource, or committing the stanza to offline storage. (Note that servers SHOULD add a delay element with the original (failed) delivery timestamp, as per [Delayed Delivery \(XEP-0203\)](#) ¹¹).
- A user-oriented client SHOULD try to silently resend the stanzas upon reconnection or inform the user of the failure via appropriate user-interface elements. Clients SHOULD also add a delay element with the original (failed) send timestamp, so that the original send date is preserved. Otherwise receiving clients could only display the reconnection timestamp of the sending client, which may confuse users.

Because unacknowledged stanzas might have been received by the other party, resending them might result in duplicates; there is no way to prevent such a result in this protocol, although use of the XMPP 'id' attribute on all stanzas can at least assist the intended recipients in weeding out duplicate stanzas.

5 Resumption

It can happen that an XML stream is terminated unexpectedly (e.g., because of network outages). In this case, it is desirable to quickly resume the former stream rather than complete the tedious process of stream establishment, roster retrieval, and presence broadcast.

In addition, this protocol exchanges the sequence numbers of the last received stanzas on the previous connection, allowing entities to establish definitively which stanzas require retransmission and which do not, eliminating duplication through replay.

To request that the stream will be resumable, when enabling stream management the client MUST add a 'resume' attribute to the <enable/> element with a value of "true" or "1" ¹².

Listing 8: Client enables stream management

```
<enable xmlns='urn:xmpp:sm:3' resume='true' />
```

If the server will allow the stream to be resumed, it MUST include a 'resume' attribute set to "true" or "1" on the <enabled/> element and MUST include an 'id' attribute that specifies an identifier for the stream.

Listing 9: Server allows stream resumption

```
<enabled xmlns='urn:xmpp:sm:3' id='some-long-sm-id' resume='true' />
```

¹¹XEP-0203: Delayed Delivery <<https://xmpp.org/extensions/xep-0203.html>>.

¹²In 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.

Definition: The 'id' attribute defines a unique identifier for purposes of stream management (an "SM-ID"). The SM-ID MUST be generated by the server. The client MUST consider the SM-ID to be opaque and therefore MUST NOT assign any semantic meaning to the SM-ID. The server MAY encode any information it deems useful into the SM-ID, such as the full JID <localpart@domain.tld/resource> of a connected client (e.g., the full JID plus a nonce value). Any characters allowed in an XML attribute are allowed. The SM-ID MUST NOT be reused for simultaneous or subsequent sessions (but the server need not ensure that SM-IDs are unique for all time, only for as long as the server is continuously running). The SM-ID SHOULD NOT be longer than 4000 bytes.

As noted, the <enabled/> element MAY include a 'location' attribute that specifies the server's preferred location for reconnecting (e.g., a particular connection manager that hold session state for the connected client).

Listing 10: Server prefers reconnection at a particular location

```
<enabled xmlns='urn:xmpp:sm:3'  
  id='some-long-sm-id'  
  location='[2001:41D0:1:A49b::1]:9222'  
  resume='true' />
```

If the stream is terminated unexpectedly, the client would then open a TCP connection to the server. The order of events is as follows:

1. After disconnection, the client opens a new TCP connection to the server, preferring the address specified in the 'location' attribute (if any).
2. Client sends initial stream header.
3. Server sends response stream header.
4. Server sends stream features.
5. Client sends STARTTLS request.
6. Server informs client to proceed with the TLS negotiation.
7. The parties complete a TLS handshake. (Note: When performing session resumption and also utilizing TLS, it is RECOMMENDED to take advantage of TLS session resumption RFC 5077¹³ to further optimize the resumption of the XML stream.)
8. Client sends new initial stream header.
9. Server sends response stream header.

¹³RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State <<http://tools.ietf.org/html/rfc5077>>.

10. Server sends stream features, requiring SASL negotiation and offering appropriate SASL mechanisms. (Note: If the server considers the information provided during TLS session resumption to be sufficient authentication, it MAY offer the SASL EXTERNAL mechanism; for details, refer to [draft-cridland-sasl-tls-sessions](#) ¹⁴.)
11. The parties complete SASL negotiation.
12. Client sends new initial stream header.
13. Server sends response stream header.
14. Server sends stream features, offering the SM feature.
15. Client requests resumption of the former stream.

Note: The order of events might differ from those shown above, depending on when the server offers the SM feature, whether the client chooses STARTTLS, etc. Furthermore, in practice server-to-server streams often do not complete SASL negotiation or even TLS negotiation. The foregoing text does not modify any rules about the stream negotiation process specified in RFC 6120. However, since stream management applies to the exchange of stanzas (not any other XML elements), it makes sense for the server to offer the SM feature when it will be possible for the other party to start sending stanzas, not before. See also [Recommended Order of Stream Feature Negotiation \(XEP-0170\)](#) ¹⁵.

To request resumption of the former stream, the client sends a <resume/> element qualified by the 'urn:xmpp:sm:3' namespace. The <resume/> element MUST include a 'previd' attribute whose value is the SM-ID of the former stream and MUST include an 'h' attribute that identifies the sequence number of the last handled stanza sent over the former stream from the server to the client (in the unlikely case that the client never received any stanzas, it would set 'h' to zero).

Listing 11: Stream resumption request

```
<resume xmlns='urn:xmpp:sm:3'  
  h='some-sequence-number'  
  previd='some-long-sm-id' />
```

If the server can resume the former stream, it MUST return a <resumed/> element, which MUST include a 'previd' attribute set to the SM-ID of the former stream and MUST also include an 'h' attribute set to the sequence number of the last handled stanza sent over the former stream from the client to the server (in the unlikely case that the server never received any stanzas, it would set 'h' to zero).

¹⁴On the use of TLS Session resumption and SASL EXTERNAL <<http://tools.ietf.org/html/draft-cridland-sasl-tls-sessions>>. Work in progress.

¹⁵XEP-0170: Recommended Order of Stream Feature Negotiation <<https://xmpp.org/extensions/xep-0170.html>>.

Listing 12: Stream resumed

```
<resumed xmlns='urn:xmpp:sm:3'
         h='another-sequence-number'
         previd='some-long-sm-id' />
```

If the server does not support session resumption, it MUST return a <failed/> element, which SHOULD include an error condition of <feature-not-implemented/>. If the server does not recognize the 'previd' as an earlier session (e.g., because the former session has timed out), it MUST return a <failed/> element, which SHOULD include an error condition of <item-not-found/>. If the server recognizes the 'previd' as an earlier session that has timed out the server MAY also include a 'h' attribute indicating the number of stanzas received before the timeout. (Note: For this to work the server has to store the SM-ID/sequence number tuple past the time out of the actual session.)

Listing 13: Stream timed out

```
<failed xmlns='urn:xmpp:sm:3'
        h='another-sequence-number' >
  <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
</failed>
```

In both of these failure cases, the server SHOULD allow the client to bind a resource at this point rather than forcing the client to restart the stream negotiation process and re-authenticate.

If the former stream is resumed and the server still has the stream for the previously-identified session open at this time, the server SHOULD send a 'conflict' stream error and close that stream.

Listing 14: Server closes old stream that was replaced by new

```
<stream:error>
  <conflict xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>
</stream:stream>
```

When a session is resumed, the parties proceed as follows:

- The sequence values are carried over from the previous session and are not reset for the new stream.
- Upon receiving a <resume/> or <resumed/> element the client and server use the 'h' attribute to retransmit any stanzas lost by the disconnection. In effect, it should handle the element's 'h' attribute as it would handle it on an <a/> element (i.e., marking stanzas in its outgoing queue as handled), except that after processing it MUST re-send to the peer any stanzas that are still marked as unhandled.

- Both parties SHOULD retransmit any stanzas that were not handled during the previous session, based on the sequence number reported by the peer.
- A reconnecting client SHOULD NOT request the roster, because any roster changes that occurred while the client was disconnected will be sent to the client after the stream management session resumes.
- The client SHOULD NOT resend presence stanzas in an attempt to restore its former presence state, since this state will have been retained by the server.
- Both parties SHOULD NOT try to re-establish state information (e.g., [Service Discovery \(XEP-0030\)](#)¹⁶ information).

6 Error Handling

If an error occurs with regard to an `<enable/>` or `<resume/>` element, the server MUST return a `<failed/>` element. This element SHOULD contain an error condition, which MUST be one of the stanza error conditions defined in RFC 6120.

An example follows.

Listing 15: Server returns error

```
<failed xmlns='urn:xmpp:sm:3'>
  <unexpected-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
</failed>
```

Stream management errors SHOULD be considered recoverable; however, misuse of stream management MAY result in termination of the stream.

When a remote entity acknowledges that it has handled a number of stanzas that is higher than the amount of stanzas that it was sent (by sending an 'h' value that is too high), the local entity SHOULD generate an undefined-condition stream error that includes a `<handled-count-too-high/>` element, and close the stream:

Listing 16: Entity closes stream because peer acknowledges more stanzas than it was sent

```
<stream:error>
  <undefined-condition xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  <handled-count-too-high xmlns='urn:xmpp:sm:3' h='10' send-count='8' />
  <text xml:lang='en' xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
    You acknowledged 10 stanzas, but I only sent you 8 so far.
  </text>
</stream:error>
</stream:stream>
```

¹⁶XEP-0030: Service Discovery <<https://xmpp.org/extensions/xep-0030.html>>.

7 Stream Closure

A cleanly closed stream differs from an unfinished stream. If a client wishes to cleanly close its stream and end its session, it **MUST** send a `</stream:stream>` so that the server can send unavailable presence on the client's behalf.

If the stream is not cleanly closed then the server **SHOULD** consider the stream to be unfinished (even if the client closes its TCP connection to the server) and **SHOULD** maintain the session on behalf of the client for a limited amount of time. The client can send whatever presence it wishes before leaving the stream in an unfinished state.

8 Scenarios

The following scenarios illustrate several different uses of stream management. The examples are that of a client and a server, but stream management can also be used for server-to-server streams.

8.1 Basic Acking Scenario

The Stream Management protocol can be used to improve reliability using acks without the ability to resume a session. A basic implementation would do the following:

- As a client, send `<enable/>` with no attributes, and ignore the attributes on the `<enabled/>` response.
- As a server, ignore the attributes on the `<enable/>` element received, and respond via `<enabled/>` with no attributes.
- When receiving an `<r/>` element, immediately respond via an `<a/>` element where the value of 'h' returned is the sequence number of the last handled stanza.
- Keep an integer X for this stream session, initially set to zero. When about to send a stanza, first put the stanza (paired with the current value of X) in an "unacknowledged" queue. Then send the stanza over the wire with `<r/>` to request acknowledgement of that outbound stanza, and increment X by 1. When receiving an `<a/>` element with an 'h' attribute, all stanzas whose paired value (X at the time of queueing) is less than or equal to the value of 'h' can be removed from the unacknowledged queue.

This is enough of an implementation to minimally satisfy the peer, and allows basic tracking of each outbound stanza. If the stream connection is broken, the application has a queue of unacknowledged stanzas that it can choose to handle appropriately (e.g., warn a human user or silently send after reconnecting).

The following examples illustrate basic acking (here the client automatically acks each stanza

it has received from the server, without first being prompted via an `<r/>` element). First, after authentication and resource binding, the client enables stream management.

Listing 17: Client enables stream management

```
<enable xmlns='urn:xmpp:sm:3' />
```

The server then enables stream management.

Listing 18: Server enables stream management

```
<enabled xmlns='urn:xmpp:sm:3' />
```

The client then retrieves its roster and immediately sends an `<r/>` element to request acknowledgement.

Listing 19: Client sends a stanza and requests acknowledgement

```
<iq id='ls72g593' type='get'>
  <query xmlns='jabber:iq:roster' />
</iq>

<r xmlns='urn:xmpp:sm:3' />
```

The server handles the client stanza (here returning the roster) and sends an `<a/>` element to acknowledge handling of the stanza.

Listing 20: Server handles client stanza and acknowledges handling of client stanza

```
<iq id='ls72g593' type='result'>
  <query xmlns='jabber:iq:roster'>
    <item jid='juliet@capulet.lit' />
    <item jid='benvolio@montague.lit' />
  </query>
</iq>

<a xmlns='urn:xmpp:sm:3' h='1' />
```

The client then chooses to acknowledge receipt of the server's stanza (although here it is under no obligation to do so, since the server has not requested an ack), sends initial presence, and immediately sends an `<r/>` element to request acknowledgement, incrementing by one its internal representation of how many stanzas have been handled by the server.

Listing 21: Client acks handling of first server stanza, sends a stanza, and requests acknowledgement

```
<a xmlns='urn:xmpp:sm:3' h='1' />
```

```
<presence/>
<r xmlns='urn:xmpp:sm:3' />
```

The server immediately sends an `<a/>` element to acknowledge handling of the stanza and then broadcasts the user's presence (including to the client itself as shown below).

Listing 22: Server acks handling of second client stanza and sends a stanza

```
<a xmlns='urn:xmpp:sm:3' h='2' />
<presence from='romeo@montague.lit/orchard'
  to='romeo@montague.lit/orchard' />
```

The client then acks the server's second stanza and sends an outbound message followed by an `<r/>` element.

Listing 23: Client acks receipt of second server stanza, sends a stanza, and requests acknowledgement

```
<a xmlns='urn:xmpp:sm:3' h='2' />
<message to='juliet@capulet.lit'>
  <body>ciao!</body>
</message>
<r xmlns='urn:xmpp:sm:3' />
```

The server immediately sends an `<a/>` element to acknowledge handling of the third client stanza and then routes the stanza to the remote contact (not shown here because the server does not send a stanza to the client).

Listing 24: Server acknowledges handling of third client stanza

```
<a xmlns='urn:xmpp:sm:3' h='3' />
```

And so on.

8.2 Efficient Acking Scenario

The basic acking scenario is wasteful because the client requested an ack for each stanza. A more efficient approach is to periodically request acks (e.g., every 5 stanzas). This is shown schematically in the following pseudo-XML.

Listing 25: An efficient session


```
<!--{}- Client -{}-->
<enable/>
<!--{}- Server -{}-->
<enabled/>
<!--{}- Client -{}-->
<message/>
<message/>
<message/>
<message/>
<message/>
<r/>
<!--{}- Server -{}-->
<a h='5' />
<!--{}- Client -{}-->
<message/>
<message/>
<message/>
<message/>
<message/>
<r/>
<!--{}- Server -{}-->
<a h='10' />
```

In particular, on mobile networks, it is advisable to only request and/or send acknowledgements when an entity has other data to send, or in lieu of a whitespace keepalive or XMPP ping (XEP-0199).

9 SASL2 And BIND2 Interaction

This section is about [Extensible SASL Profile \(XEP-0388\)](https://xmpp.org/extensions/xep-0388.html)¹⁷ (SASL2) and [Bind 2.0 \(XEP-0386\)](https://xmpp.org/extensions/xep-0386.html)¹⁸ (BIND2) interaction. You don't have to implement this if you don't implement SASL2 and BIND2.

[Extensible SASL Profile \(XEP-0388\)](https://xmpp.org/extensions/xep-0388.html)¹⁹ (SASL2) describes a way of inlining the stream resumption into the authentication process to reduce the round-trips needed for authentication and subsequent stream resumption. Similarly [Bind 2.0 \(XEP-0386\)](https://xmpp.org/extensions/xep-0386.html)²⁰ (BIND2) defines a way of inlining the stream management `<enable/>` into the resource binding process to reduce round-trips.

¹⁷XEP-0388: Extensible SASL Profile <<https://xmpp.org/extensions/xep-0388.html>>.

¹⁸XEP-0386: Bind 2.0 <<https://xmpp.org/extensions/xep-0386.html>>.

¹⁹XEP-0388: Extensible SASL Profile <<https://xmpp.org/extensions/xep-0388.html>>.

²⁰XEP-0386: Bind 2.0 <<https://xmpp.org/extensions/xep-0386.html>>.

9.1 Enable Stream Management Inline

To indicate support for inlining the activation of Stream Management into the resource binding process, the server adds a <feature/> element with var attribute set to "urn:xmpp:sm:3" in the <inline/> element of BIND2 which is sent in the stream features.

If the client wishes to begin a new session (i.e. it has no prior session to resume), it simply includes the <enable/> element defined by this specification in its [Bind 2.0 \(XEP-0386\)](#)²¹ <bind/> request which itself is a child of the <authenticate/> element of SASL2.

9.1.1 Failed To Enable Stream Management

In the unexpected case where the server was able to bind a resource for the client, but unable to enable stream management, it will include a <failed/> element as defined by this specification within the <bound/> response defined by [Bind 2.0 \(XEP-0386\)](#)²².

9.2 Inline Stream Resumption

To indicate support for inlining stream resumption into the authentication process, the server adds a <sm/> element in the namespace "urn:xmpp:sm:3" to the <inline/> element of SASL2.

If the client wishes to resume an existing session it, it simply includes the <resume/> element defined by this specification in the SASL2 <authenticate/> element.

Note: If the client included a <resume/> element in its SASL2 <authenticate/> element, that MUST be processed first by the server. If that resumption is successful, the server MUST skip resource binding (a resumed session already has a resource bound) and MUST entirely ignore the <bind/> request that might also be inlined in the <authenticate/> element.

[Extensible SASL Profile \(XEP-0388\)](#)²³ mandates that the <success> element is immediately followed by stream features. If a former stream has been successfully resumed using this specification, the stream is considered re-established immediately after the <success/> element instead and stream features MUST NOT be sent in this case.

9.2.1 Failed Resumption

Sometimes resumption might fail - for example, because the session has been disconnected longer than the server's resumption timeout. In this case, the server MUST include the <failed/> element defined by this specification in its SASL2 <success/> response, but also MUST continue to process the <bind/> in order to establish a new session for the client.

The client can find details about its new session in the <bound/> response (defined by [Bind 2.0 \(XEP-0386\)](#)²⁴).

²¹XEP-0386: Bind 2.0 <<https://xmpp.org/extensions/xep-0386.html>>.

²²XEP-0386: Bind 2.0 <<https://xmpp.org/extensions/xep-0386.html>>.

²³XEP-0388: Extensible SASL Profile <<https://xmpp.org/extensions/xep-0388.html>>.

²⁴XEP-0386: Bind 2.0 <<https://xmpp.org/extensions/xep-0386.html>>.

9.3 Inlining Examples

Listing 26: Server announces support for inlining activation of stream management and stream resumption

```
<stream:features>
  <authentication xmlns="urn:xmpp:sasl:2">
    <mechanism>SCRAM-SHA-1</mechanism>
    <inline>
      <sm xmlns="urn:xmpp:sm:3"/>
      <bind xmlns="urn:xmpp:bind2:1">
        <feature var='urn:xmpp:sm:3'/>
      </bind>
    </inline>
  </authentication>
</stream:features>
```

Listing 27: Client tries to resume stream and enables stream management inlined in its bind request

```
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='SCRAM-SHA-1'>
  <initial-response>[base64 encoded SASL data]</initial-response>
  <resume xmlns="urn:xmpp:sm:3" h="some-sequence-number" previd="some-
    long-sm-id"/>
  <bind xmlns="urn:xmpp:bind2:1">
    <tag>AwesomeXMPP</tag>
    <enable xmlns="urn:xmpp:sm:3"/>
  </bind>
</authenticate>
```

Listing 28: Client failed to enable stream management

```
<success xmlns='urn:xmpp:sasl:2'>
  <additional-data>[base64 encoded SASL data]</additional-data>
  <authorization-identifier>user@example.com/resource</authorization-
    identifier>
  <bound xmlns='urn:xmpp:bind2:1'>
    <failed xmlns='urn:xmpp:sm:3'>
      <internal-server-error xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    </failed>
    <latest-id>0312a1b8</latest-id>
  </bound>
</success>
```

Listing 29: Client failed to resume stream

```
<success xmlns='urn:xmpp:sasl:2'>
```

```

<additional-data>[base64 encoded SASL data]</additional-data>
<authorization-identifier>user@example.com/resource</authorization-
  identifier>
<failed xmlns='urn:xmpp:sm:3' h='another-sequence-number'>
  <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
</failed>
<bound xmlns='urn:xmpp:bind2:1'>
  <latest-id>0312a1b8</latest-id>
</bound>
</success>

```

Listing 30: Client failed to resume stream and to enable management of newly bound stream

```

<success xmlns='urn:xmpp:sasl:2'>
  <additional-data>[base64 encoded SASL data]</additional-data>
  <authorization-identifier>user@example.com/resource</authorization-
    identifier>
  <failed xmlns='urn:xmpp:sm:3' h='another-sequence-number'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </failed>
  <bound xmlns='urn:xmpp:bind2:1'>
    <failed xmlns='urn:xmpp:sm:3'>
      <internal-server-error xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas' />
    </failed>
    <latest-id>0312a1b8</latest-id>
  </bound>
</success>

```

10 Security Considerations

As noted, a server MUST NOT allow a client to resume a stream management session until after the client has authenticated (for some value of "authentication"); this helps to prevent session hijacking.

If SASL2 is used to inline stream resumption, implementations must adhere to the security considerations defined in [Extensible SASL Profile \(XEP-0388\)](#)²⁵ regarding the inclusion of SASL2 requests and inline feature negotiation in TLS 0-RTT ("early data") extensions. That is, these elements MUST NOT be sent or processed if the stream would be resumed solely based on 0-RTT data, except when appropriate mitigations are in place (which are beyond the scope of this document, but may be defined by others).

²⁵XEP-0388: Extensible SASL Profile <<https://xmpp.org/extensions/xep-0388.html>>.

11 IANA Considerations

This XEP requires no interaction with the [Internet Assigned Numbers Authority \(IANA\)](#) ²⁶.

12 XMPP Registrar Considerations

12.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:sm:3

The [XMPP Registrar](#) ²⁷ includes the foregoing namespace in its registry at [<https://xmpp.org/registrar/namespaces.html>](https://xmpp.org/registrar/namespaces.html), as described in Section 4 of [XMPP Registrar Function \(XEP-0053\)](#) ²⁸.

12.2 Protocol Versioning

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

12.3 Stream Features

The XMPP Registrar includes 'urn:xmpp:sm:3' in its registry of stream features at [<https://xmpp.org/registrar/stream-features.html>](https://xmpp.org/registrar/stream-features.html).

13 XML Schemas

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

²⁶The 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/>.

²⁷The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

²⁸XEP-0053: XMPP Registrar Function [<https://xmpp.org/extensions/xep-0053.html>](https://xmpp.org/extensions/xep-0053.html).

```
xmlns:xs='http://www.w3.org/2001/XMLSchema'
targetNamespace='urn:xmpp:sm:3'
xmlns='urn:xmpp:sm:3'
elementFormDefault='qualified'>

<xs:annotation>
  <xs:documentation>
    The protocol documented by this schema is defined in
    XEP-0198: http://www.xmpp.org/extensions/xep-0198.html
  </xs:documentation>
</xs:annotation>

<xs:import namespace='urn:ietf:params:xml:ns:xmpp-stanzas'
           schemaLocation='http://xmpp.org/schemas/stanzaerror.xsd'/
>

<xs:element name='a'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='h'
                     type='xs:unsignedInt'
                     use='required' />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='enable'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='max'
                     type='xs:positiveInteger'
                     use='optional' />
        <xs:attribute name='resume'
                     type='xs:boolean'
                     use='optional'
                     default='false' />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='enabled'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='id'
                     type='xs:string'
                     use='optional'
                     default='' />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
        type='xs:string'
        use='optional' />
    <xs:attribute name='location'
        type='xs:string'
        use='optional' />
    <xs:attribute name='max'
        type='xs:positiveInteger'
        use='optional' />
    <xs:attribute name='resume'
        type='xs:boolean'
        use='optional'
        default='false' />
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>

<xs:element name='failed'>
    <xs:complexType>
        <xs:sequence xmlns:err='urn:ietf:params:xml:ns:xmpp-stanzas'
            minOccurs='0'
            maxOccurs='1'>
            <xs:group ref='err:stanzaErrorGroup' />
        </xs:sequence>
        <xs:attribute name='h'
            type='xs:unsignedInt'
            use='optional' />
    </xs:complexType>
</xs:element>

<xs:element name='r' type='empty' />

<xs:element name='resume' type='resumptionElementType' />

<xs:element name='resumed' type='resumptionElementType' />

<xs:element name='sm'>
    <xs:complexType>
        <xs:choice>
            <xs:element name='optional' type='empty' />
            <xs:element name='required' type='empty' />
        </xs:choice>
    </xs:complexType>
</xs:element>

<xs:complexType name='resumptionElementType'>
    <xs:simpleContent>
        <xs:extension base='empty'>
            <xs:attribute name='h'
```

```
                type='xs:unsignedInt'
                use='required' />
        <xs:attribute name='previd'
                    type='xs:string'
                    use='required' />
    </xs:extension>
</xs:simpleContent>
</xs:complexType>

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

<xs:element name='handled-count-too-high'>
  <xs:complexType>
    <xs:attribute name='send-count'
                  type='xs:unsignedInt'
                  use='optional' />
    <xs:attribute name='h'
                  type='xs:unsignedInt'
                  use='optional' />
  </xs:complexType>
</xs:element>

</xs:schema>
```

14 Acknowledgements

Thanks to Bruce Campbell, Jack Erwin, Philipp Hancke, Curtis King, Tobias Markmann, Alexey Melnikov, Pedro Melo, Robin Redeker, Mickaël Rémond, Florian Schmaus, and Tomasz Sterna for their feedback.