# XEP-0085: Chat State Notifications


Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

Dave Smith
mailto:dizzyd@jabber.org
xmpp:dizzyd@jabber.org



2009-09-23
Version 2.1

| Status | Type | Short Name |
|---|---|---|
| Final | Standards Track | chatstates |



This document defines an XMPP protocol extension for communicating the status of a user in a chat session, thus indicating whether a chat partner is actively engaged in the chat, composing a message, temporarily paused, inactive, or gone. The protocol can be used in the context of a one-to-one chat session or a multi-user chat room.

# Legal

## Copyright



## Permissions



## Warranty



## Liability



## 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

Many instant messaging systems include notifications about the state of one's conversation partner in a one-to-one chat (or, sometimes, in a many-to-many chat). Usually these are limited to notification that one's partner is currently typing -- e.g., the Composing event in the older (deprecated) Message Events (XEP-0022) [1] protocol. However, a composing event is essentially information about a person's participation in or involvement with the chat "session" and therefore is really a session-level state rather than a per-message event (in contrast to the Delivered and Displayed events in XEP-0022). While the composing event is interesting, the concept of a session-level state can be extended to answer a variety of questions about the participation of a person in a real-time chat conversation, such as:

- Has this person paused the composition?
- Is this person actively paying attention to the chat?
- Is this person temporarily inactive (i.e., not paying attention right now)?
- Is this person simply gone (i.e., no longer participating in the chat)?

To answer such questions, this document supplements the traditional composing state by defining four additional chat states (paused, active, inactive, gone), for a total of five states that (it is hoped) together fully describe the possible states of a person's participation in or involvement with a chat conversation. [2]

# 2 Definitions

In essence, chat state notifications can be thought of as a form of chat-specific presence. For example, consider what might happen if a user "loses" a chat window on his desktop; the user might still be interacting with his messaging client (thus never automatically changing his basic presence to "away"), but the user's state with regard to the chat session might change progressively from active to inactive to gone. This information would help the user's conversation partner understand why she has not received a response to her messages in the chat session.
Chat state notifications can appear in two kinds of <message/> stanzas:

[1] XEP-0022: Message Events <https://xmpp.org/extensions/xep-0022.html>.
[2] These states do not necessarily refer to the state of the client interface and certainly not to the disposition of a particular message. However, the user's involvement with the system, device, chat session interface, or input interface can provide important clues regarding the user's involvement with the chat session; these clues can be used by the client in determining when to generate chat state notifications.

- A "content message" -- that is, a message stanza whose primary meaning is contained in standard messaging content such as the XMPP <body/> or any other properly-namespaced child element(s) other than those defined for chat state notifications in this specification.
- A "standalone notification" -- that is, a message stanza that does not contain standard messaging content but instead is intended to specify only the chat state since it contains only a child element qualified by the "http://jabber.org/protocol/chatstates" namespace (or possibly also the XMPP <thread/> element; see the Threads section below).

The five chat states specified in this document are described below. The suggested triggers are simply that: *suggestions*. It is up to the implementation to determine when to generate chat state notifications and which notifications to generate.

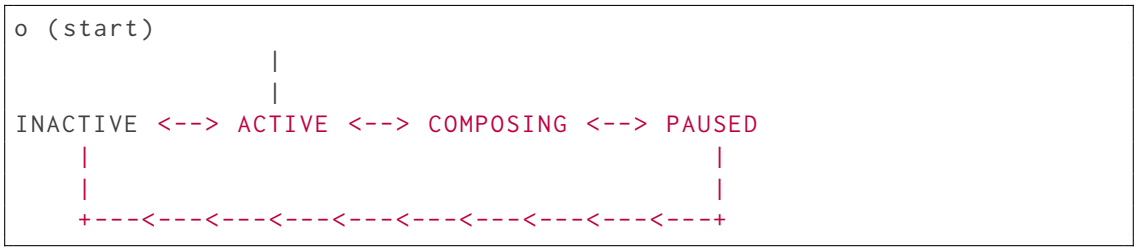
| State | Definition | Suggested Triggers |
|---|---|---|
| <active/> | User is actively participating in the chat session. | User accepts an initial content message, sends a content message, gives focus to the chat session interface (perhaps after being inactive), or is otherwise paying attention to the conversation. |
| <inactive/> | User has not been actively participating in the chat session. | User has not interacted with the chat session interface for an intermediate period of time (e.g., 2 minutes). |
| <gone/> | User has effectively ended their participation in the chat session. | User has not interacted with the chat session interface, system, or device for a relatively long period of time (e.g., 10 minutes). |
| <composing/> | User is composing a message. | User is actively interacting with a message input interface specific to this chat session (e.g., by typing in the input area of a chat window). |
| <paused/> | User had been composing but now has stopped. | User was composing but has not interacted with the message input interface for a short period of time (e.g., 30 seconds). |

Note that the <active/>, <inactive/>, and <gone/> states refer to the overall chat session interface whereas the <composing/> and <paused/> states refer to the message input interface (and are in some sense a subset of <active/>). Some implementations might support only events related to the message input interface, some implementations might support only

events related to the overall chat session interface, and some implementations might support both kinds of events.

## 3 State Chart

The following figure attempts to capture the most common state transitions in visual form (all four of the states shown can also transition to the GONE state).




Note: Other transitions are not forbidden if the developers of an implementation feel that such transitions are desirable (e.g., INACTIVE to PAUSED if a user returns to a chat session interface containing an unfinished message).

## 4 Determining Support

If an entity supports the Chat State Notifications protocol, it MUST advertise that fact in its responses to Service Discovery (XEP-0030) [3] information ("disco#info") requests by returning a feature of "http://jabber.org/protocol/chatstates":

Listing 1: A disco#info query

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

Listing 2: A disco#info response

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

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

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

In order for an application to determine whether an entity supports this protocol, where possible it SHOULD use the dynamic, presence-based profile of service discovery defined in Entity Capabilities (XEP-0115) [4]. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.

# 5 Business Rules

## 5.1 Generation of Notifications

Before generating chat state notifications, a User SHOULD explicitly discover whether the Contact supports the protocol defined herein (as described in the Discovering Support section of this document) or explicitly negotiate the use of chat state notifications with the Contact (e.g., via Stanza Session Negotiation (XEP-0155) [5]).
In the absence of explicit discovery or negotiation, the User MAY implicitly request and discover the use of chat state notifications in a one-to-one chat session by adhering to the following business rules:

1. If the User desires chat state notifications, the message(s) that it sends to the Contact before receiving a reply MUST contain a chat state notification extension, which SHOULD be <active/>.
2. If the Contact replies but does not include a chat state notification extension, the User MUST NOT send subsequent chat state notifications to the Contact.
3. If the Contact replies and includes an <active/> notification (or sends a standalone notification to the User), the User and Contact SHOULD send subsequent notifications for supported chat states (as specified in the next subsection) by including an <active/> notification in each content message and sending standalone notifications for the chat states they support (at a minimum, the <composing/> state).

The foregoing rules imply that the sending of chat state notifications is bidirectional (i.e., both User and Contact will either send or not send chat state notifications) rather than unidirectional (i.e., one of the conversation partners will send chat state notifications but the other will not); this is by design.

## 5.2 Support Requirements

[4] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.
[5] XEP-0155: Stanza Session Negotiation <https://xmpp.org/extensions/xep-0155.html>.

| Chat State | Requirement |
|---|---|
| <active/> | MUST |
| <composing/> | MUST |
| <paused/> | SHOULD |
| <inactive/> | SHOULD |
| <gone/> | SHOULD |

A client MUST allow users to configure whether they want to send chat state notifications. Note: Support for only <active/> and <composing/> is functionally equivalent to supporting the Composing event from XEP-0022.

### 5.3 Repetition

Even if the user types continuously for a long time (e.g., while composing a lengthy reply), the client MUST NOT send more than one standalone <composing/> notification in a row. More generally, a client MUST NOT send a second instance of any given standalone notification (i.e., a standalone notification MUST be followed by a different state, not repetition of the same state). However, every content message SHOULD contain an <active/> notification.

### 5.4 Context of Usage

1. This protocol MUST NOT be used with stanzas other than <message/>.
2. This protocol SHOULD NOT be used with message types other than "chat" or "groupchat".
3. The 'type' attribute for content messages and standalone notifications SHOULD be set to a value of "chat" (for one-to-one sessions) or "groupchat" (for many-to-many sessions).
4. A chat session MAY span multiple user sessions (i.e., chat state is orthogonal to the presence of one's conversation partner), although this is unlikely given the suggested timing of event triggers.

### 5.5 Use in Groupchat

Chat state notifications MAY be sent in the context of groupchat rooms (e.g., as defined in Multi-User Chat (XEP-0045) [6]). The following business rules apply:

1. A client MAY send chat state notifications even if not all room occupants do so.

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

2. A client SHOULD NOT generate <gone/> notifications.

3. A client SHOULD ignore <gone/> notifications received from other room occupants.

Note: Use of chat state notifications in the context of groupchat can result in multicasting of such notifications. Forewarned is forearmed.

## 5.6 Syntax of Notifications

1. A message stanza MUST NOT contain more than one child element qualified by the 'http://jabber.org/protocol/chatstates' namespace.

2. A message stanza that contains standard instant messaging content SHOULD NOT contain a chat state notification extension other than <active/>, where "standard instant messaging content" is taken to mean the <body/>, <subject/>, and <thread/> child elements defined in XMPP IM [7] or any other child element that would lead the recipient to treat the stanza as an instant message as explained in Message Stanza Profiles (XEP-0226) [8].

3. A message stanza that does not contain standard messaging content and is intended to specify only the chat state MUST NOT contain any child elements other than the chat state notification extension, which SHOULD be a state other than <active/>; however, if threads are used (see below) then the standalone notification MUST also contain the <thread/> element.

## 5.7 Threads

While chat state notifications provide a mechanism for managing chat threads as communicated by inclusion of the XMPP <thread/> element, support for threads is OPTIONAL (for further information about threads, refer to Best Practices for Message Threads (XEP-0201) [9]). However, if all of the clients participating in a chat both support and use threads, the following additional business rules apply:

1. Clients MUST copy back Thread IDs (i.e., the value of the <thread/> element) in any replies.

2. When a client terminates a one-to-one chat session (e.g., when a user closes the chat session interface), it MUST generate a <gone/> event.

3. Upon receiving a <gone/> event, a client MUST NOT re-use the same Thread ID and MUST generate a new Thread ID for any subsequent chat messages sent to the conversation partner.

[7] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[8] XEP-0226: Message Stanza Profiles <https://xmpp.org/extensions/xep-0226.html>.
[9] XEP-0201: Best Practices for Message Threads <https://xmpp.org/extensions/xep-0201.html>.

### 5.8 Server Handling of Notifications

Servers in constrained network environments (e.g., serving small-footprint clients via Jabber HTTP Polling (XEP-0025) [10] or BOSH (XEP-0124) [11]) and services that rebroadcast message stanzas (e.g., Multi-User Chat services) MAY process standalone notifications differently from other messages. In particular, a server or service MAY refuse to deliver standalone notifications to its users, and SHOULD NOT store them offline. In contrast to XEP-0022, chat state notifications are completely the responsibility of the client, and MUST NOT be generated by a server or service.

## 6 A Simple Example

In the following conversation, both User <bernardo@shakespeare.lit> and Contact <francisco@shakespeare.lit> support chat state notifications.

Listing 3: User Sends Initial Content Message With <active/> Notification

```
<message
    from='bernardo@shakespeare.lit/pda'
    to='francisco@shakespeare.lit'
    type='chat'>
  <body>Who's there?</body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Listing 4: Contact's Client Sends Content Message Reply With <active/> Notification

```
<message
    from='francisco@shakespeare.lit/elsinore'
    to='bernardo@shakespeare.lit/pda'
    type='chat'>
  <body>Nay, answer me: stand, and unfold yourself.</body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Because the User now knows that the Contact supports chat state notifications, the User can send other notification types.

Listing 5: User Sends Standalone <composing/> Notification

```
<message
    from='bernardo@shakespeare.lit/pda'
    to='francisco@shakespeare.lit/elsinore'
```

[10] XEP-0025: Jabber HTTP Polling <https://xmpp.org/extensions/xep-0025.html>.
[11] XEP-0124: Bidirectional-streams Over Synchronous HTTP <https://xmpp.org/extensions/xep-0124.html>.

```
    type='chat'>
  <composing xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Listing 6: User Sends a Content Message Reply With <active/> Notification

```
<message
    from='bernardo@shakespeare.lit/pda'
    to='francisco@shakespeare.lit/elsinore'
    type='chat'>
  <body>Long live the king!</body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

And so forth.

## 7 A Detailed Conversation

The following conversation flow illustrates in more detail the workings of chat state notifications (in this case also using threads) between a User <romeo@shakespeare.lit> and a Contact <juliet@capulet.com>.

Listing 7: User Sends Initial Content Message

```
<message
    from='romeo@shakespeare.lit/orchard'
    to='juliet@capulet.com'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <body>
    I take thee at thy word:
    Call me but love, and I'll␣be␣new␣baptized;
␣␣␣␣Henceforth␣I␣never␣will␣be␣Romeo.
␣␣</body>
␣␣<active␣xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

At this point Juliet's client knows that Romeo's client supports chat state notifications. Thus she replies to the content message and her client includes a notification that her state is <active/>:

Listing 8: Contact's Client Sends Content Message Reply With <active/> Notification

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
```

```
    type='chat'>
  <thread>act2scene2chat1</thread>
  <body>
    What man art thou that thus bescreen'd in night
    So stumblest on my counsel?
  </body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

And so the conversation continues. After a while, Juliet asks a question that brings Romeo up short:

Listing 9: Contact Sends Another Message

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <body>Art thou not Romeo, and a Montague?</body>
</message>
```

Romeo begins composing a reply to Juliet's heartfelt question, and his client notifies Juliet that he is composing a reply.

Listing 10: User's Client Sends Standalone <composing/> Notification

```
<message
    from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <composing xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Romeo realizes his reply is too rash and pauses to choose the right words; after some (configurable) time period, his client senses the delay and sends a state of <paused/>.

Listing 11: User's Client Sends Standalone <paused/> Notification

```
<message
    from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <paused xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Romeo starts composing again, and his Jabber client sends a <composing/> notification to Juliet's client.

Listing 12: User's Clients Sends Standalone <composing/> Notification

```
<message
    from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <composing xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Romeo finally sends his reply.

Listing 13: User Replies

```
<message
    from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <body>Neither, fair saint, if either thee dislike.</body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

The conversation ebbs and flows, waxes and wanes, until Juliet is called away by her Nurse...

Listing 14: Contact's Client Sends Content Message

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <body>
    I hear some noise within; dear love, adieu!
    Anon, good nurse! Sweet Montague, be true.
    Stay but a little, I will come again.
  </body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

We suppose that Juliet minimizes the chat window, so her client generates an <inactive/> notification:

Listing 15: Contact's Client Sends Standalone <inactive/> Notification

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <inactive xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

When she returns and brings the window up again, her client generates an <active/> notification:

Listing 16: Contact's Client Sends Standalone <active/> Notification

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

The conversation continues, but Juliet is called away again by that nagging Nurse:

Listing 17: Contact's Client Sends Content Message

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <body>
    A thousand times good night!
  </body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

We suppose that Juliet closes the chat window, so her client generates a <gone/> notification:

Listing 18: Contact's Client Sends Standalone <gone/> Notification

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat1</thread>
  <gone xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

Romeo's client now considers the chat thread to be over and generates a new Thread ID when he sends a new message:

Listing 19: User's Client Sends Content Message with New Thread ID

```
<message
    from='romeo@shakespeare.lit/orchard'
    to='juliet@capulet.com/balcony'
    type='chat'>
  <thread>act2scene2chat2</thread>
  <body>
    A thousand times the worse, to want thy light.
    Love goes toward love, as schoolboys from their books,
    But love from love, toward school with heavy looks.
  </body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

When Juliet returns to her computer on the balcony, she finds the new message from Romeo. When she finishes her reply, her client includes both an <active/> notification and the new Thread ID with the body of her reply:

Listing 20: Contact's Client Sends Content Message

```
<message
    from='juliet@capulet.com/balcony'
    to='romeo@shakespeare.lit/orchard'
    type='chat'>
  <thread>act2scene2chat2</thread>
  <body>
    Hist! Romeo, hist! O, for a falconer's voice,....
  </body>
  <active xmlns='http://jabber.org/protocol/chatstates'/>
</message>
```

And so forth.
My, these star-crossed lovers do go on, don't they?

## 8 Implementation Notes

A client that receives a chat state notification might never receive another message or chat state notification from the other entity (e.g., because the other entity crashes or goes offline) and needs to plan accordingly.

## 9 Security Considerations

The states of a chat thread can reveal information about a user's interaction with his or her computer, including his or her physical presence; such information SHOULD NOT be revealed to conversation partners who are not trusted to know such information. Client implementations MUST provide a mechanism that enables the user to disable chat state notifications if desired.

## 10 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [12].

## 11 XMPP Registrar Considerations

### 11.1 Protocol Namespaces

The XMPP Registrar [13] includes 'http://jabber.org/protocol/chatstates' in its registry of protocol namespaces.

## 12 XML Schema

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0085: http://www.xmpp.org/extensions/xep-0085.html
    </xs:documentation>
```

[12] 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/>.

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

```
  </xs:annotation>

  <xs:element name='active' type='empty'/>
  <xs:element name='composing' type='empty'/>
  <xs:element name='gone' type='empty'/>
  <xs:element name='inactive' type='empty'/>
  <xs:element name='paused' type='empty'/>

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

</xs:schema>
```