XEP-0301: In-Band Real Time Text

Mark Rejhon
mailto:mark@realjabber.org
xmpp:markybox@gmail.com
http://www.realjabber.org

Gunnar Hellstrom
mailto:gunnar.hellstrom@omnitor.se
http://www.omnitor.se

2013-10-08
Version 1.0

Status	Type	Short Name
Draft	Standards Track	rtt

This is a specification for real-time text transmitted in-band over an XMPP session. Real-time text is text transmitted instantly while it is being typed or created.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

This document defines a specification for real-time text transmitted in-band over an XMPP network.
Real-time text is text transmitted instantly while it is being typed or created. The recipient can immediately read the sender's text as it is written, without waiting. It allows text to be used as conversationally as a telephone conversation, including in situations where speech is not practical (e.g., environments that must be quiet, environments too noisy to hear, restrictions on phone use, situations where speaking is a privacy or security concern, and/or when participant(s) are deaf or hard of hearing). It is also used for transmission of live speech transcription.
Real-time text is found in various implementations:

- The 'talk' command on UNIX systems since the 1970's.
- Session Initiation Protocol (SIP), utilizing RFC 4103 [1] real-time text.
- Instant messaging enhancements, including a **Gallaudet University** [2] collaboration.
- Next generation emergency services (RFC 6443 [3]).

For a visual animation of real-time text, see **Real-Time Text Taskforce** [4].

2 Requirements

2.1 Fluid Real-Time Text

1. Allow reliable transmission of real-time text with a low latency.
2. Support message editing in real-time, including text insertions and deletions.
3. Support transmission and reproduction of the original intervals between key presses, to preserve look-and-feel of typing independently of transmission intervals.

2.2 In-Band Transmission

1. Be backwards compatible with XMPP clients that do not support real-time text.
2. Be compatible with Multi-User Chat (XEP-0045) [5] and simultaneous logins.

[1] RFC 4103: RTP Payload for Text Conversation <http://tools.ietf.org/html/rfc4103>.
[2] Gallaudet University Technology Access Program collaboration project: Real-Time Text <http://tap.gallaudet.edu/rtt/>.
[3] RFC 6443: Framework for Emergency Calling Using Internet Multimedia <http://tools.ietf.org/html/rfc6443>.
[4] Real-Time Text Taskforce, a foundation for real-time text standardization <http://www.realtimetext.org>.
[5] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

3. Minimize reliance on out-of-band transmission protocols, for simpler network traversal.

2.3 Flexible and Interoperable

1. Allow seamless integration of real-time text into instant messaging clients, with minimal user interface modifications.
2. Be able to function securely over intermittent and unreliable connections, including mobile phones.
3. Allow use within gateways to interoperate with other real-time text protocols, including **RFC 4103** and **ITU-T T.140** [6].
4. Be usable in an international setting.

2.4 Accessible

1. Allow XMPP applications to be able to implement **ITU-T Rec. F.703** [7] Total Conversation standard for simultaneous voice, video, and real-time text.
2. Be a candidate technology for use with next generation emergency services (e.g., 9-1-1 and 1-1-2).
3. Be suitable for transcription services and (when coupled with voice at user's choice) for TTY/text telephone alternatives, relay services, and captioned telephone systems.

3 Glossary

action element An XML element that represents a single real-time message edit, such as text insertion or deletion.

character A single Unicode code point. See Unicode Character Counting.

real-time A conversational latency of less than 1 second, as defined by ITU-T Rec. F.700 ITU-T Rec. F.700: Framework Recommendation for multimedia services <http://www.itu.int/rec/T-REC-F.700>., section 2.1.2.1.

real-time text Text transmitted instantly while it is being typed or created, to allow recipient(s) to immediately read the sender's text as it is written, without waiting.

real-time message Recipient's real-time view of the sender's message still being typed or created.

RTT Acronym for real-time text.

simultaneous login Multiple simultaneous sessions, on multiple clients, using the same login (Jabber Identifier).

[6] ITU-T T.140: Protocol for multimedia application text conversation <http://www.itu.int/rec/T-REC-T.140>.
[7] ITU-T Rec. F.703: Multimedia conversational services <http://www.itu.int/rec/T-REC-F.703>.

4 Protocol

4.1 RTT Element

Real-time text is transmitted via an <rtt/> child element of a <message/> stanza. The <rtt/> element is transmitted at regular intervals by the sender client while a message is being composed. This allows the recipient to see the latest message text from the sender, without waiting for the full message to be sent in a <body/> element.
This is a basic example of a **real-time message** "Hello, my Juliet!" transmitted in real-time while it is being typed, before a final message delivery in a <body/> element (to remain Backwards Compatible):
Example 1: Introductory Example

```
<message to='juliet@capulet.lit' from='romeo@montague.lit/orchard'
    type='chat' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='0' event='new'>
    <t>Hello, </t>
  </rtt>
</message>

<message to='juliet@capulet.lit' from='romeo@montague.lit/orchard'
    type='chat' id='a02'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='1'>
    <t>my J</t>
  </rtt>
</message>

<message to='juliet@capulet.lit' from='romeo@montague.lit/orchard'
    type='chat' id='a03'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='2'>
    <t>uliet!</t>
  </rtt>
</message>

<message to='juliet@capulet.lit' from='romeo@montague.lit/orchard'
    type='chat' id='a04'>
  <body>Hello, my Juliet!</body>
</message>
```

The <rtt/> element contains one or more child elements that represent Real-Time Text Actions such as text being appended, inserted, or deleted. Example 1 illustrates only the <t/> **action element**, which appends text to the end of a message.
Transmission of the <rtt/> element occurs at a regular Transmission Interval whenever the sender is actively composing a message. If there are no changes to the message since the last transmission, no transmission occurs.
There MUST NOT be more than one <rtt/> element per <message/> stanza.
The namespace of the <rtt/> element is "urn:xmpp:rtt:0".

4.2 RTT Attributes

4.2.1 seq

This REQUIRED attribute is a counter to maintain synchronization of real-time text. Senders MUST increment this value by 1 for each subsequent edit to the same real-time message, including when appending new text. Receiving clients MUST monitor this 'seq' value as a lightweight verification on the synchronization of real-time text messages. The bounds of 'seq' is 31-bits, the range of positive values for a signed 32-bit integer. See Keeping Real-Time Text Synchronized.

4.2.2 event

This attribute signals events for real-time text.

event	Description	Action Elements	Sender Support	Recipient Support
new	Begin a new real-time message.	Yes	REQUIRED	REQUIRED
reset	Re-initialize the real-time message.	Yes	RECOMMENDED	REQUIRED
edit	Modify existing real-time message.	Yes	OPTIONAL	REQUIRED
init	Signals activation of real-time text.	No	OPTIONAL	RECOMMENDED
cancel	Signals deactivation of real-time text.	No	OPTIONAL	RECOMMENDED

If the 'event' attribute is omitted, event="edit" is assumed as the default. When Action Elements are used (e.g., text appends, insertions and deletions), the <rtt/> element MAY contain one or more of any action elements, in any order. When action elements are not allowed, the <rtt/> element MUST be empty. Recipient clients MUST ignore <rtt/> elements containing unrecognized 'event' values.

4.2.3 id

This attribute is used only if Last Message Correction (XEP-0308) [8] is implemented along with this specification. See Usage with Last Message Correction to enable real-time text during editing of the previous message.

4.3 Processing Rules

- **Initialize a new real-time message: <rtt event="new"/> and <rtt event="reset"/>**
  Sender clients MUST use an <rtt/> element containing either event="new" or event="reset" in the first transmission of a new real-time message. Recipient clients MUST initialize a new blank real-time message for display, and then process all Action Elements (e.g., text insertions and deletions) included within the <rtt/> element. If a real-time message already exists from the same sender in the same chat session, its content MUST be seamlessly replaced (i.e., cleared prior to immediately processing action elements).

- **Both <rtt event="new"/> and <rtt event="reset"/> are logically identical to recipients, except for presentation:**
  For recipients, these differ only for optional presentation purposes (e.g., highlighting newly started incoming messages). Senders SHOULD use event="new" when sending the first text of a new message (e.g., the first key presses), and only use event="reset" when doing Message Refresh or Simple Real-Time Text. See Keeping Real-Time Text Synchronized.

- **Sending modifications of a real-time message: Outgoing <rtt event="edit"/> or <rtt/>**
  Sender clients SHOULD transmit this element at a regular Transmission Interval while the message is being modified. The 'seq' attribute MUST increment by 1 for every consecutive modification transmitted. See Sending Real-Time Text.

- **Receiving modifications of a real-time message: Incoming <rtt event="edit"/> or <rtt/>**
  Recipient clients must verify that the 'seq' attribute increments by 1 in consecutively received <rtt/> elements from the same sender. If 'seq' increments as expected, the Action Elements (e.g., text insertions and deletions) included with this element MUST be processed to modify the existing real-time message. Otherwise, if 'seq' does not increment as expected, or if no real-time message already exists, the real-time message is considered out of sync and all subsequent modifications MUST be ignored until a new real-time message is initialized via event="new" or event="reset". See Keeping

[8] XEP-0308: Last Message Correction <https://xmpp.org/extensions/xep-0308.html>.

Real-Time Text Synchronized.

- **Committing a real-time message: Delivery of a <body/> element**
  A real-time message is considered complete upon receiving <body/>. See Body Element.

- **Starting real-time text: <rtt event="init"/>**
  Clients MAY use this value to signal activation of real-time text without first starting a real-time message, since the sender may not start composing immediately. The 'seq' attribute is ignored by recipient clients. See Guidelines for Initiating Real-Time Text.

- **Ending real-time text: <rtt event="cancel"/>**
  Clients MAY use this value to signal deactivation of real-time text. Clients receiving this element SHOULD also discontinue sending <rtt/> elements for the remainder of the same one-to-one chat session (until event="init" is used again), and handle any unfinished real-time messages appropriately (e.g., clearing or saving the message). The 'seq' attribute is ignored by recipient clients. See Guidelines for Initiating Real-Time Text.

- **Starting value for seq attribute:**
  Sender clients MAY use any new starting value for 'seq' when initializing a real-time message using event="new" or event="reset". Recipient clients receiving such elements MUST use this 'seq' value as the new starting value. A random starting value is RECOMMENDED to improve reliability of Keeping Real-Time Text Synchronized during Usage with Multi-User Chat and Simultaneous Logins.

4.4 Body Element

The real-time message is considered complete upon receipt of a standard <body/> element (as qualified by the 'jabber:client' namespace in XMPP IM [9]). The delivered text within <body/> is considered the final message text, and supersedes the real-time message. In the ideal case, the text within <body/> is redundant since it is identical to the final contents of the real-time message.
Sender clients MAY transmit the <body/> element in the same or separate <message/> stanza as the one containing the final <rtt/> element for the real-time message. To continue sending real-time text in subsequent <message/> stanzas, the sender client MUST first initialize a new real-time message according to Processing Rules.

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

4.4.1 Backwards Compatible

This real-time text standard simply provides early delivery of text before the <body/> element. The <body/> element continues to follow the XMPP IM specification. In particular, XMPP implementations need to ignore XML elements they do not understand. Clients, that do not support real-time text, will continue to behave normally, displaying complete lines of messages as they are delivered.

4.5 Transmission Interval

For the best balance between interoperability and usability, the default transmission interval of <rtt/> elements for a continuously-changing message SHOULD be approximately **700 milliseconds**. This interval makes it possible for clients to meet **ITU-T Rec. F.700** Section A.3.2.1 for good quality real-time text conversation in many network environments. If a different transmission interval needs to be used, the interval SHOULD be **between 300 and 1000 milliseconds**.
A longer interval will lead to a less optimal user experience. Conversely, a much shorter interval can lead to Congestion Considerations. To provide fluid real-time text, one or more of the following methods can be used:

- Preserving Key Press Intervals for natural typing display, independently of the transmission interval.
- Use of Time Critical and Low Latency Methods, for real-time captioning/speech transcription.
- For other options or reduced-precision options, see Low-Bandwidth and Low-Precision Text Smoothing.

4.6 Real-Time Text Actions

The <rtt/> element MAY contain one or more **action elements** representing real-time text operations, including text being appended, inserted, or deleted.
Many chat clients allow a sender to edit their message before sending (via a Send button, or pressing Enter). The seamless inclusion of real-time text functionality, in existing client software, needs to preserve the sender's existing expectation of being able to edit their messages. In a chat session with real-time text, the recipient can see the sender compose and edit their message before it is completed.

4.6.1 Action Elements

This is a short summary of action elements that operate on a real-time message.

Action	Element	Description	Sender Support	Recipient Support
Insert Text	<t p='#'>text</t>	Insert specified text at position 'p' in message.	REQUIRED	REQUIRED
Erase Text	<e p='#' n='#'/>	Remove 'n' characters before position 'p' in message.	RECOMMENDED	REQUIRED
Wait Interval	<w n='#'/>	Wait 'n' milliseconds.	RECOMMENDED	RECOMMENDED

These elements are kept compact in order to save bandwidth, since a single <rtt/> element can contain a large number of action elements (e.g., during Preserving Key Press Intervals). See List of Action Elements for details.

4.6.2 Attribute Values

- For Element <e/> – Erase Text: The 'n' attribute is a length value, in number of characters. If 'n' is omitted, the default value of 'n' MUST be "1".

- For Element <t/> – Insert Text and Element <e/> – Erase Text: The 'p' attribute is an absolute position value, as a character position index into the real-time message, where "0" represents the beginning of the message. If 'p' is omitted, the default value of 'p' MUST point to the end of the message (i.e., 'p' is set to the current length of the real-time message).

- For the purpose of this specification, the word "character" represents a single Unicode code point. See Unicode Character Counting.

- Senders MUST NOT use negative values for any attribute, nor use 'p' values bigger than the current message length. However, recipients receiving such values MUST clip negative values to "0", and clip excessively high 'p' values to the current length of the real-time message. Modifications only occur within the boundaries of the current real-time message.

4.6.3 List of Action Elements

Recipients MUST be able to process all <t/> and <e/> action elements for incoming <rtt/> transmissions, even if senders do not use all of these for outgoing <rtt/> transmissions

(e.g., Simple Real-Time Text). Support for <w/> is RECOMMENDED for both senders and recipients, in order to accommodate Preserving Key Press Intervals. Recipients MUST ignore unexpected or unsupported elements within <rtt/>, while continuing to process subsequent action elements. Compatibility is ensured via Namespace Versioning. Action elements are immediate child elements of the <rtt/> element, and are never nested. See examples in Use Cases.
Supports the transmission of text, including key presses, and text block inserts. *Note: Text can be any subset of text allowed in the <body/> element of a <message/>. If <t/> is empty or blank, no text modification takes place.*

```
<t>text</t>
```

Append specified **text** at the end of message. ('p' defaults to message length). *Note: This action element is the minimum support REQUIRED for sender clients (i.e., speech transcription, chat bots, and Simple Real-Time Text are still possible without supporting additional action elements).*

```
<t p='#'>text</t>
```

Inserts specified **text** at position 'p' in the message text.
Supports the behavior of backspace key presses. Text is removed towards beginning of the message. This element is also used for all delete operations, including the backspace key, the delete key, and text block deletes. *Note: Excess backspaces MUST be ignored by the receiving client. Thus, text is backspaced only to the beginning of the message, in situations where n is larger than p.*

```
<e/>
```

Remove 1 character from end of message. ('n' defaults to "1", and 'p' defaults to message length)

```
<e p='#'/>
```

Remove 1 character before character position 'p' in message. ('n' defaults to "1")

```
<e n='#'/>
```

Remove 'n' characters from end of message. ('p' defaults to message length)

```
<e n='#' p='#'/>
```

Remove 'n' characters before character position 'p' in message.
Allow for the transmission of intervals, between real-time text actions, to recreate the pauses between key presses. See Preserving Key Press Intervals.

```
<w n='#'/>
```

Wait 'n' milliseconds before processing the next action element. This pause MAY be approximate, and not necessarily be of millisecond precision. Sender clients SHOULD NOT send large 'n' values that exceed the average Transmission Interval. Recipient clients MAY selectively shorten or ignore the pauses ('n') in <w/> action elements to avoid lag in a chat session. Situations such as network congestion can result in a surge of <w/> elements where the total of pauses exceeds a transmission interval cycle. See Receiving Real-Time Text.

4.7 Keeping Real-Time Text Synchronized

During a chat session, real-time text needs to be identical on both the sender and recipient ends. A missing <rtt/> transmission can represent missing text or missing edits. Also, recipients can connect after the sender has already started composing a message. To address this, a Message Refresh mechanism allows recipient clients to recover the sender's real-time message that is actively in-progress. This synchronizes real-time text in many situations, including:

- After recipient client reconnections (e.g., due to wireless reception, due to user restarting client).
- After recipient client discarded Stale Messages (e.g., sender resumes composing hours later).
- Simultaneous Logins (e.g., user switching between devices/clients or between windows/tabs in a client).
- During Usage with Multi-User Chat (e.g., participants joining/leaving while other participants are composing).
- After message stanzas are lost in transit (e.g., Congestion Considerations).

Recipient clients MUST keep track of separate real-time messages on a per-contact basis, including tracking independent 'seq' attribute values. Recipient clients MAY track incoming <rtt/> elements per bare JID <localpart@domain.tld> to keep only one real-time message per contact. The remainder of this section automatically handles conflicting <rtt/> elements (e.g., typing coming concurrently from separate Simultaneous Logins, contrary to the common case of one typist per contact). Alternatively, recipient clients MAY track incoming <rtt/> elements per full JID <localpart@domain.tld/resource> and/or per <thread/>, to keep multiple separate real-time messages for the same contact. For more information about <thread/>, see Best Practices for Message Threads (XEP-0201) [10].

[10] XEP-0201: Best Practices for Message Threads <https://xmpp.org/extensions/xep-0201.html>.

4.7.1 Staying In Sync

By following Processing Rules, the recipient client creates a new real-time message when receiving <rtt event="new"/> or <rtt event="reset"/>. Thereafter, when receiving text modifications (i.e., <rtt event="edit"/> or <rtt/> without an 'event' attribute):

1. There MUST be an existing real-time message (created via <rtt event="new"/> or <rtt event="reset"/>);
2. Senders MUST increment the 'seq' attribute in steps of 1, for consecutively transmitted text modifications.
3. Recipients MUST verify that the 'seq' attribute is incrementing by 1, for consecutively received text modifications.

4.7.2 Recovery From Loss of Sync

Loss of sync occurs during receiving text modifications if the 'seq' attribute does not increment by 1 as expected, or if no real-time message exists. In this case:

- Recipients MUST keep the real-time message unchanged (if any exists); and
- Recipients MUST ignore subsequent text modifications (i.e., <rtt event="edit"/> or <rtt/> without an 'event' attribute); and
- An indication can be used to show the loss of sync (e.g., color coding, modified chat state message).

Recovery occurs when the recipient receives the following:

- A <body/> element. The Body Element supersedes the real-time message.
- An <rtt/> element with an 'event' attribute of 'new' or 'reset' (e.g., new message, or Message Refresh).

4.7.3 Message Refresh

A message refresh is the sender's partially composed text being (re)transmitted via <rtt event="reset"/>. The recipient client(s) can seamlessly redisplay the real-time message as a result. This allows real-time text to resume quickly, without waiting for senders to start a new message:

```
<rtt event='reset' seq='#' xmlns='urn:xmpp:rtt:0'>
  <t>This is a retransmission of the entire real-time message.</t>
</rtt>
```

The message refresh SHOULD be transmitted at intervals during active typing or composing. The RECOMMENDED interval is 10 seconds. This interval is frequent enough to minimize user waiting time, while being infrequent enough to not cause a significant bandwidth overhead. This interval can be varied, or be set to a longer time period, in order to reduce average bandwidth (e.g., long messages, infrequent or minor message changes). To save bandwidth, message refreshes SHOULD NOT occur continuously while the sender is idle. To allow quicker resumption of real-time text, sender clients MAY adjust the timing of the message refresh to occur right after any of the following additional events:

- When the recipient starts sending messages from a different full JID (e.g., switched clients);
- When the recipient presence changes to a more available state (e.g., <show/> value of “chat”);
- When the sender resumes composing after an extended pause (e.g., recipient may have cleared Stale Messages);
- When the conversation is unlocked (e.g., section 5.1 of XMPP IM);

If the recipient already has an existing real-time message from the sender, Processing Rules require that the real-time message MUST be seamlessly replaced. Thus, if the recipient is successfully Staying In Sync, the recipient user sees no visible effect since the text contained within <rtt event=”reset”/> is a duplicate of the existing real-time message. If the recipient client was out of sync (Recovery From Loss of Sync) or it has no real-time message, the recipient user sees the real-time message immediately “catch up”.
Note: The use of <rtt event=”reset”/> is not limited to message refresh, as it can contain any number of Action Elements in any order. Sender clients MAY combine a message refresh with additional action elements (e.g., re-transmitting a whole message in one Element <t/> – Insert Text, followed by some additional action elements, such as additional typing or backspacing, to seamlessly allow Preserving Key Press Intervals).

4.8 Accurate Processing of Action Elements

Real-time text is generated based on text normally allowed to be transmitted within the <body/> element.
Incorrectly generated Action Elements and Attribute Values can lead to inconsistencies between the sender and recipient during real-time editing. The Unicode characters of the real-time text need to be transmitted unaltered from the sender to the recipient, without unexpected modifications after sender pre-processing. This is the chain between the sender’s creation of real-time text, to the recipient’s processing of real-time text. Unaltered transmission of Unicode characters is possible with sender pre-processing, as long as the transmission from the sender to the recipient remains standards-compliant, including compliant XML processors and compliant XMPP servers.

If unexpected Unicode inconsistencies occur during real-time message editing, the recipient client will normally recover the message upon receiving a Body Element or a Message Refresh.

4.8.1 Unicode Character Counting

For this specification, a "character" represents a single Unicode code point. This is the same definition used in section 1.1 of **RFC 5198** [11]. For platform-independent interoperability of Action Elements, calculations on Attribute Values (***p*** and ***n***) MUST be based on counts of Unicode code points.
Many platforms use different internal encodings (i.e., string formats) that are different from the transmission encoding (UTF-8). These factors need to be considered:

- Multiple Unicode code points (e.g., combining marks, accents) can form a combining character sequence. This can occur in situations where there isn't a visually equivalent composite character of a single code point (e.g., when doing Unicode normalization). *Action elements operate on Unicode code points individually.*

- Unicode code points U+10000 through U+10FFFF are represented as a surrogate pair in some Unicode encodings (e.g., UTF-16). *Action elements operate on Unicode code points as a whole, not on separate components of a surrogate pair.*

- XMPP transmission uses UTF-8, which uses a variable number of bytes per Unicode code point.*Action elements operate on Unicode code points as a whole, not on separate bytes.*

Lengths and positions in Attribute Values are relative to the internal Unicode text of the real-time message, independently of the directionality of actual displayed text. As a result, any valid Unicode text direction can be used with real-time text (right-to-left, left-to-right, and bidirectional). One way for implementers to visualize this, is to simply visualize Unicode text as an array of individual code points, and treat Attribute Values as array indexes.

4.8.2 Guidelines for Senders

Sender clients MUST generate real-time text (Action Elements and Attribute Values) based on the plain text version of the sender's message with pre-processing completed. This is separate from and concurrent to any displayed presentation of the same message (e.g., formatting, emoticon graphics, XHTML-IM (XEP-0071) [12]).
Pre-processing before generating real-time text includes Unicode normalization, conversion

[11] RFC 5198: Unicode Format for Network Interchange <http://tools.ietf.org/html/rfc5198>.
[12] XEP-0071: XHTML-IM <https://xmpp.org/extensions/xep-0071.html>.

of emoticons graphics to text, removal of illegal characters, line-break conversion, and any other necessary text modifications. For Unicode normalization, sender clients SHOULD ensure the message is in **Unicode Normalization Form C** [13] ("NFC"), as recommended within section 3 of **RFC 5198**, and within many other standards such as Canonical XML 1.0.
If Unicode combining character sequences (e.g., letter with multiple accents) are used for Element <t/> – Insert Text, then complete combining character sequences SHOULD be sent. In situations where modifications are required to an existing combining character sequence (e.g., adding an additional accent), an Element <e/> – Erase Text SHOULD be used to delete the existing combining character sequence, before transmitting a complete replacement sequence via the <t/> element. (However, recipients SHOULD NOT assume this behavior from sending clients. See Guidelines for Recipients.)
For the purpose of calculating Attribute Values, any line breaks MUST be treated as a single character. Conversion of line breaks into a single LINE FEED U+000A is REQUIRED for XML processors, according to section 2.11 of **XML** [14].

4.8.3 Guidelines for Recipients

For Element <t/> – Insert Text, text MUST be obtained using compliant XML processing (including entities converted to characters). Recipient clients SHOULD ensure that the received text is in Unicode Normalization Form C ("NFC"). After this, recipient clients MUST NOT do any other modifications to resulting real-time messages. This is to allow accurate processing of subsequent Action Elements and Attribute Values (the recipient client can separately process/modify a copy of the same real-time message text, if necessary for the purpose of display presentation).
It is possible for sender clients to send Element <t/> – Insert Text with an incomplete combining character sequence (e.g., combining mark(s) without a Unicode base character). This is valid when extending an existing combining character sequence into a longer valid complete combining character sequence (e.g., adding an additional accent mark). It is also possible for senders to send Element <e/> – Erase Text to remove code points from an existing combining character sequence, into a shorter valid complete combining character sequence (e.g., removing an accent mark). In all cases, recipient clients MUST process these elements in accordance to Action Elements.

5 Determining Support

If a client supports this real-time text protocol, it MUST advertise that fact in its responses to Service Discovery (XEP-0030) [15] information requests ("disco#info") by returning a feature of '**urn:xmpp:rtt:0**'.

[13] Unicode Standard Annex #15: Unicode Normalization Forms <http://www.unicode.org/reports/tr15/>.
[14] XML: Extensible Markup Language 1.0 (Fifth Edition) <http://www.w3.org/TR/xml/>.
[15] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

Example 1. A disco#info query

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

Example 2. A disco#info response

```
<iq from='juliet@capulet.lit/balcony'
    id='disco1'
    to='romeo@montague.lit/orchard'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='urn:xmpp:rtt:0'/>
  </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) [16]. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.
See Guidelines for Initiating Real-Time Text for more information, including implicit discovery.

5.1 Support for Groupchat

Real-time text MAY also be used with **Multi-User Chat**. Before transmitting <rtt/> elements to a groupchat room, clients MUST follow section 17.1.1 of **XEP-0045** to verify that the service allows any extension or that 'urn:xmpp:rtt:0' is listed as an allowable namespace.
Services explicitly allowing this extension MUST follow section 17.1.1 of **XEP-0045** to include 'urn:xmpp:rtt:0' as an allowable namespace.
See Usage with Multi-User Chat.

6 Guidelines for Initiating Real-Time Text

Some clients can choose to send outgoing real-time text at all times by default. Other clients might choose to do user-initiated activation (e.g., via a button). These guidelines provide interoperability between clients that use different methods of initiating real-time text.

[16] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

6.1 Activating Real-Time Text

In the simplest case, sender clients MAY simply begin transmitting real-time text (i.e., send <rtt/> elements) upon determining support.
For one-to-one chats, it can be beneficial for clients to easily synchronize the enabling/disabling of real-time text. Upon receiving incoming real-time text, recipient clients MAY automatically do an appropriate response, such as:

- Activate immediately (begin transmitting <rtt/> elements too); or
- Activate after user confirmation prompt (for Privacy considerations); or
- Deny (transmit <rtt event="cancel"/>); or
- Ignore (discard incoming <rtt/> elements); or
- Display only incoming real-time text (e.g., Usage with Multi-User Chat participants control their own outgoing real-time text).

To prevent transmission loops, senders SHOULD NOT transmit <rtt event="init"/> automatically in response to incoming <rtt event="init"/>. Upon sending any <rtt/> elements (except <rtt event="cancel"/>), real-time text is considered activated on the sender side and it is not necessary to transmit <rtt event="init"/> again for the chat session while real-time text is active.
For any client, the preferred first <rtt/> element to send is <rtt event="init"/> as it can quickly signal activation of real-time text, without waiting for the sender to begin composing a new message, and since it is usable regardless of discovery. Also, if the sender was already composing a message when activating real-time text, Message Refresh handles this situation.
While explicit discovery is REQUIRED where possible (see Determining Support), it is not possible to use explicit discovery when the sender does not share a presence subscription with the the contact and knows only their bare JID (e.g., they have yet to receive stanzas from the contact). In this case, the sender client MAY implicitly request and discover the use of real-time text, by sending <rtt event="init"/> upon activation. Senders SHOULD NOT send any further <rtt/> elements, until support is confirmed either by incoming <rtt/> elements or via discovery. Implicit discovery makes it possible to use real-time text as an enhancement to Chat State Notifications (XEP-0085) [17] (Section 5.1), during all situations where it can be used (e.g., when an actively-composing sender appears invisible/offline to the recipient). See Usage with Chat States.

6.2 Deactivating Real-Time Text

Real-time text MAY be deactivated by transmitting <rtt event="cancel"/>, or simply by ending the chat session. Recipient clients SHOULD respond to deactivation with appropriate

[17] XEP-0085: Chat State Notifications <https://xmpp.org/extensions/xep-0085.html>.

response(s), including:

- Stop transmitting <rtt/> elements as well (not applicable to Usage with Multi-User Chat); and
- Handle the sender's unfinished incoming real-time message; and
- Inform the recipient user that sender ended real-time text (or denied/cancelled, if no real-time text was received).

Any client MAY also send an <rtt event="cancel"/> when ending the chat session (e.g., user closes a chat window) or when deactivating real-time text while continuing the chat session. Clients receiving <rtt event="cancel"/> do not need to also transmit <rtt event="cancel"/> back.
Senders deactivating real-time text while in the middle of composing a message can continue composing their message without real-time text being sent. Completed messages continue to be transmitted normally via the Body Element. Recipients that no longer receive further real-time updates, MAY handle the incomplete sender's real-time message appropriately (e.g., clearing/greying-out/saving the message, or using Stale Messages handling).
After deactivation, any client MAY reactivate real-time text again using <rtt event="init"/>.

7 Implementation Notes

7.1 Text Presentation

7.1.1 Avoid Bursty Text Presentation

If a long Transmission Interval is used without Preserving Key Press Intervals, then incoming text will appear in intermittent bursts if the display of text is not smoothed. This hurts user experience of real-time text.

7.1.2 Preserving Key Press Intervals

For high quality presentation of real-time text, the original look-and-feel of typing can be preserved independently of the transmission interval. This is achieved using Element <w/> – Wait Interval between other Action Elements. Sender clients can transmit the length of pauses between key presses, and send multiple key presses in a single <message/> stanza. Recipient clients that process <w/> elements are able to display the sender's typing smoothly without sudden bursts of text. See Examples of Key Press Intervals.
When key press intervals are preserved at high precision, all subtleties of typing are preserved, including the 'mood' (calm typing versus panicked or emphatic typing, etc.). Much as Voice over IP (VoIP) allows accurate packet transmission of sound, this spec allows accurate packet transmission of original typing look-and-feel. This enables the real-time feel of typing

over virtually any network connection, without requiring frequent transmission intervals. Look and feel of typing is also preserved over variable latency connections including XMPP Over BOSH (XEP-0206) [18], mobile phone, satellite and long international connections with heavy packet-bursting tendencies.

7.1.3 Time Critical and Low Latency Methods

There are specialized situations such as live transcriptions and captioning (e.g., transcription service, closed captioning provider, captioned telephone, Communication Access Realtime Translation (CART), relay services) that demand low latency transmission. Such systems typically use voice recognition and/or stenotype machines, which output text in word or phrase bursts rather than a character at a time. It can be acceptable for senders with bursty output to immediately transmit word or phrase bursts of text without buffering, as long as the average stanza rate is not excessive. This eliminates any lag caused by the Transmission Interval. It is not necessary to transmit Element <w/> – Wait Interval for real-time transcription.

7.1.4 Low-Bandwidth and Low-Precision Text Smoothing

Some software platforms (e.g., JavaScript, BOSH, mobile devices) may have low-precision timers that impact Transmission Interval and/or Preserving Key Press Intervals. Clients can optimize for bandwidth, performance and/or screen repaints by eliminating, merging, or ignoring Element <w/> – Wait Interval selectively, especially those containing shorter intervals. In addition, it is acceptable for the transmission interval of <rtt/> to vary, either intentionally for optimizations, or due to precision limitation, preferably within the range recommended by Transmission Interval. Compression can also be used to reduce bandwidth (e.g., TLS compression or Stream Compression (XEP-0138) [19]).
Clients can choose to implement alternate text-smoothing methods, such as adaptive-rate character-at-a-time output, and/or word buffering for incoming real-time text. Word buffering prevents most typing mistakes from being displayed, which can be a useful mode of operation for certain recipients who may dislike watching the sender's typing mistakes.

7.2 Optional Remote Cursor

Recipient clients can choose to display a remote cursor within incoming real-time messages. A remote cursor is a separate cursor/caret indicator within incoming real-time messages, separate of the user's local cursor for outgoing messages. This can improve usability of real-time text, since it becomes easier for a recipient to observe the sender's real-time message edits. For clients that do not implement a remote cursor, skip this section.

[18] XEP-0206: XMPP Over BOSH <https://xmpp.org/extensions/xep-0206.html>.
[19] XEP-0138: Stream Compression <https://xmpp.org/extensions/xep-0138.html>.

Action Elements use only absolute positioning (relative positions are not used by this specification), so clients do not need to remember the position value from previous action elements. Recipient software can calculate the remote cursor position as follows:

- Upon receiving Element <t/> – Insert Text, the cursor position is the 'p' attribute plus the length of the text being inserted. The cursor position is put at the end of the inserted text. *This allows normal forward cursor movement during text insertion.*

- Upon receiving Element <e/> – Erase Text, the cursor position is the 'p' attribute minus the 'n' attribute. *This allows normal backwards cursor movement to a backspace key.*

- Upon receiving an empty Element <t/> – Insert Text (e.g., <t p='#'/> or <t p='#'></t>), the cursor position is the 'p' attribute and no text modification is done. Senders can send these elements when only the cursor position has changed (e.g., arrow keys, mouse repositioning). These are non-operative elements on recipients that do not implement a remote cursor.

7.3 Sending Real-Time Text

This section lists several possible methods of generating real-time text for transmission. For most situations, the preferred methodology is Monitoring Message Changes Instead Of Key Presses.

7.3.1 Monitoring Message Changes Instead Of Key Presses

Experience has found that the most reliable method for generating real-time text, is to monitor for **text changes** to the sender's message entry field, instead of key press events. Text change events have the following advantages:

- It captures all typing, including edits and deletes.
- It captures copy & paste operations, as well as edits made via a pointing device.
- It captures all automatic text changes (e.g., spell checker, auto-correct, macros, transcription, assistive devices).
- It captures characters requiring multiple key presses to compose (e.g., accents, combining marks).
- It makes no assumptions about different keyboards or input method editors (e.g., Chinese).

- Text change events are more portable across platforms, including on mobile phones.

During a text change event, the sender's current message text can be compared to the old message text from the previous text change event. The difference in text, between consecutive text change events, is typically a one character difference (e.g., key press) or one text block difference (e.g., auto-correct, cut, paste). In order to calculate what text changes took place, the first changed character and the last changed character are determined. From this, it is simple to generate Action Elements for a single text block deletion and/or insertion. In addition, if Preserving Key Press Intervals is supported, then Element <w/> – Wait Interval records the time elapsed between text change events.
Sender software can do the following:

1. Monitor for text changes in the sender's message. Whenever a text change event occurs, compute action element(s) and append these action element(s) to a buffer. Repeating this step during every text change event, is equivalent to recording a small sequence of typing.

2. During every Transmission Interval, all buffered action elements are transmitted in <rtt/> element in a <message/> stanza. This is equivalent to transmitting a small sequence of typing at a time.

3. If there are no message changes occurring, no unnecessary transmission takes place.

7.3.2 Monitoring Key Presses Directly

Real-time text can be generated via monitoring key presses. However, this does not have the advantages of Monitoring Message Changes Instead Of Key Presses. Care needs be taken with automatic changes to the message, generated by means other than key presses. This includes spell check auto-correct, copy and pastes, transcription, input method editors, and multiple key presses required to compose a character (i.e., accents). Key press events can miss these text changes, and this can potentially cause incorrect real-time text to be transmitted.

7.3.3 Append-Only Real-Time Text

The use of Element <t/> – Insert Text without any attributes, simply appends text to the end of a message, while the use of Element <e/> – Erase Text without any attributes, simply erases text from the end of the message. This sending method can also be useful for special-purpose clients where mid-message editing capabilities are not used (e.g., simple transcription, news tickers, relay services, captioned telephone).

7.3.4 Simple Real-Time Text

It is possible for sender clients to use Message Refresh to simply re-transmit the whole real-time message, as a method of transmitting text changes. The advantage is very simple implementation. Disadvantages can include the lack of Preserving Key Press Intervals, and extra bandwidth consumption that can occur with longer messages, unless stream compression is used. The below illustrates transmission of the real-time message "**Hello there!**" at a regular Transmission Interval while the sender is typing.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>Hel</t>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='b02'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='456002' event='reset'>
    <t>Hello th</t>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='c03'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='789003' event='reset'>
    <t>Hello there!</t>
  </rtt>
</message>
```

Note: The 'seq' attribute can be restarted at any value with <rtt event="reset"/> and <rtt event="new"/>. See Processing Rules.

7.4 Receiving Real-Time Text

In order to allow Preserving Key Press Intervals in incoming real-time text, recipient clients can do the following:

1. Upon receiving Action Elements in incoming <rtt/> elements, they are added to a queue in the order they are received. This provides immunity to variable network conditions, since the queueing action will smooth out incoming transmission (e.g., receiving new <rtt/> while still processing action elements from a delayed <rtt/>).

2. The recipient client processes action elements in the queue in sequential order, including pauses from Element <w/> – Wait Interval, if supported. This is equivalent to

playing back the sender's original typing.

If Element <w/> – Wait Interval is supported, excess lag in incoming real-time text can occur when delayed <rtt/> elements get delivered (e.g., congestion, intermittent wireless reception). To avoid delayed presentation of real-time text, the recipient client needs to speed up processing of action elements. This can be accomplished through a variety of techniques, such as shortening the pauses ('n' value) in <w/> elements, ignoring excess <w/> elements, immediately outputting action elements that are still queued, and/or keeping action elements from a limited number of <rtt/> elements queued (immediately outputting any prior action elements). This allows lagged real-time text to catch up more quickly.
Upon receiving a Body Element indicating a completed message, it is acceptable for the full message text from <body/> to be displayed immediately in place of the real-time message, and discard any unprocessed action elements. This prevents any delay in displaying the final message delivery, however, this may cause a sudden surge of text in some situations.
If the <w/> element is not supported, receiving clients can use an alternate text-smoothing method in order to Avoid Bursty Text Presentation (e.g., time-smoothed progressive output of received real-time text).

7.5 Other Guidelines

7.5.1 Message Length

A large sequence of action elements can result in an <rtt/> larger than the size of a message <body/>. This can occur normally during fast typing when Preserving Key Press Intervals during small messages. However, if the <rtt/> element becomes unusually large (e.g., macros, multiple copy and pastes, leading to an <rtt/> exceeding one kilobyte) a Message Refresh can instead be used, in order to save bandwidth. (Stream compression is another approach.)
Clients can limit the length of the text input for the sender's message, in order to keep the size of <message/> stanzas reasonable, including during Message Refresh. Also, large <rtt/> elements might occur in situations such as large copy and pastes. To keep message stanza sizes reasonable, <rtt/> can be transmitted in a separate <message/> than the one containing <body/>.
For clients that send continuous real-time text (e.g., news ticker, captioning, speech transcription, TTY/text telephone gateway), a Body Element can be sent and then a new real-time message started immediately after, every time a message reaches a specific size. This allows continuous real-time text without real-time messages becoming excessively large.

7.5.2 Usage with Chat States

Real-time text can be used in conjunction with **Chat State Notifications**. It is best to handle **XEP-0301** and **XEP-0085** transmissions in separate <message/> stanzas. Chat states such as <composing/> or <active/> are sent separately from <rtt/> elements.

Chat states are handled as specified by **XEP-0085**. The continuous transmission of real-time text corresponds to a <composing/> chat state. Therefore, the timing of the <composing/> chat state coincides with the beginning of continuous <rtt/> transmission.

7.5.3 Usage with Last Message Correction

It is possible to use **Last Message Correction** with real-time text. If **XEP-0308** is implemented in concert with this specification, the following rules apply:

- For all <rtt/> elements transmitted during composing a new message, the 'id' attribute of <rtt/> is not used.

- For all <rtt/> elements transmitted during editing of the previous message, the 'id' attribute of <rtt/> matches the 'id' attribute of the old <message/> stanza containing the <body/> text being edited (see 'Business Rules' in **XEP-0308**). This enables recipient clients to display real-time text while the sender is editing the previously-delivered message.

- Senders clients need to transmit a Message Refresh when transmitting <rtt/> for a different message than the previously transmitted <rtt/> (i.e., the value of the 'id' attribute changes, 'id' becomes included, or 'id' becomes not included). This keeps real-time text synchronized when beginning to edit a previously delivered message versus continuing to compose a new message.

- The **XEP-0301** and **XEP-0308** protocols operate concurrently via separate message stanzas. Thus, a message stanza never simultaneously includes both <rtt/> and <replace/>.

- The Body Element delivers a finished new message or a finished message correction (<replace/> is used with <body/> in accordance to **XEP-0308**).

7.5.4 Usage with Multi-User Chat

For simplicity, clients can implement real-time text only for one-to-one chat, and not for **Multi-User Chat**. However, it can be appropriate to support <rtt/> elements in groupchat rooms, even if not all participants support real-time text, as long as the service allows it (See Support for Groupchat).
Participants that enable real-time text during group chat need to keep track of multiple concurrent real-time messages on a per-participant basis. Participants, with real-time text, will

see real-time text coming from each participant that has real-time text enabled. Participant clients without real-time text (whether unsupported or turned off) will simply see group chat function normally on a line-by-line basis, since it is Backwards Compatible.
Participants that turn off real-time text for themselves, can simply ignore incoming <rtt/> and not transmit outgoing <rtt/>. Participant clients in groupchat receiving an incoming <rtt event='cancel'/> needs to keep outgoing transmission unaffected during Deactivating Real-Time Text (otherwise, one participant could deny real-time text between other willing participants).
To minimize on-screen clutter of multiple idle real-time messages, clients can hide idle messages, clear old Stale Messages, and/or prioritize the display of the most useful real-time messages. Prominent visibility of real-time text can be assigned to recent typists and/or moderators (e.g., classroom teacher, convention speaker). For the same participant logged in multiple times in the same room, see Simultaneous Logins for handling this situation. In situations of simultaneous typing by a large number of participants, see Congestion Considerations.

7.5.5 Simultaneous Logins

In situations where there are multiple sessions from the same JID (i.e., simultaneous logins on multiple clients/devices), transmitting of <rtt/> works in one-to-many situations without any special software support. For many-to-one situations where there is incoming <rtt/> from multiple sessions under the same JID, Keeping Real-Time Text Synchronized will pause the real-time message upon conflicting <rtt/>, and resume during the next Message Refresh, presumably from the active session. This provides a seamless system-switching experience. A good implementation of Message Refresh will improve user experience, regardless of whether or not the client follows Best Practices for Resource Locking (XEP-0296) [20]. Clients can choose to distinguish the <rtt/> streams (via full JID and/or via <thread/>) and keep multiple concurrent real-time messages similar in manner to Usage with Multi-User Chat, with the Stale Messages being timed-out.

7.5.6 Stale Messages

There are situations where senders pause typing indefinitely. This can result in recipients displaying a real-time message for an extended time period. It may also be a screen clutter concern during Usage with Multi-User Chat. In addition, it may be a resource-consumption concern, as part of Congestion Considerations.
It is acceptable for recipients to clear (and/or save) incoming real-time messages that have been idle for an extended time period. There is no specific time-out period defined by this specification. For Usage with Multi-User Chat, the time-out period might be shorter because of the need to reduce screen clutter. For one-to-one chat sessions, the time-out period might need to be longer to allow reasonable interruptions (i.e., sender pausing during a long phone call or other interruption).

[20] XEP-0296: Best Practices for Resource Locking <https://xmpp.org/extensions/xep-0296.html>.

Senders that resume composing a message (i.e., continues a partially-composed message hours later) can do a Message Refresh, which allows recipients to redisplay the real-time message.

7.5.7 Performance & Efficiency

With real-time text, frequent screen updates can occur. Screen updates are a potential performance bottleneck, since fast typists type many key presses per second. Optimizing screen updates is more important on slower platforms. The real-time message might be implemented as a separate window or separate display element.
Battery life considerations are closely related to performance, as the addition of real-time text can have an impact on battery life. If Preserving Key Press Intervals is supported, then support for Element <w/> – Wait Interval needs to be implemented in a battery-efficient manner. The Transmission Interval can vary dynamically to optimize for battery life and wireless reception. For devices where screen updates are an unavoidable, inefficient bottleneck, see Low-Bandwidth and Low-Precision Text Smoothing to reduce the number of screen updates per second.

8 Use Cases

Most of these examples are deliberately kept simple. In complete software implementations supporting key press intervals, transmissions will most resemble the last example, Full Message Including Key Press Intervals. For simplicity, these examples use a bare JID, even in situations where a full JID might be more appropriate.

8.1 Introductory Examples of Real-Time Text

All three examples shown below result in the same real-time message "**HELLO**" created by writing "**HLL**", backspacing two times, and then "**ELLO**". The action elements are Element <t/> – Insert Text and Element <e/> – Erase Text.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>HLL</t>
    <e/><e/>
    <t>ELLO</t>
  </rtt>
</message>
```

The example above sends the misspelled "**HLL**", then **<e/><e/>** backspaces 2 times, then sends "**HELLO**".

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>HLL</t>
    <e n='2'/>
    <t>ELLO</t>
  </rtt>
</message>
```

The example above shows that **<e n='2'/>** does the same thing as **<e/><e/>**.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>HLL</t>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='b02'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123002'>
    <e n='2'/>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='c03'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123003'>
    <t>ELLO</t>
  </rtt>
</message>
```

The example above splits the same real-time text over multiple <message/> stanzas, which would occur if the typing was occurring more slowly, over several Transmission Interval cycles.

8.2 Example of Multiple Messages

The example below represents a short chat session of three separate messages: Bob says: "Hello Alice" Bob says: "This is Bob" Bob says: "How are you?"

```
<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='a01'>
```

```
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>Hello</t>
  </rtt>
</message>

<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='b02'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123002'>
    <t> Alice</t>
  </rtt>
  <body>Hello Alice</body>
</message>


<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='c03'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='456001' event='new'>
    <t>This i</t>
  </rtt>
</message>

<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='d04'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='456002'>
    <t>s Bob</t>
  </rtt>
  <body>This is Bob</body>
</message>


<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='e05'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='789001' event='new'>
    <t>How a</t>
  </rtt>
</message>

<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='f06'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='789002'>
    <t>re yo</t>
  </rtt>
</message>

<message to='alice@example.com' from='bob@example.com/home' type='chat
    ' id='g07'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='789003'>
    <t>u?</t>
  </rtt>
```

```
  <body>How are you?</body>
</message>
```

The example above represents moderate typing speed during a normal Transmission Interval, such as 700 milliseconds between <message/> stanzas for continuous typing. It illustrates the following RTT Attributes:

- The 'event' attribute equals "new" for the start of every new message.
- The 'seq' attribute increments within the same message.
- The 'seq' attribute randomizes when beginning a new message.

8.3 Examples of Message Edits

These examples illustrate real-time message editing via Action Elements. Note: In most situations, during normal human typing speeds at a normal Transmission Interval, smaller fragments of text will be spread over multiple <rtt/> elements, than these demonstration examples below. See Sending Real-Time Text.

8.3.1 Deleting Text From Message

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>Hello Bob, this is Alice!</t>
    <e n='4' p='9'/>
  </rtt>
</message>
```

Final result of real-time message: "**Hello, this is Alice!**" This example outputs "**Hello Bob, this is Alice!**" then **<e n='4' p='9'/>** erases 4 characters before character position index 9. The Element <e/> – Erase Text removes the text " **Bob**" including the preceding space character.

8.3.2 Inserting Text Into Message

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>Hello, this is Alice!</t>
    <t p='5'> Bob</t>
  </rtt>
</message>
```

Final result of real-time message: "**Hello Bob, this is Alice!**" This is because this example outputs "**Hello, this is Alice!**" then the **<t p='5'>** inserts the specified text " **Bob**" at position 5, using Element <t/> – Insert Text.

8.3.3 Deleting and Replacing Text In Message

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>Hello Bob, tihsd is Alice!</t>
    <e p='16' n='5'/>
    <t p='11'>this</t>
  </rtt>
</message>
```

Final result of real-time message: "**Hello Bob, this is Alice!**" This example outputs "**Hello Bob, tihsd is Alice!**", then **<e p='16' n='5'/>** erases 5 characters at position 16 in the string of text (which erases the mistyped word "**tihsd**"). Finally, **<t p='11'>this</t>** inserts the text "**this**" place of the original misspelled word.

8.3.4 Multiple Message Edits

This is an example message containing multiple consecutive real-time message edits. This illustrates valid use of the <rtt/> element.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>Helo</t>
    <e/>
    <t>lo...planet</t>
    <e n='6'/>
    <t> World</t>
    <e n='3' p='8'/>
    <t p='5'> there,</t>
  </rtt>
</message>
```

Resulting real-time message: "**Hello there, World**", completed in the following series of action elements:

Element	Action	Real -Time Message	Cursor Position*
<t>Helo</t>	Output "Helo"	Helo	4

Element	Action	Real -Time Message	Cursor Position*
<e/>	Erase 1 character from end of line.	Hel	3
<t>lo...planet</t>	Output "lo...planet" at end of line.	Hello...planet	14
<e n='6'/>	Erase 6 characters from end of line	Hello...	8
<t> World</t>	Output " World" at end of line.	Hello... World	14
<e n='3' p='8'/>	Erase 3 characters before position 8	Hello World	5
<t p='5'> there,</t>	Output " there," at position 5	Hello there, World	12

*The Cursor Position column is only relevant if the Optional Remote Cursor is implemented. This example does not illustrate Preserving Key Press Intervals. Also, it is noted that most situations, during normal typing speeds at a normal Transmission Interval, the above series of Action Elements will normally be spread over multiple separate <rtt/> elements.

8.4 Examples of Key Press Intervals

8.4.1 Comparison With and Without Intervals

All examples shown below, result in the same real-time message "**HELLO**". Only the last example follows Preserving Key Press Intervals.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>HELLO</t>
  </rtt>
</message>
```

The above example outputs "**HELLO**" in a single action element (Element <t/> – Insert Text).

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>H</t>
    <t>E</t>
    <t>L</t>
    <t>L</t>
    <t>O</t>
  </rtt>
</message>
```

The above example outputs "**HELLO**" in separate action elements for each key press.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>H</t><w n='101'/>
    <t>E</t><w n='110'/>
    <t>L</t><w n='125'/>
    <t>L</t><w n='103'/>
    <t>O</t><w n='110'/>
  </rtt>
</message>
```

The above example outputs "**HELLO**" in separate action elements for each key press, while also Preserving Key Press Intervals. The Element <w/> – Wait Interval specifies the number of milliseconds between key presses, to allow smooth presentation in recipient clients that support <w/> action elements.

8.4.2 Full Message Including Key Press Intervals

This example is a transmission of "Hello there!" while Preserving Key Press Intervals. It illustrates a four-second typing sequence:

- The misspelled phrase "Hello tehre!" is typed;
- Optional transmission of cursor movements towards the typing mistake;
- Two backspaces to delete the typing mistake;
- Two correct key presses to correctly spell the word "there".

The use Element <w/> – Wait Interval, between key presses, allows the receiving client to execute a small pause between action elements. This allows recipient clients to play back the sender's typing fluidly.

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='a01'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123001' event='new'>
    <t>H</t>
    <w n='115'/><t>e</t>
    <w n='154'/><t>l</t>
    <w n='151'/><t>l</t>
    <w n='115'/><t>o</t>
    <w n='165'/>
  </rtt>
</message>
```

```
<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='b02'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123002'>
    <w n='40'/><t> </t>
    <w n='161'/><t>t</t>
    <w n='137'/><t>e</t>
    <w n='135'/><t>h</t>
    <w n='134'/><t>r</t>
    <w n='93'/>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='c03'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123003'>
    <w n='109'/><t>e</t>
    <w n='115'/><t>!</t>
    <w n='330'/><t p='11'/>
    <w n='108'/><t p='10'/>
    <w n='38'/>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='d04'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123004'>
    <w n='109'/><t p='9'/>
    <w n='111'/><e p='9'/>
    <w n='106'/><e p='8'/>
    <w n='138'/><t p='7'>h</t>
    <w n='209'/><t p='8'>e</t>
    <w n='27'/>
  </rtt>
</message>

<message to='bob@example.com' from='alice@example.com/home' type='chat
    ' id='d04'>
  <rtt xmlns='urn:xmpp:rtt:0' seq='123005'>
    <w n='445'/><t p='12'/>
  </rtt>
  <body>Hello there!</body>
</message>
```

This example also illustrates the following:

- Typing is done via Element <t/> – Insert Text.
- Backspaces are done via Element <e/> – Erase Text.

- There is a final transmission with a Body Element, when the message is finished.
- Intervals between key presses are done via Element <w/> – Wait Interval.
- Each <message/> is delivered at a regular Transmission Interval, typically 700 milliseconds.
- Cursor movements via empty <t/> elements. Sender transmission is not essential, but can be desirable for recipient clients supporting an Optional Remote Cursor.
- Recipient clients that do not support Preserving Key Press Intervals and/or Optional Remote Cursor, will still display this message normally.
- The total sum of all values in Element <w/> – Wait Interval in one <message/> equal the Transmission Interval during periods of continuous typing. This also results in some <w/> interval elements being split between consecutive messages. Although not critical, it can further improve the fluidity of Receiving Real-Time Text.
- See Monitoring Message Changes Instead Of Key Presses for the best method of implementation.

9 Interoperability Considerations

There are other real-time text formats with interoperability considerations relating to the session setup level, the media transport level, and presentation level. Interoperability specifications between multiple real-time text formats can be found at Real-Time Text Taskforce (R3TF).
Implementers ought to choose the most appropriate real-time text approach for the session control technology in use during a particular session. For example, clients that use XMPP can utilize this **XEP-0301** specification, and clients that use SIP might utilize **RFC 4103**, **RFC 5194** [21] and **ITU-T T.140**. Clients that run on multiple networks, might need to utilize multiple real-time text technologies. To interoperate between incompatible real-time text technologies, gateway servers can transcode between different real-time text technologies, along with other media such as audio and video. This can include TTY and textphones.

9.1 RFC 4103 and T.140

In the SIP environment, real-time text is specified in **RFC 4103** and **ITU-T T.140**. SIP is a popular real-time session control protocol, and there are many implementations of real-time text controlled by SIP. This includes emergency services in some regions.
Interoperability considerations include addressing translation, media negotiation and translation, and media transcoding. Transcoding is straightforward between this specification and

[21] RFC 5194: Framework for Real-Time Text over IP Using the Session Initiation Protocol (SIP) <http://tools.ietf.org/html/rfc5194>.

T.140 / RFC 4103, except for editing in the middle of messages. Text insertions or deletions, occurring far back in the message, can cause a large number of erase operations in **T.140** that consume time and bandwidth. **T.140** specifies the use of ISO 6429 control codes for presentation characteristics, such as text color, that are not supported by this specification. During transcoding, these control codes needs to be filtered off in order to not disturb the presentation of text. Guidance on address translation and conveyance between XMPP and SIP can be found in **draft-ietf-stox-core** [22].

9.2 Total Conversation – Combination with Audio and Video

According to **ITU-T Rec. F.703**, the "Total Conversation" standard defines the simultaneous use of audio, video, and real-time text. For convenience, real-time communication applications can be designed to have automatic negotiation of as many as possible of the three media preferred by the users.
In the XMPP session environment, the Jingle protocol (Jingle (XEP-0166) [23]) is available for negotiation and transport of the more time-critical, real-time audio and video media. Any combination of audio, video, and real-time text can be used together simultaneously.

10 Internationalization Considerations

The primary internationalization consideration involves real-time message editing using Action Elements, where text is inserted and deleted using position and length values. For this, Accurate Processing of Action Elements including correct Unicode Character Counting will ensure that all possible valid Unicode text can be used via this protocol. This includes text containing multiple scripts/languages, ideographic symbols (e.g., Chinese), right-to-left text (e.g., Arabic), and bidirectional text.
For accessibility considerations, there is an **International Symbol of Real-Time Text** [24] to alert users to the existence of this feature.

11 Security Considerations

11.1 Privacy

It is important for users to be made aware of real-time text (e.g., user consent, software notice, introductory explanation). Users of real-time text need to be aware that their typing is now visible in real-time to everyone in the current chat conversation. There can be potential

[22] Interworking between the Session Initiation Protocol (SIP) and the Extensible Messaging and Presence Protocol (XMPP): Addresses and Error Conditions <http://tools.ietf.org/html/draft-ietf-stox-core>.
[23] XEP-0166: Jingle <https://xmpp.org/extensions/xep-0166.html>.
[24] The International Symbol of Real-Time Text <http://www.fasttext.org>.

security implications if users copy & paste private information into their chat entry buffer (e.g., a shopping invoice) before editing out the private parts of the pasted text (e.g., a credit card number) and then sending the message. There can also be implications for chat clients that suddenly pop up a chat window upon incoming messages and takes keyboard focus unexpectedly, resulting in the sender typing sensitive information into the wrong window. These accidental privacy risks are also apparent for traditional chat (e.g., accidentally sending a message) but are more immediate for real-time text. With real-time message editing, recipients can watch all text changes that occur in the sender's text, before the sender finishes the message.
Such risks can be avoided by good user interface design. In addition, implementation behaviors and improved education can be added to reduce privacy issues. Examples include showing an introduction upon first activation of feature, special handling for copy and pastes (i.e., preventing them, or prompting for confirmation), recipient confirmation of real-time text via Guidelines for Initiating Real-Time Text, etc.

11.2 Encryption

Real-time text (<rtt/> elements) transmits the content contained within messages. Therefore, a client that encrypts <body/> also needs to encrypt <rtt/> as well:

- Encryption at the stream level (e.g., TLS) can be used normally with this specification. Stream-level encryption is the most common form of encryption.

- Encryption at the <message/> stanza level (e.g., XEP-0200) can be used for all stanzas containing either <rtt/> or <body/>. It is noted that real-time text can have a higher rate of message stanzas, contributing to additional overhead. See Congestion Considerations.

- Encryption at the <body/> level (e.g., deprecated XEP-0027) does not encrypt <rtt/>. In this case, <rtt/> needs to be encrypted separately. It is preferable to use a broader level of encryption, where possible.

It is possible for the timing of individual key presses to be used as a timing attack on encryption. Protection against this is provided by buffering of key presses into a regular Transmission Interval. As an additional measure of security, the risk of timing attacks can be further mitigated by padding <rtt/> elements to lengths not clearly related to the number of characters in the message. Alternatively, general XMPP protection mechanisms hiding length information can be applied on the complete message exchange instead of (or in concert with) <rtt/> specific protection mechanisms.

11.3 Congestion Considerations

The nature of real-time text can result in more frequent transmission of <message/> stanzas than would otherwise happen in a non-real-time text conversation. This can lead to increased network and server loading of XMPP networks.
Transmission of real-time text can be throttled temporarily during poor network conditions. It is appropriate to use latency monitoring mechanisms (e.g., Message Delivery Receipts (XEP-0184) [25] or Stream Management (XEP-0198) [26]) in order to temporarily adjust the Transmission Interval of real-time text beyond the recommended range. This results in lagged text (less real-time) but is better than failure during poor network conditions. The use of Message Refresh can also retransmit real-time text lost by poor network conditions, including stanzas dropped during a network issue or server error. These techniques are useful for mission-critical applications such as next generation emergency services (e.g., text to 9-1-1).
Excess numbers of real-time messages (e.g., during a Denial of Service (DoS) scenario in Usage with Multi-User Chat) might cause local resource-consumption issues, which can be mitigated by accelerated time-out of Stale Messages. Also see Best Practices to Discourage Denial of Service Attacks (XEP-0205) [27].
According to multiple university studies worldwide (including **Carnegie Mellon University Study** [28]), the average length of instant messages is under 40 characters. The additional incremental bandwidth overhead of real-time text can be very low for an existing XMPP client, especially one already using many extensions. Bandwidth can also be further mitigated using stream compression, to benefit bandwidth-constrained networks (e.g., GPRS, 3G, satellite).

12 IANA Considerations

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

13 XMPP Registrar Considerations

13.1 Protocol Namespaces

The XMPP Registrar [29] includes ”urn:xmpp:rtt:0” in its registry of protocol namespaces (see <http://xmpp.org/registrar/namespaces.html>).

[25] XEP-0184: Message Delivery Receipts <https://xmpp.org/extensions/xep-0184.html>.
[26] XEP-0198: Stream Management <https://xmpp.org/extensions/xep-0198.html>.
[27] XEP-0205: Best Practices to Discourage Denial of Service Attacks <https://xmpp.org/extensions/xep-0205.html>.
[28] Communication Characteristics of Instant Messaging: Effects and Predictions of Interpersonal Relationships <http://seattle.intel-research.net/~{}davraham/pubs/Avrahami_CSCW_06.pdf>.
[29] 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/>.

13.2 Namespace 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.

14 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:rtt:0'
    xmlns='urn:xmpp:rtt:0'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0301: http://www.xmpp.org/extensions/xep-0301.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='rtt'>
    <xs:complexType>
      <xs:attribute name='seq' type='xs:unsignedInt' use='required'/>
      <xs:attribute name='event' use='optional' default='edit'>
        <xs:simpleType>
          <xs:restriction base='xs:string'>
            <xs:enumeration value='new'/>
            <xs:enumeration value='reset'/>
            <xs:enumeration value='edit'/>
            <xs:enumeration value='init'/>
            <xs:enumeration value='cancel'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name='id' type='xs:string' use='optional'>
      <xs:sequence>
        <xs:element ref='t' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element ref='e' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element ref='w' minOccurs='0' maxOccurs='unbounded'/>
      </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name='t' type='xs:string'>
    <xs:complexType>
      <xs:attribute name='p' type='xs:unsignedInt' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='e' type='empty'>
    <xs:complexType>
      <xs:attribute name='p' type='xs:unsignedInt' use='optional'/>
      <xs:attribute name='n' type='xs:unsignedInt' use='optional'
          default='1'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='w' type='empty'>
    <xs:complexType>
      <xs:attribute name='n' type='xs:unsignedInt' use='required'/>
    </xs:complexType>
  </xs:element>

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

</xs:schema>
```

15 Acknowledgments

The members of the Real-Time Text Taskforce [30] made significant contributions to this specification. Mark Rejhon leads the Jabber/XMPP Taskgroup at R3TF. Members of R3TF who have contributed to this specification include Gunnar Hellstrom, Paul E. Jones, Gregg Vanderheiden, Barry Dingle, and Arnoud van Wijk. Others contributors include Bernard Aboba, Mark Grady, Darren Sturman, Christian Vogler, Norm Williams, and several members from the XMPP Standards Mailing List, including Kevin Smith, Peter Saint-Andre and many others.

The technique of Preserving Key Press Intervals, otherwise called ”natural typing”, was created by Mark Rejhon, who is deaf. It is incorporated into this specification in compliance with the XSF IPR Policy [31].

[30] Real-Time Text Taskforce (R3TF) <http://realtimetext.org/>.

[31] The XSF IPR Policy defines the XMPP Standards Foundation's official policy regarding intellectual property rights (IPR) as they pertain to XMPP Extension Protocols (XEPs). For further information, see <https://xmpp.org/a

bout/xsf/ipr-policy>.