API Reference

Co-browse API

5/8/2025

Co-browse API

Contents

This API provides methods and callbacks to work with Co-browse and can be used to implement a custom UI for co-browsing.

> **Important**
>
> See Accessing the Co-browse APIs for information on accessing this API.

Co-browse in iframes

Some Co-browse UI elements such as the the co-browsing button and toolbar should not appear when Co-browse is in an iframe. Common Co-browse UI elements such as notifications that an element is masked should appear whether or no Co-browse is in an iframe. As such, there are two contexts for the Co-browse JavaScript API:

- Top context, available when Co-browse is not rendered in an iframe.
- Child context, used when a page is rendered in an iframe. For the child context, a subset of the top context API is available.

isTopContext

The `isTopContext` variable can be used determine which context Co-browse is rendered in. `isTopContext` is passed to the onReady method and equals `true` if Co-browse is rendered in the top context and `false` otherwise.

Example:

```
var _genesys = {
    cobrowse: {
        onReady: function(api, isTopContext) {
            // common functionality
            api.onMaskedElement.add(function() {/* deal with masked elements here*/});
            if (!isTopContext) {
                return;
            }
            // top context functionality goes below
    }
};
```

> **Tip**
>
> See Accessing the Co-browse APIs if you are unfamiliar with the onReady syntax above.

Signals and Callbacks

The Co-browse API exposes a number of **signals** in both the top and child contexts. Each signal is object with the three following methods:

- `add(function)`—adds a callback
- `addOnce(function)`—adds a callback that will be executed only once
- `remove(function)`—removes a callback

The naming convention for signal names begins with "on" and follows the format **onSomethingHappened**.

> **Important**
>
> **Signals** act similar to **deferred** objects. If you add a callback to an event that has already happened, the callback will be called immediately. For example, if you add a callback to the `onAgentJoined` signal when the event has already happened, the callback will be called immediately.

Session Object

Many callbacks receive a `session` object as an argument. This object has the following properties:

- `token`—String containing the session token shared with the agent and possibly shown in the UI. The token is a 9 digit string such as "535176834".
- `agents`—Array of connected agents. Each element in the array is an object with no properties.

Common API

The following elements and properties are available from both the top and child Co-browse contexts:

VERSION

String containing current JS version. For example, `8.5.000.90`.

```
console.log(_genesys.cobrowse.VERSION);
```

> **Tip**
>
> - Available since Genesys Co-browse 8.5.

- The JavaScript version does not necessarily match the product or server version.

onSessionStarted

This signal is dispatched when a Co-browse session is successfully started and joined by the customer such as when the `Co-browse` button is pressed or when `startSession()` is called.

Arguments:

- `session`—Session `object` representing the ongoing session.

Example:

```
function notifyCobrowseStarted(session) {
    alert('Co-browse has started. Spell this session token to our representative: ' +
session.token);
}
cobrowseApi.onSessionStarted.add(notifyCobrowseStarted);
```

Important

Starting in release 9.0.005.15, `onSessionStarted` is available from both the main page context and from nested iframes contexts. Handlers for these signals will be executed in all existing contexts where handlers are present. To call handlers only in top context, as in earlier Co-browse releases, use the isTopContext JavaScript API method to distinguish top and child co-browse contexts.

onSessionEnded

This signal is dispatched when a Co-browse session ends.

Arguments:

- `details`—Object with the follwing field:
  - `reason`—Field with value of a string or undefined. Possible string values:
    - `self`—The user has exited the session by clicking the `Exit` button or calling the `exitSession()` API method.
    - `external`—The agent has closed the session. Some server errors may also result in this value.
    - `timeout`—The session has timed out such as when a user reopens a page with an expired Co-browse cookie.
    - `intactivityTimeout`—The agent did not join a session in the configured amount of time.
    - `serverUnavailable`—The Co-browse server was unreachable. Added in Genesys Co-browse release 8.5.001.xx.

- `sessionsOverLimit`—Agent is busy with another co-browse session and is prohibited from starting another session at the same time, see One-Session Agent Limitation. Added in Genesys Co-browse release 8.5.003.04.
- `error`—There is an error such as a critical misconfiguration.

Example:

```
var cbEndedMessages = {
    self: 'You exited Co-browse session. Co-browse terminated',
    external: 'Co-browse session ended',
    timeout: 'Co-browse session timed out',
    inactivityTimeout: 'Agent did not join. Closing Co-browse session.',
    serverUnavailable: 'Could not reach Co-browse server',
    sessionsOverLimit: 'Agent is busy in another Co-browse session'
}
cobrowseApi.onSessionEnded.add(function(details) {
    alert(cbEndedMessages[details.reason] || 'Something went wrong. Co-browse terminated.');
    showCobrowseButton();
});
```

> **Important**
>
> Starting in release 9.0.005.15, `onSessionEnded` is available from both the main page context and from nested iframes contexts. Handlers for these signals will be executed in all existing contexts where handlers are present. To call handlers only in top context, as in earlier Co-browse releases, use the isTopContext JavaScript API method to distinguish top and child co-browse contexts.

markServiceElement(element)

Service elements do not show up in the agent's view. This function is used to mark service elements in a custom Co-browse UI.

Arguments:

- `element`—HTML element that will be masked.

> **Important**
>
> Elements must be marked as **service** elements **before** the Co-browse session begins. If the Co-browse session has already started, **service** elements should be marked before they are added to the DOM. It is also possible to mark elements as **service** without using this function. Doing so is useful for static HTML content. To do so, add an attribute `data-gcb-service-node` with value `true`. This available since version 8.5.001.20. Use `_genesys.cobrowse.VERSION` to check the version.

> **Important**
>
> The `markServiceElement()` method should not be used to hide sensitive information. Business functions like DOM Control or Data Masking should be used for sensitive content such as private user data.

Plain DOM Example:

```
function createCustomCobrowseUI(cobrowseApi) {
    var toolbar = document.createElement('div');
    toolbar.className = 'cobrowseToolbar';
    toolbar.textContent = 'Co-browse is going on';
    cobrowseApi.markServiceElement(toolbar); // don't show the toolbar to agents
    cobrowseApi.onConnected.add(function() {
        document.body.appendChild(toolbar);
    })
}
```

jQuery Example:

```
// Create a simple jQuery plugin
$.fn.cbMarkNode = function() {
    return this.each(function() {
        cobrowseApi.markServiceElement(this);
    });
};

// And then:
$('<div class="cobrowseToolbar">Co-browse is going on</div>').cbMarkNode().appendTo('body');
```

Static content example, without JS API usage:

```
<div id="myChatWidget" data-gcb-service-node="true">...</div>
```

onMaskedElement

This signal is dispatched when Co-browse encounters an element that is subject to data masking.

Arguments:

- `element`—HTML Element

This signal is dispatched multiple times when Co-browse initiates and can be dispatched again if a masked element is added to the page dynamically.

Example:

```
cobrowseApi.onMaskedElement.add(function(element) {
    $(element).tooltip({
        content: 'Content of this elements is masked for representatives.'
    });
});
```

Consider a scenario where you have the following HTML elements on your **example.html** page:

```
<div class="vcard">
    <p class="fn"><a class="url" href="#">Dr. John Doe</a><p>
    <p class="adr">
        <span class="street-address">Imaginary Hospital</span><br>
        <span class="region">Doctorville</span><br>
        <span class="postal-code">742617</span><br>
        <span class="country-name">Great Britain</span>
    </p>
    <p class="tel">+44 (0)1234 567890</p>
</div>
```

And you also have the following data masking configuration:

```
<?xml version="1.0" encoding='UTF-8' ?>
<domRestrictions>
    <restrictionsSet>
        <uriTemplate type="regexp" pattern="example\.html"/>
        <dataMasking>
            <element selector=".adr"/>
            <element selector=".tel"/>
        </dataMasking>
    </restrictionsSet>
</domRestrictions>
```

In this scenario, the callback is called two times when Co-browse is initiated, and then each time you dynamically add an `.addr` or `.tel` element to the page.

Top Context API

The following methods and properties are available only when Co-browse is rendered in the **top** context.

isBrowserSupported()

> **Important**
>
> For a list of officially supported browsers see Browser Support. This method checks for the presence of required browser APIs and may return `true` for browsers not officially supported.

This method checks for the presence of MutationObserver and a few other required APIs, not for browser type and version. It returns a boolean with the value of `true` when the browser supports required APIs and `false` otherwise. The built-in integration module uses this function to show a message if a user tries to start Co-browse in an unsupported browser. You may use it, for example, to hide the Co-browse button completely.

startSession()

This method instantiates a new Co-browse session. It will throw an error if the browser is not supported.

exitSession()

This method exits and ends an ongoing Co-browse session.

downgradeMode()

This method immediately switches the current session from Write Mode to Pointer Mode. The built-in Co-browse UI executes this method when an end user clicks "Revoke Control" while in Write Mode.

See related signals: onModeUpgradeRequested and onModeChanged.

onInitialized

This signal is dispatched after the page is loaded and the Co-browse business logic is initialized.

Arguments:

- `session`— Session `object` representing the ongoing session or `null` if there is no ongoing session.

Example:

```
cobrowseApi.onInitialized.add(function(session) {
    if (!session) {
        showCobrowseButton();
    } else {
        showCobrowseToolbar(session);
    }
})
```

onAgentJoined

This signal is dispatched when an agent successfully joins a session.

Arguments:

- `agent`—Object representing the new agent. This object has no properties.
- `session`—Session `object` representing the ongoing session.

Example:

```
cobrowseApi.onAgentJoined.add(function(agent, session) {
    alert('Representative has joined the session');
});
```

onAgentNavigated

This signal is dispatched when the **agent** user initiates navigation such as refresh, back, forward, or when the agent enters a URL into the agent Co-browse UI. Signal is dispatched a few seconds before the navigation happens. This can be used to, for example, send a warning to the user or disable the `Exit session` button before navigation.

Arguments:

- `details`—Object containing the following navigation detail fields:
  - `command`—String with the value of `back`, `refresh`, `forward`, or `url`.
  - `url`—Optional string that is present only if the `command` field has the value of `url`

Example:

```
// Let's assume we have a "growl" function available to show growl-like notifications
cobrowseApi.onAgentNavigated.add(function(details) {
    if (details.url) {
        growl('Representative navigated to the page: ' + details.url);
    } else {
        growl('Representative has pressed the "' + details.command + '" button. Page will be
refreshed');
    }
});
```

onNavigationFailed

This signal is dispatched when the navigation request from the agent fails to execute such as when the agent navigates forward when there is no forward history. You can use this signal to to re-enable the `Exit` button and/or show a notification.

The callback receives no arguments.

Example:

```
// Let's assume we have a "growl" function available to show growl-like notifications
cobrowseApi.onNavigationFailed.add(function() {
    growl('Navigation request by representative has failed');
});
```

onModeUpgradeRequested

This signal is dispatched when an agent requests upgrading the Co-browse session to Write Mode.

Arguments:

- done—The function passed by the Co-browse code. Call it with `true` to allow the transition to Write Mode, or with `false` to prohibit.

Example:

```
cobrowseApi.onModeUpgradeRequested.add(function(done) {
  if (confirm('Representative requests control over the web page. Allow?') {
    done(true); // allow upgrading to Write Mode
  } else {
    done(false); // disallow and stay in Pointer Mode
  }
});
```

Note: If you're going to implement something similar to the example above, don't forget to disable the built-in UI.

onModeChanged

This signal is dispatched when the Co-browse session Mode changes, either to Pointer or Write.

Arguments:

- `mode`—An object with two boolean properties:
  - `pointer`—This is `true` if the session has switched from Write to Pointer Mode. Otherwise, it's `false`.
  - `write`—This is `true` when the session has switched from Pointer to Write Mode.

Example:

```
cobrowseApi.onModeChanged.add(function(mode) {
    if (mode.write) {
        alert("Representative has now control over the page");
    } else if (mode.pointer) {
        alert("Representative can no longer control the page").
    }
});
```