

# **GENESYS**

This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

## **API** Reference

Controlling the Chat Session

# Controlling the Chat Session

|   | Events  |                          |
|---|---|--------------------------|
| Commands  | About chat session events                       |                          |
|   | session.onError                                 |                          |
| <ul> <li>About chat session<br/>commands</li> </ul> | session.onAgentConnected                        | Miscollanoous            |
| session.getTranscript                               | <ul> <li>session.onAgentDisconnected</li> </ul> | Miscellaneous            |
| session.sendMessage                                 | <ul> <li>session.onMessageReceived</li> </ul>   | session.isAgentConnected |
| <ul> <li>session.sendTyping</li> </ul>              | <ul> <li>session.onAgentTyping</li> </ul>       |                          |
| session.leave                                       | session.onInterrupted                           |                          |
|   | session.onContinued                             |                          |
|   | session.onSessionEnded                          |                          |

## Chat session commands

Session commands are used for client-to-server communication: sending commands to chat server. All commands receive their arguments as a single "options" object, similar to startSession and restoreSession. For example,

session.getTranscript({fromIndex: 0});

// instead of
session.getTranscript(0);

All exceptions to this rule are documented here.

#### **Returned Promises**

All of the commands return a "promise" object with two properties: done and fail.

Some of the "done" callbacks can be used to obtain specific information provided as the result of command execution. If that is the case, the information is specifically documented for particular commands. Most of the time however these callbacks serve as a way to simply acknowledge that the command was successfully executed. In this case, there is no specific documentation for the corresponding command callback.

"fail" callbacks can be used to catch errors that happen during command execution and all receive an object with exact same structure:

| event.error.code        | Code specifying the particular error                    |
|-------------------------|---|
| event.error.description | Description of the error (English is default language). |

There is no specific documentation for each command's "fail" callback.

#### Here is an example of general command usage:

```
chat.<ABSTRACT_CHAT_COMMAND>(<COMMAND_OPTIONS>).done(function(<POSSIBLE_RESULTS>) {
    // Command executed successfully.
}).fail(function(event) {
    // Something went wrong. See event.error.code and event.error.description.
});
```

## session.getTranscript

#### Description

A low-level method used to obtain the transcript for the current session.

Important

No playback is assumed.

#### Options

| Parameter | Туре   | Default value | Mandatory | Description  |
|-----------|--------|---------------|-----------|--|
| fromIndex | number | 0             | No        | 0-based index to<br>retrieve chat<br>transcript from a<br>certain position.<br>0 means obtaining the<br>entire transcript. |

#### "done" callback

Receives the transcript in serialized JS form.

| Parameter        | Туре  | Description  |
|------------------|-------|--|
| event.transcript | Array | Array of JS objects representing<br>transcipt events.<br>Sample:<br>[<br>{ |

| Parameter | Туре | Description  |
|-----------|------|--|
|           |      | <pre>type: 'AgentConnected',<br/>party: {id: 2 /*ID of<br/>this party unique for THIS<br/>chat session*/, type:<br/>'Agent', name: <name an<br="" of="">agent&gt;},<br/>index: <index of="" this<br="">message in chat transcript&gt;,<br/>timestamp: <utc<br>Timestampt&gt;<br/>},<br/>{<br/>type: 'MessageReceived',<br/>party: {id: 1, type:<br/><type of="" party="" sent<br="" which="">message: 'Client' or 'Agent'<br/>or 'External'&gt;, name: <name<br>of party&gt;},<br/>content: {text: <text<br>which was sent&gt;, type: <type<br>of text: 'text' or 'url'&gt;},<br/>index: <index of="" this<br="">message in chat transcript&gt;,<br/>timestamp: <utc<br>Timestampt&gt;<br/>},<br/>{<br/>type:<br/>'AgentDisconnected',<br/>party: {id: 2 /*ID of<br/>this party unique for THIS<br/>chat session*/, type:<br/>'Agent', name: <name an<br="" of="">agent&gt;},<br/>index: <index of="" this<br="">message in chat transcript&gt;,<br/>timestamp: <utc<br>Timestampt&gt;<br/>},<br/>{<br/>type: 'SessionEnded',<br/>reason: {code: <code of<br="">reason: 1: by leave request,<br/>2 - by an agent, 3 - by<br/>error&gt;, description:<br/><default description="">,<br/>timestamp: <utc<br>Timestampt&gt;<br/>}<br/>]</utc<br></default></code></utc<br></index></name></utc<br></index></type<br></text<br></name<br></type></utc<br></index></name></pre> |

session.getTranscript().done(function(event) {
 console.log('Full transcript of current chat session: ', event.transcript);
});

## session.sendMessage

#### Description

Send a message to Chat Server.

#### Options

| Parameter | Туре   | Default value | Mandatory | Description   |
|-----------|--------|---------------|-----------|---|
| type      | string | 'text'        | No        | Type of message:<br>"text" or "url"<br>If absent, "text" will be<br>used. |
| message   | string | undefined     | Yes       | Message to be sent  |

Since sending a "text" message is a much more frequent operation than sending a "url", a shortcut is available; you can pass a string with message contents directly to the method instead of the options object.

```
// This
session.sendMessage({ type: 'text', message: 'foobar' });
// is equivalent to this:
session.sendMessage('foobar');
```

## session.sendTyping

#### Description

Notify Chat Server that client started or stopped typing in chat session.

#### Options

| Parameter | Туре    | Default value | Mandatory | Description   |
|-----------|---------|---------------|-----------|---|
| isTyping  | boolean | true          | No        | <ul> <li>Boolean (true or false) which specifies exact meaning of this command</li> <li>true — visitor started typing (or continues typing) in chat session.</li> <li>false — visitor stopped typing</li> </ul> |

| Parameter | Туре | Default value | Mandatory | Description  |
|-----------|------|---------------|-----------|--|
|           |      |               |           | in chat session<br>(typically a<br>stop or pause<br>in typing for a<br>certain<br>duration, for<br>example — 5<br>seconds) |

#### session.leave

#### Description

This command is used to complete chat session in Chat Server by request from the visitor side.

#### Options

This command does not take parameters.

## Chat session events (callbacks)

You can pass callback functions into a chat session that will be called each time a chat context is updated, or whenever other changes take place within the session (for example, agent joins/leaves, Chat Server stops responding, and so on). Another use for session events is in a "playback" scenario, when a chat session is restored in a new browser context (for example, after page reload/navigation).

To add a callback, pass the callback function directly to the corresponding event method. Most of the callbacks receive an event object with properties containing event details. For example,

```
session.onError(function(event) {
    // event.error.code
    // event.error.description
});
```

## session.onError

This event will be sent in reaction to an unexpected error occurring during the flow of the chat session.

Additionally, this event is sent when the chat component needs to notify clients about unusual cases. For example:

- Chat Server stops responding and the component tries to restore the session on another server.
- A chat session is restored on another instance of Chat Server.
- A channel to the Chat Server is opened, but the server is not yet ready to send/receive operations.

Another important scenario for this callback — it is triggered if an incorrect set of parameters (or invalid parameter value) is detecing in an incoming "raw" event. For example, if the content of a message in a received message event is not a string.

#### **Event structure**

| Parameter               | Description                          |
|-------------------------|--------------------------------------|
| event.error.code        | Code specifying the particular error |
| event.error.description | Default description of error         |

#### Tip

For a list of possible error codes, see Error Codes.

## session.onAgentConnected

Executed when an agent joins a chat session.

#### **Event structure**

| Parameter        | Description   |
|------------------|---|
| event.party.name | String that represents name of the agent joined to the chat session   |
| event.timestamp  | Number  |
| event.party.id   | Theoretically, the agent name and visitor name<br>could be the same (especially since the visitor<br>might be filling out the the visitor name). To handle<br>this scenario, this id is used to distinguish between<br>agent and visitor names. |
| event.index      | Index of this message in chat transcript  |
| event restored   | Optional.<br>If present and true, it means that the event was restored during   |
|                  | session restoration (in other words, event was already reported to the consumer previously).  |

## session.onAgentDisconnected

Executed when agent leaves chat session, for any of these possible reasons:

- Session closes because of a logout request from the Chat Widget side.
- Agent leaves the conference.
- Agent transfers the session to another agent.
- agent's desktop stops responding.
- Chat server stops responding and session is restored on the new chat session.

#### **Event structure**

| Parameter        | Description   |
|------------------|---|
| event.timestamp  | Number  |
| event.party.name | String that represents the name of the agent leaving chat session.  |
| event.party.id   | String with ID of party in chat session.  |
| event.index      | Index of this message in chat transcript  |
| event.restored   | Optional.<br>If present and true, it means that the event was restored during session restoration (in other words, was already reported to the consumer previously) |
|                  | consumer previously).   |

## session.onMessageReceived

Executed when a new message appears in the chat transcript (or in the context of the "playback" process during chat session restoration).

### Important

Messages sent by session.sendMessage are "returned back" via this event as well.

#### **Event structure**

| Parameter          | Description  |
|--------------------|--|
| event.index        | 0-based index of this message in chat session transcript |
| event.timestamp    | Number   |
| event.content.text | String with message added to the chat context            |

| Parameter  | Description   |
|--|---|
|  | One of these is true depending on message type.<br>Usage example:   |
| event.content.type.url<br>OR event.content.type.text | <pre>session.onMessageReceived(function(event) {     if (event.content.type.url) {         // this is "url" message     } else if (event.content.type.text) {         // this is "text" message     } });</pre>                             |
| event.party.id                                       | Theoretically, the agent name and visitor name<br>could be the same (especially since the visitor may<br>be the one filling out the visitor name). To handle<br>this scenario, this ID is used to distinguish between<br>agent and visitor. |
|  | One of these is set to true, depending on who sends the message.  |
| event.party.type.agent                               | <ul> <li>agent — message is sent by the agent;</li> </ul>   |
| OR<br>event.party.type.client<br>OR                  | <ul> <li>client — message is sent by the visitor (in other<br/>words, the actual user of this API via<br/>session.sendMessage())</li> </ul>   |
| event.party.type.external OR                         | • <b>external</b> — message is sent by the system (for  |
| event.party.type.supervisor                          | <ul> <li>supervisor — message is sent by the supervisor</li> </ul>  |
| event.party.name                                     | String with name of party.  |
|  | Optional.   |
| event.restored                                       | If present and true, it means that the event was restored during session restoration (in other words, event was already reported to the consumer).  |

## session.onAgentTyping

Executed when agent starts, continues, or stops typing.

#### **Event structure**

| Parameter        | Description  |
|------------------|--|
| event.party.id   | ID of party in chat session.                             |
| event.party.name | Name of agent.   |
| event.isTyping   | • <b>true</b> — when agent starts (or continues) typing; |

| Parameter | Description                               |
|-----------|---|
|           | • <b>false</b> — when agent stops typing. |

## session.onInterrupted

Executed when a connection to the server is interrupted (for example, a network interruption or server is down). If default transport is used, chat tries to automatically reconnect and fires session.onContinued when the connection is restored.

Callbacks receive no event object.

#### Important

This event duplicates the session.onError event with the "150" (network interruption) Error code. Genesys recommends that you use session.onInterrupted/onContinued if you want to track the connection status. The "150" error code is deprecated and might be removed in future versions.

## session.onContinued

Executed only after session.onInterrupted when the connection to the server is restored.

Callbacks receive no event object.

## session.onSessionEnded

Executed when the client drops out of a chat session due to one of the following reasons:

- Logout request.
- Chat session is finished from the agent's side.
- Chat Server stops responding, resulting in a timeout during which the session is not restored.

#### **Parameters**

| Parameter                | Description  |
|--------------------------|--|
| event.reason.error       | One of these is set to true depending on the reason      |
| OR<br>event.reason.agent | • <b>error</b> — a major error occurs (for example, Chat |

| Parameter                    | Description   |
|------------------------------|---|
| OR event.reason.leaveRequest | <ul> <li>Server stops responding)</li> <li>agent — agent ends the session;</li> <li>leaveRequest — visitor sends a session.leave command and it was successful</li> </ul> |

## Miscellaneous methods

#### session.isAgentConnected

Provides a convenient way to synchronously determine if at least one agent is present in the session.

Returns boolean (true or false).