API Reference

startChat(options)

5/1/2025

startChat(options)

Description

This is the main entry point for configuring and starting a chat session.

Returned Promise

`startChat` returns a "promise" object with two chainable methods: `done` and `fail`.

> **Important**
> Currently, the promise is resolved or rejected only if the chat is started in "embedded" mode. If it is started in separate window ("popup" mode), you have to use promise returned by `startChatInThisWindow` method to get access to the Chat Session API.

done

Use this method to get access to the chat session service API as resolved with an instance of the session object.

```
chat.startChat(options).done(function(session) {
  // session.sendMessage, session.onAgentConnected and all other method are at your disposal.
});
```

> **Tip**
> See Chat session commands for Chat Session API documentation . Note that if you need to access the Chat Session API, you will probably want to get access not only in cases when the session is started, but also when it is restored. You can use `restoreChat`'s "done" callback for that. See Getting Access to Chat Session API for more info.

> **Important**
> The promise is never resolved before the chat session is created. This means that if registration is enabled, the **done** callback will not fire until the registration is complete and processed by the server.

fail

Resolved with an event containing an error describing what went wrong.

Event structure

Parameter	Meaning
event.error.code	Code of error
event.error.descpription	Description of the error (English is default language).

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

Chat widget error codes

In addition to the regular list of possible errors, the Chat Widget adds some of its own errors:

Error code	Error description
Chat widget-specific error codes (range 200 -249)	
200	Chat is already running on this page.
201	Chat is already running on another page.

For example,

```
chat.startChat(options).fail(function(event) {
    if (event.error === 201) {
```

```
        alert('Chat is already running on another page');
    }
});
```

Options

serverUrl

Type	Default Value	Mandatory	Description
string	undefined	Yes (except when transport options is provided)	URL of the CometD chat server for default (built-in) CometD transport.

embedded

Type	Default Value	Mandatory	Description
boolean	false	No	Sets chat mode of operation: "embedded" (chat widget is rendered directly on a page) or "popup" (chat opens in a separate browser window). Default is "popup". Pass the value `true` to switch to "embedded" mode.

> **Important**
>
> If chat is configured for `embedded` mode, the chat widget will disappear as soon as the user leaves the website or navigates to any non-instrumented web page. If the user returns to the page before the timeout has expired, chat will automatically be restored. To configure the timeout, use the maxOfflineDuration option. The `maxOfflineDuration` options may be pre-configured when you use chat as part of a particular solution such as the Integrated JavaScript Applicaiton

transport

Type	Default Value	Mandatory	Description
Object	undefined	No (except when serverUrl is omitted)	Custom transport instance (for example, REST-based).

localization

Type	Default Value	Mandatory	Description
Object, string, or function(function?)	undefined	No	Provider of customer localization. This value can be one of the following: • JavaScript object with localization data. Added in **850.0.0**. • Function that returns an object with localization data. Added in **850.4.0**. • Function that accepts a callback and calls that callback with an object containing localization data. Added in **850.4.0**. • URL of and external JSON file with the localization datav If omitted, default English localization will be used. See Localization for more on how to localize the chat widget. Important If you use Template-based Customization, all localization data is available inside your templates as a `data.nls` object.

templates

Type	Default Value	Mandatory	Description
string	none	No	The URL of the HTML file containing templates used to render the chat widget. The request is made using either JSONP or AJAX, following the same logic for localization files (see Localization). Default templates are included into the JavaScript source so by default, there are no requests made to load them. For more information, see Template-based Customization. Important Since chat version **850.5.0** you do not necessarily have to download the custom templates over the network. See Template-based Customization for more.

widgetUrl

Type	Default Value	Mandatory	Description
string	undefined	No (except when embedded is set to `false` - "popup" mode)	URL of chat widget html that will be open in external window when operating in "popup" mode.

windowSize

Type	Default Value	Mandatory	Description
Object {width: number, height: number}	{ width: 400, height: 500 }	No	Size of external chat window when operating in "popup" mode. Important Note that `windowOptions` can override `windowSize`.

windowName

Type	Default Value	Mandatory	Description
string	`genesysChatWindow`	No	A string representing the new window that is passed to the `window.open` call when opening the chat widget window. **Important** `windowName` does **not** specify the title of the new window. **Tip** This option only works in "popup" mode (`embedded` is either absent or set to `false`) For example, `chat.startChat({ windowName: 'myWindowName' //... }); // => window.open(<widgetUrl>, 'myWindowName', ...);`

windowOptions

Type	Default Value	Mandatory	Description

Object	value of `windowSize` option	No	An object containing the options that will be passed `window.open` when opening a chat widget window.
			
			Important
			This option only works in "popup" mode (`embedded` is either absent or set to `false`)
			
			Use this object to pass any window options, such as position (`top, left`), whether to show browswer buttons (`toolbar`), location bar (`location`), and so on. See Window.open for the full list. All options are converted to a string that is passed to `window.open` call. For example,
			
			`chat.startChat({ // open chat widget in top left corner of the screen windowOptions: { left: 0, top: 0 }, // ... }); // => window.open(<widgetUrl>, <windowName>, 'left=0,top=0,...')`
			
			Note that `windowOptions` is merged with `windowSize`, but has higher priority. For example,
			
			`chat.startChat({ windowSize: { width: 200, height: 400 }, windowOptions: {`

Type	Default Value	Mandatory	Description
			left: 0, top: 0, width: 300 // this value will be used, and height will be taken from windowSize }, // ... }); // => window.open(<widgetUrl>, <windowName>, 'left=0,top=0,width=300,height=400')

debug

Type	Default Value	Mandatory	Description
boolean	false	No	Pass the value `true` to enable chat debugging logs (by default standard `console.log` is used, see the logger option if you want to override that).

logger

Pass a function that will be used for chat logging (if debug is set to `true`) instead of the default `console.log`. The function has to support the interface of the `console.log` — it must accept an arbitrary number of arguments and argument types.

> **Important**
>
> To use the custom logging function in a separate window, you have to pass it directly on the widget page to the `startChatInThisWindow` method.

registration

Type	Default Value	Mandatory	Description
(boolean|function)	true	No	By default chat starts with a built-in registration form (that you can customize using `ui.onBeforeRegistration`). Pass the value `false` to disable this default built-in registration form.

Custom registration

Pass a function to customize registration workflow.

The function accepts one argument: a done function that must be called with an object containing the data collected during registration.

This may sound complex but actually it is pretty straightforward:

```
chat.startChat({
  registration: function(done) {
    done({
      EmailAddress: 'john.doe@example.com'
    });
  }
  //...
});
```

In the example above, it is assumed that all data is known beforehand and so registration may be completed synchronously and in a way that hides the operation from the end user. However, in reality you may want to send an additional request to obtain the necessary data:

```
chat.startChat({
  registration: function(done) {
    // Suppose you have a special URL that returns current user's credentials that you want
to pass to chat session:
    jQuery.get('/account/credentials', function(data) {
      done(data);
    });
  }
  //...
});
```

You may have noticed that both of these examples are artificial, in the sense that they do not provide any UI but simply silently register the user with already available data. To provide a registration UI, you will have to return the DOM object representing your UI from your custom registration function. For example, like this:

```
chat.startChat({
  // Simple jQuery-based example
  registration: function(done) {
    var $form = $('<form />'),
        $email = $('<input type="email" name="email" placeholder="Enter your Email" />');

    // bind done function to be called when the form is submitted
    $form.on('submit', function() {
      done({
        EmailAddress: $email.val()
      }};
    });

    $email.appendTo($form);

    // return form DOM representation: it will be displayed in the chat widget
    return $form.get(0);
  }
  //...
});
```

Although the example above may seem complicated, this approach is very powerful as it allows you to reuse any JavaScript stack you use on your site, be it jQuery, client-side templating, more full-featured frameworks like AngularJS or any other JS-based technology or their combinations.

userData

Can be used to directly attach necessary UserData to a chat session. See Custom Registration and Extended API to support integrated solutions for other ways of working with UserData.

```
chatAPI.startChat({
    userData: {
        visitID: currentVisitID
    }
});
```

createContact

Type	Default Value	Mandatory	Description

boolean	true	No	Determines whether new contact should be created from registration data if it doesn't match any existing contact. Only effective if registration data is present (collected either by built-in or custom registration workflow).
			
			`// createContact is not effective. Contact will not be created.`
			`chat.startChat({`
			`  registration: false`
			`  // ...`
			`});`
			
			`// Contact will be identified and if doesn't exist, it will be created.`
			`chat.startChat({`
			`  createContact: true,`
			`  // ...`
			`});`
			
			`// Contact will be identified; if doesn't exist, it won't be created.`
			`chat.startChat({`
			`  createContact: false,`
			`  // ...`
			`});`
			
			Tip
			Technically, this option controls the lookup (and possibly the creation) of client's record in the UCS database. By default,

Type	Default Value	Mandatory	Description
			• If registration is disabled (or provides no data, which can be the case with custom registration function), the contact **will not be looked up** in UCS; • If registration is enabled and provides some data (in fact any data, since we do not validate on the browser side), the contact **will be looked up** in UCS and if not found, it **will be created**. • However, if `createContact` is set to `false`, the contact **will be looked up** in UCS and if not found **it will not be created**.

ui

Type	Default Value	Mandatory	Description
(boolean|Object)	true	No	Pass the value `false` to disable the chat widget UI completely. Or pass an object with "hook" functions that can modify the built-in UI. All "hooks" receive an object: a DOM representation of a particular UI part, which you can then modify. The structure of these objects can be determined using browser developer tools (F12 in Chrome) and may be modified in future versions of chat widget. **Important** Backward compatibility of the DOM structure provided in the UI hooks is NOT guaranteed. `function makeEverythingRed(htmlElement) { jQuery(htmlElement).find('*').css({ color: 'red' }); } function makeEverythingBold(htmlElement) { jQuery(htmlElement).find('*').css({ 'font-weight': 'bold' }); } chat.startChat({ ui: { onBeforeChat: makeEverythingRed, onBeforeRegistration: makeEverythingRed, onBeforeMessage: makeEverythingBold`

Type	Default Value	Mandatory	Description
			<pre> }, // ... });</pre> Tip See more examples in the Customizing the User Interface section.

Available "hooks" are:

Hooks	Description
onBeforeChat	Sent before the "main" chat UI is rendered (messages area and message input field). Can be used to modify the layout / functionality of the chat widget.
onBeforeRegistration	Sent before the registration form is rendered (and only if it is enabled).
onBeforeMessage	Sent before every message that gets appended to the chat. For example, messages from user, agent, system, typing, and so on are all included. This hook has a special ability of "filtering out" certain messages in the chat. If the function attached to it explicitly returns `false`, that particular message is not added to the UI. This is useful for passing internal or system data using the the chat channel. To ease message analysis, the hook function receives a second argument: the message text. Tip For example usage, see: • Using the ui.onBeforeMessage Hook to Add Desktop Modifications

maxOfflineDuration

Type	Default Value	Mandatory	Description
number	5	No	Time (in seconds) during which state cookies are stored after page reload/navigation. If cookies expire, the chat is not restored. This option specifies how long the chat session will live after the user leaves the website. A value of five seconds is usually long enough for chat to survive page reloads, but the chat interaction is likely to be lost if the user navigates to another site during the chat interaction (in *embedded* mode) and comes back to your site. To support this kind of workflow, consider increasing the value of this option or use the Integrated JavaScript Application, where the option value is increased to 600 seconds by default.

disableWebSockets

Type	Default Value	Mandatory	Description
boolean	false	no	You may disable WebSockets for chat by passing `true` to this option. By default, chat attempts to use WebSockets for connections to the server. When WebSocket connections are unavailable ,such as when the load balancer does not support WebSockets, chat switches to other HTTP-based means of

Type	Default Value	Mandatory	Description
			communication. Disabling WebSockets can speed up the time it takes for chat to switch to another means of communication. Important If you choose to disable WebSockets, you should also pass this option to restoreChat(options). This option is only effective with the default (built-in) transport.