Co-Browse Administrator's Guide

11/3/2024

Table of Contents

Search the table of all articles in this guide, listed in alphabetical order, to find the article you need.

Related documentation:

-

Getting Started

Contents

- Administrator
- Developer

Co-browse lets an agent see exactly what is happening on your customer's screen.

Related documentation:

-

Sometimes your customers need help when they are browsing your website. Maybe they can’t tell exactly where to click—or perhaps they need help filling out a complex form. Genesys Co-browse lets your agents use Agent Workspace to do some of the driving for them, by showing them what the customer sees in their browser window (not the whole screen) and allowing the customer to give them control of the web page.

Here are some of the main features of Genesys Co-browse:

- The agent and the customer can browse and navigate the same web page, at the same time.
- Browsing always happens on the customer side, and both the agent and the customer can take control of the session.
- Co-browse sessions begin in Pointer Mode where the agent cannot enter information or navigate for the customer.
- The agent can send the customer a request to enter Write Mode where the agent can enter information for the customer.
- Sensitive data can be hidden and control of elements (buttons, check boxes, and so on) can be restricted.

You can find some more details about Co-browse sessions here.

Getting started

There are a few steps to take in order to get up and running with Genesys Co-browse. Here are the details:

What Genesys does for you

We configure Co-browse for you, based on the following information that you provided to us:

- Allowed origins (your domains and sites from which Co-browse can start)
- Allowed external domains (your domains where static resources for the site are stored)

What you need to do

DOM restrictions

You must configure your DOM restrictions for your website. The Genesys Co-browse DOM Restrictions Editor streamlines this process for you.

Genesys Widgets

You must install and configure Genesys Widgets on your website, in order for the end customer to start a co-browse session with an agent (associated with a chat or a voice session). Everything you need to know about Genesys Widgets is in the following pages:

- Genesys Widgets deployment guide

Here are some examples of Genesys Co-browse enabled in different Genesys Widgets.

Using Co-browse without Genesys Widgets

While Genesys Widgets is the preferred and recommended way to add Co-browse to your website, you may have some scenarios where Genesys Widgets cannot be used. In this case, see the Genesys Co-browse JavaScript API for details on how to deploy Co-browse to your website without using Genesys Widgets. This is considered advanced functionality.

Genesys Co-browse DOM Restrictions Editor

- Administrator
- Developer

You can hide sensitive customer data from agents and restrict control of elements in a co-browse session.

Related documentation:

-

The Genesys Co-browse DOM (Document Object Model) Restrictions Editor makes it easier for you to hide sensitive customer data from agents and restrict control of elements in a co-browse session. The Editor handles most of the configuration complexity behind the scenes and allows you to focus on your restrictions. You can implement two types:

- **DOM control**—the agent sees the content but won't be able to interact with it. For example, clicking a button or a link won't work for the agent, even when Co-browse is in Write Mode. The agent will see a green border surrounding content with DOM control.
- **Data masking**—the agent sees masked content as asterisks (******) instead of characters, and masked images will be grayed out. The agent will see a purple border surrounding masked characters and images.

> **Important**
>
> Since masked content is also DOM controlled (non-interactive), the DOM Restrictions Editor shows Data Masking as DOM Control and Data Masking.

With the DOM Restrictions Editor, you can easily

- create, edit, or delete a restriction.
- have an optional description.
- view a list of existing restrictions.
- logically group your restrictions.
- apply a restriction to all web pages, or to the page or set of pages that match the regular expression.

- view all restrictions on the current web page.

To allow users to log into the DOM Restrictions Editor, add them to the **Administrators** Access Group.

Link to video

Watch the video tutorial on how to use the DOM Restrictions Editor.

Genesys Co-browse sessions

Contents

- Administrator
- Developer

Co-browse sessions are the interactions between customers and agents, where privacy and security are priorities.

Related documentation:

-

A session is initiated when a customer requests to co-browse. The session stays idle until the agent joins, then the session is considered to be active. The session ends when one of the parties (the customer or the agent) exits. It is not possible to re-join a co-browse session. If one party exits accidentally, a new session must be initiated. An agent is limited to handling one co-browse session at a time.

Session identifiers

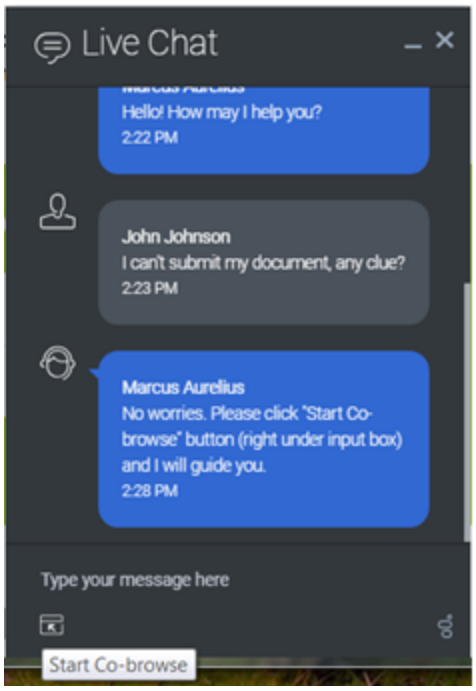
Each live session has an identifier that can be used to track the session. This session ID is a sequence of nine digits that is applicable only to live sessions.

Starting a session

A co-browse session can only be initiated by a customer. An agent does not have the option or ability to send a co-browse request to a customer. This provides greater security to the customer. In order to initiate a co-browse session, the customer must already be engaged in an interaction with an agent, be it a voice call or a chat.

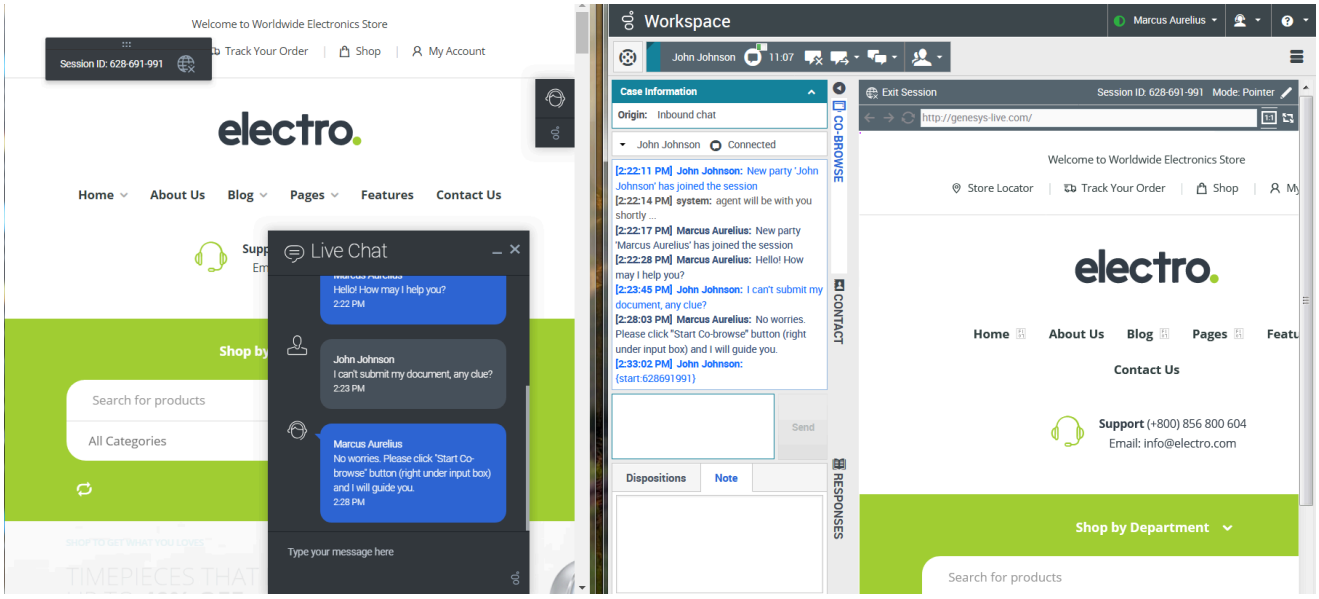
When the session is established, the agent's browser displays a view of the customer's browser. This view that the agent sees is loaded from Genesys Co-browse. The agent is not a client of the website. All actions taken by the agent are passed onto and "replayed" on the customer's side.

Starting a co-browse session from Genesys Widgets

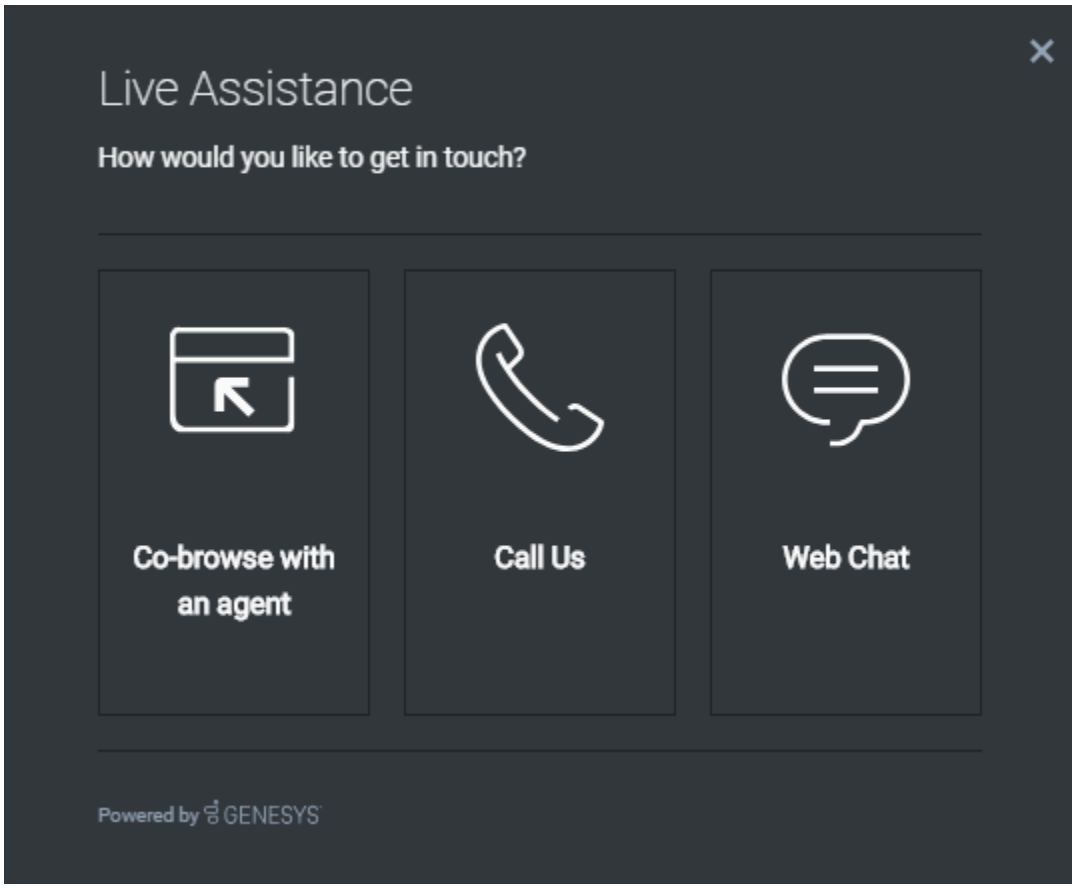


A customer initiates a co-browse session through a Genesys Widget integrated into the website. You can enable Genesys Co-browse in several Genesys Widgets.

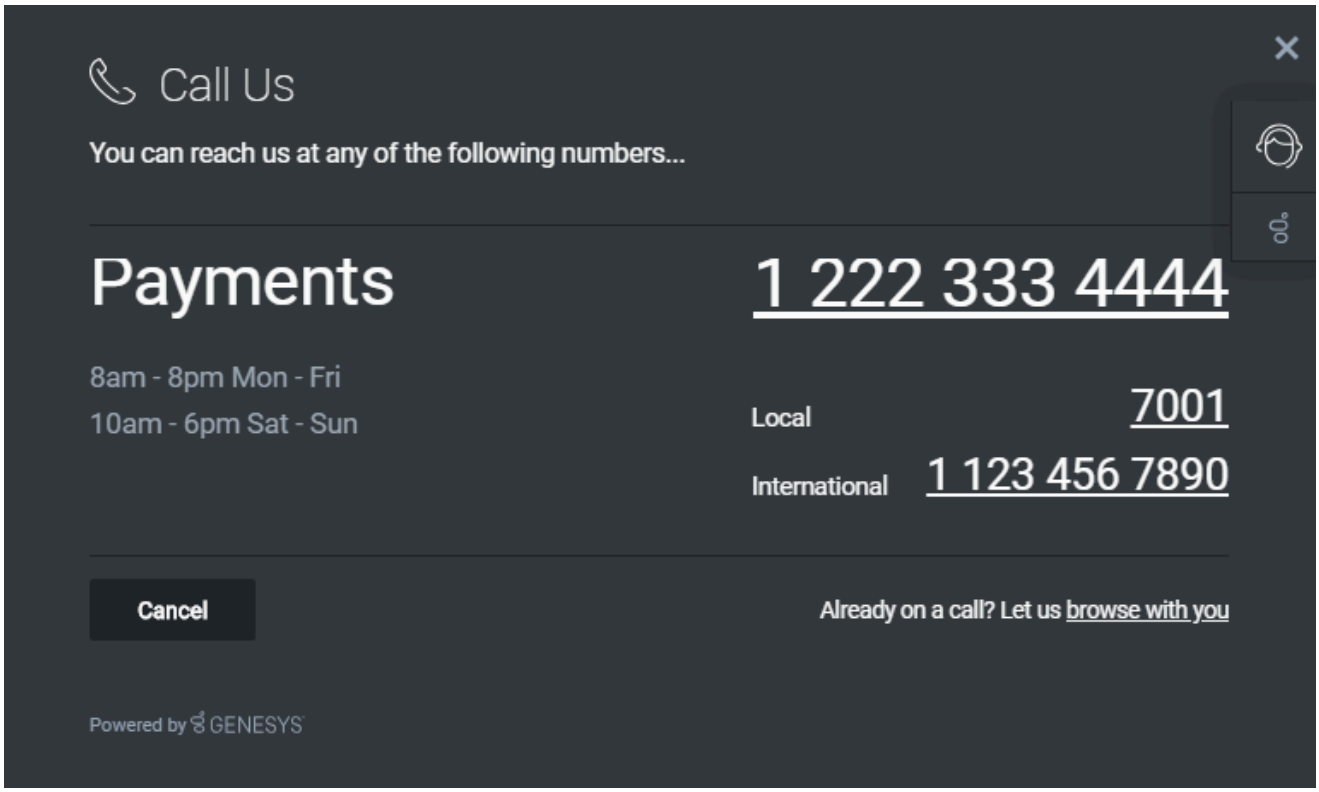
Here is an example of starting a co-browse session from the WebChat Widget.



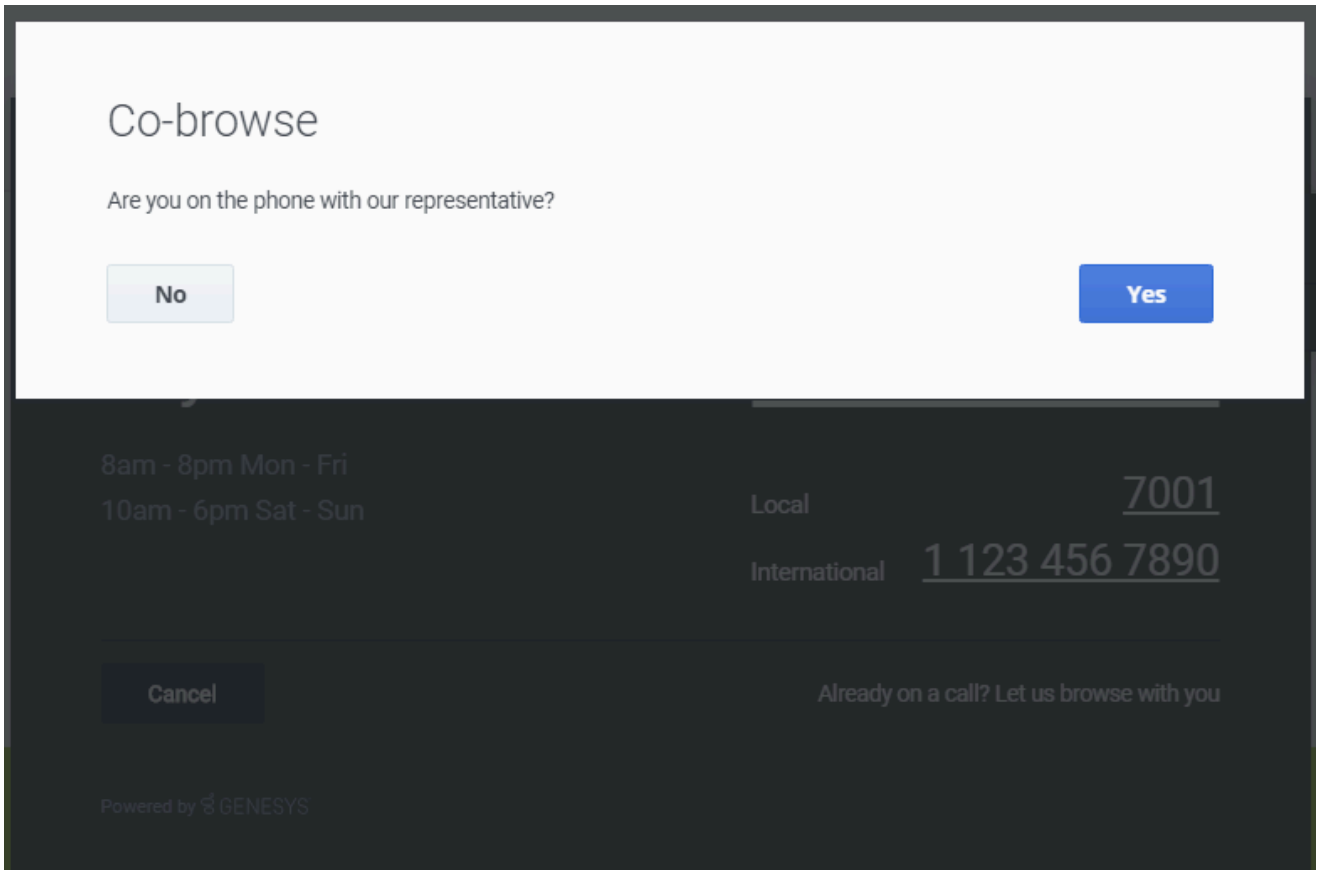
This example shows a customer and agent view of a co-browse session started from the WebChat Widget.



And here is an example of starting a co-browse session from the ChannelSelector Widget.

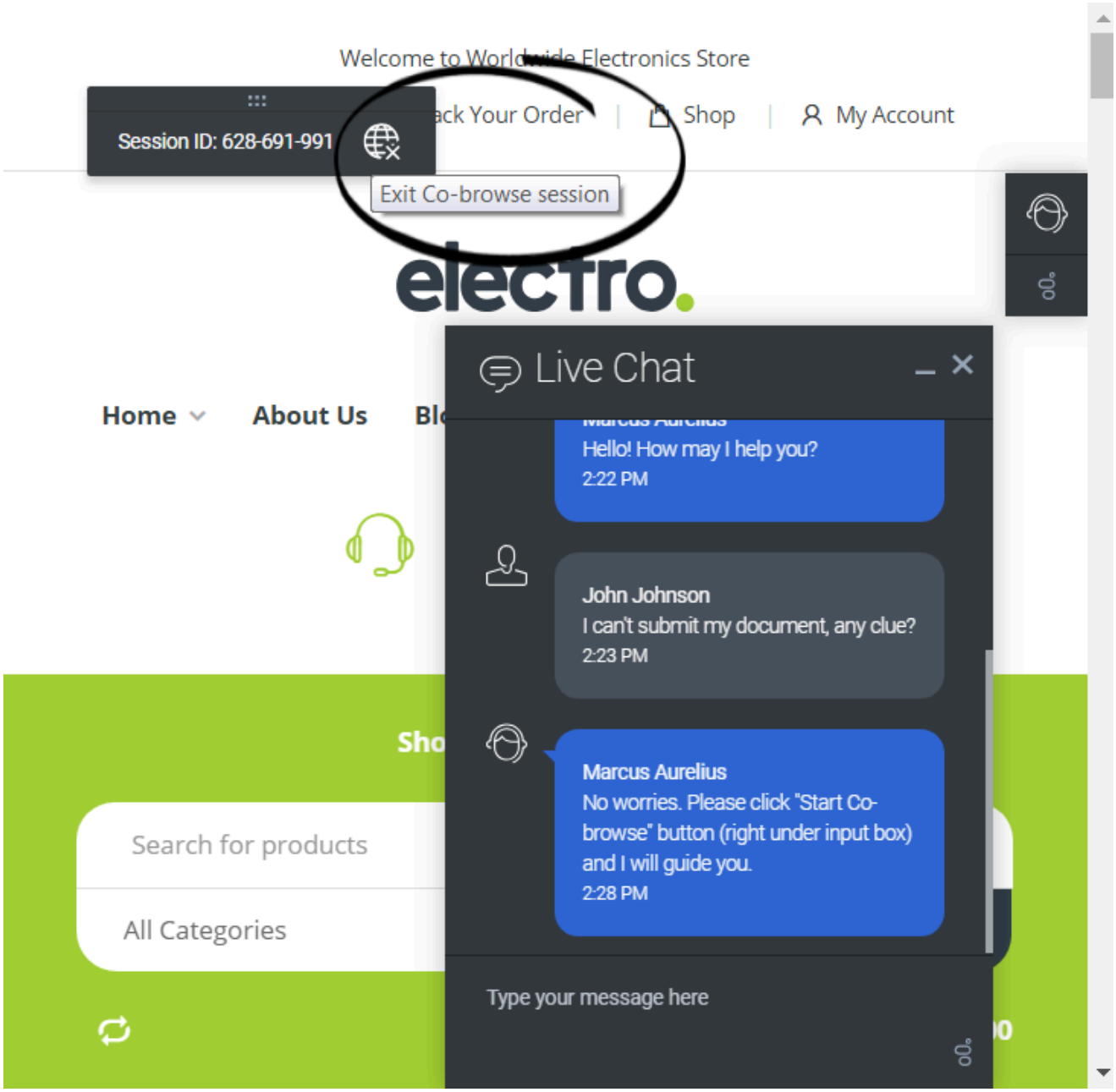


This example shows starting a co-browse session from the CallUs Widget with the **browse with you** link as the co-browse option.



After clicking **browse with you** in the CallUs Widget the customer will see this dialog.

Stopping a co-browse session from Genesys Widgets



Once a co-browse session has been established, both parties have the ability to terminate the session. At any time, either party may click the **Exit Co-browse session** icon next to the session ID. The agent can also exit by clicking **Exit Session** in Agent Workspace.

The other party will be notified that the session has ended, and the agent's browser will no longer display a view of the customer's browser. Also, if the primary interaction (chat or voice call) is terminated, the co-browse session terminates automatically. Sessions can also terminate due to

inactivity, after a pre-configured timeout expires. Likewise, if the agent closes their browser, or navigates to a third-party website, the session will terminate if the agent does not return to the session page within the pre-configured timeout.

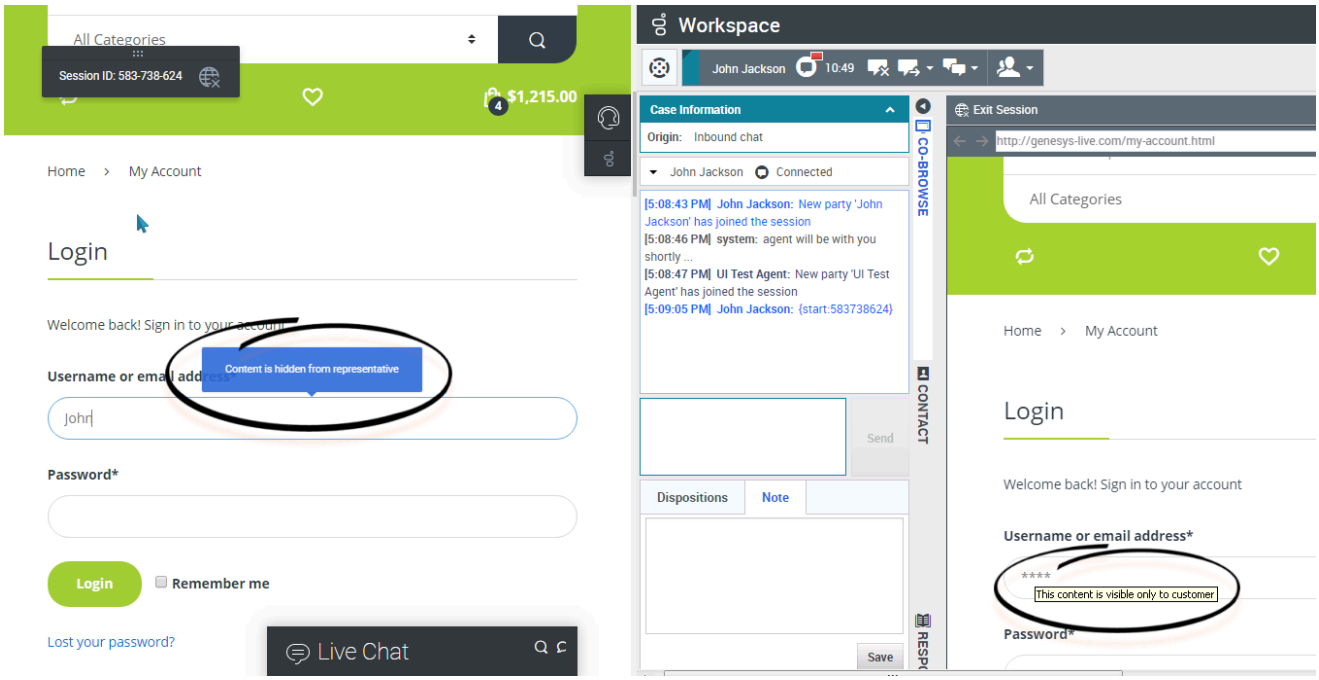
Once a session has been terminated, it cannot be reactivated. If the session was deactivated accidentally, a new session has to be initiated, with a new session identifier.

Participating in a co-browse session

Once a co-browse session begins, the agent can use his or her mouse pointer to guide the customer through the web site. Agents start co-browse sessions in Pointer Mode. In Pointer Mode, the customer and the agent can see each other's mouse pointer but the agent can not enter any information into the web page, click buttons, or navigate the customer's browser. If the agent needs to enter information into the web page or to navigate the browser, he or she can send the customer a request to switch the co-browse session to Write Mode.

All actions (mouse clicks, key presses, and so on) are actually performed on the customer side. Any actions taken by the agent are sent to the customer's browser. This ensures a secure approach, as all browsing is done on one side—the customer's side. This approach also provides for greater performance and a more seamless customer experience. Each participant can see the other participant's mouse movements as well. This enables an agent to point to specific sections on the web page to help direct the customer through their task.

Restricting visibility of sensitive data



You can limit which fields are visible to and editable by the agent and which elements are controlled by agents. This configuration task is made much easier for you by using the Co-browse DOM Restrictions Editor.

Some fields can have the data masked. For example, you might choose to hide the customer's user name, email address, password, Social Security information, and so on, from the agent. The end user can easily identify which information is hidden (data masked) from the agent. By default, all passwords are masked.

At the same time, control for some elements can be disabled. By default, all **Submit** buttons are deactivated for the agent. If he or she clicks on a **Submit** button, nothing happens. The customer always has permission to submit any web forms, just as they would while browsing normally.

Genesys Co-browse Localization

- Developer

You can localize the end-user Co-browse UI using the built-in German or French localization options.

Related documentation:

-

To localize the end-user Co-browse UI, configure the **localization** option in the global configuration object in **`window._genesys.widgets.cobrowse.`**

> **Important**
> The Co-browse agent-facing UI inherits localization from the Agent Workspace locale settings.

To set the end-user Co-browse UI language to German or French, add **de** or **fr** to the instrumentation:

Example

```
window._genesys.widgets.cobrowse = {

  src: APIGEE_GCB_URL+'/cobrowse/js/gcb.min.js?apikey=APIGEE_GCB_KEY',
  url: APIGEE_GCB_URL+'/cobrowse/',
  apikey: 'APIGEE_GCB_KEY'

lang: 'de'
};
```

If both **lang** and **localization** are provided, **localization** takes priority. This way it is possible to use built-in German or French localization and override some fields as necessary.

Example

```
window._genesys.widgets.cobrowse = {
  ...
  lang: 'de',
  localization: {
     "modalYes": "Natürlich!"
  }
```

```
};
```

If you are using Co-browse outside of Genesys Widgets, use **window._genesys.cobrowse.localization** instead of **window._genesys.cobrowse.widgets.localization**.

Example

```
if(!window._genesys)window._genesys = {};

window._genesys.widgets = {
  cobrowse: {
    src: 'https://www.website.com/cobrowse/js/gcb.min.js',
    url: 'https://www.website.com/cobrowse/',
    localization: {
      // Here we're just changing default "Session ID" to "Session token"
      // You can use this mechanism to adjust default localization to your taste, not
necessarily just for other languages
      'toolbarContent': 'Session token: {sessionId}'
    }
  }
};
```

> **Tip**
>
> You don't have to list all key-value pairs. Ones not listed are inherited from the defaults.

Default values

```
{
  "agentJoined": "Representative has joined the session",
  "youLeft": "You have left the session. Co-browse is now terminated.",
  "sessionTimedOut": "Session timed out. Co-browse is now terminated.",
  "sessionInactiveTimedOut": "Session timed out. Co-browse is now terminated.",
  "agentLeft": "Representative has left the session. Co-browse is now terminated.",
  "sessionError": "Unexpected error occured. Co-browse is now terminated.",
  "sessionsOverLimit": "Representative is currently busy with another Co-browse session. Co-
browse is now terminated.",
  "serverUnavailable": "Could not reach Co-browse server. Co-browse is now terminated.",
  "sessionStarted": "Your co-browse session ID is {sessionId}. Please spell it to our
representative to continue with co-browsing.",
  "navRefresh": "Representative has refreshed the page. Reloading.",
  "navBack": "Representative has pressed the \"Back\" button. Reloading page.",
  "navForward": "Representative has pressed the \"Forward\" button. Reloading page.",
  "navUrl": "Representative has requested navigation. Reloading page.",
  "navFailed": "Navigation request by representative has failed.",
  "toolbarContent": "Session ID: {sessionId}",
  "contentMasked": "Content is hidden from representative",
  "contentMaskedPartially": "Some content is hidden from representative",
  "exitBtnTitle": "Exit Co-browse session",
  "areYouOnPhone": "Are you on the phone with our representative?",
  "areYouOnPhoneOrChat": "Are you on the phone or chat with our representative?",
  "connectBeforeCobrowse": "You need to be connected with our representative to continue with
co-browsing. Please call us or start a live chat with us, and then start Co-browse again.",
  "sessionStartedAutoConnect": "Co-browse session started. Waiting for representative to
connect to the session…",
```

```
  "browserUnsupported": "Unfortunately, your browser is not currently supported. Supported
browsers are: Google Chrome, Mozilla Firefox, Internet Explorer 11 and above, and Safari 6
and above.",
  "modalTitle": "Co-browse",
  "modalYes": "Yes",
  "modalNo": "No",
  "writeModeInProgress": "Agent has control over the page.",
  "downgradeMode": "Revoke control",
  "modeUpgraded": "Co-browse session was upgraded. Agent has control over the page.",
  "modeDowngraded": "Co-browse session was downgraded. Agent has no control",
  "modeUpgradeRequested": "Agent requests upgrading Co-browse session to \"write\" mode. In
\"write\" mode agent will have control over the page."
}
```

Genesys Co-browse JavaScript API

Contents

- Developer

For advanced users, deploying Co-browse to your website without using Genesys Widgets can give you the control you want for your environment.

Related documentation:

-

> **Important**
>
> This article contains advanced functionality and assumes that you are **not** using Genesys Widgets to add Co-browse to your website. Genesys Widgets is recommended in most cases.

Deploying Co-browse to your website

Prerequisite: you must have Co-browse provisioned and an API Key provided to you.

A good starting point is this script:

Configuring Co-browse

Co-browse is configured via a `global _genesys.cobrowse` variable.

The following options are configurable as properties of an object passed to `_genesys.cobrowse`:

debug

Default: false

Set to `true` to enable debugging console logs.

```
window._genesys.cobrowse = {
  debug: true;
};
```

disableBuiltInUI

Default: false

Set to `true` to use a custom Co-browse UI. Use the Co-browse JavaScript API to implement a custom UI.

```
window._genesys.cobrowse = {
  disableBuiltInUI: true
};
```

You can still start the Co-browse session with just the configuration above, but the main components of the UI, such as the toolbar and notifications, will be missing.

primaryMedia

Default: null

Used to pass an object implementing an external media adapter interface.

Example:

```
var myPrimaryMedia = {
  initializeAsync: function(done) { /* initialize your media here and then call done() */ },
  isAgentConnected: function() { /* return true or false depending on whether an agent is
connected */ },
  sendCbSessionToken: function(token) { /* send the Co-browse session token to agent */ }
};

window._genesys.cobrowse = {
  primaryMedia: myPrimaryMedia
};
```

See Integrating Co-browse with Chat for more details.

If Co-browse does not detect any primary media or detects that the agent is not connected with the primary media, Co-browse will still ask the user, "Are you on the phone with representative?" before starting the Co-browse session.

setDocumentDomain

Default: false

Determines if Co-browse sets the `document.domain` property. If set to `true`, Co-browse modifies the `document.domain` property. If set to `false`, Co-browse does not modify `document.domain`.

Co-browse modifies `document.domain` to support cross-subdomain communication between iframes and the topmost context. Setting `setDocumentDomain` to `false` stops synchronization of subdomain iframes from working.

```
window._genesys.cobrowse = {
  setDocumentDomain: true
};
```

disableBackForwardCache

Default: true

By default, Co-browse disables Safari's Back/Forward cache, which can stop Co-browse sessions from

functioning.

Setting `disableBackForwardCache` to `false` can make Co-browse unusable in Safari when users click the back or forward browser buttons.

```
window._genesys.cobrowse = {
  disableBackForwardCache: false
};
```

Accessing the API

Since the main Co-browse JavaScript file is added to the page asynchronously, you cannot instantly access the Co-browse APIs. Instead, you must use a function that will accept the APIs as an argument.

For that, add a special `.onReady` property in `_genesys.cobrowse` configuration and set it to empty array.

```
if(!window._genesys)window._genesys = {};
window._genesys.cobrowse = {
  apikey: ,
  onReady: []
};
```

Now, you can use `_genesys.cobrowse.onReady.push(callbackFn)` anywhere in your code. When the Co-browse JavaScript is loaded and the API is available, Co-browse will call back the `callbackFn` with the reference to the API object.

```
_genesys.cobrowse.onReady.push(function(cobrowseApi) {
  // use the API here
});
```

Using the API

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

Co-browse in iframes

Co-browse synchronizes the content of any iframes that exist on the page, given:

- the iframe is on the same domain as the page
- the page in the iframe has Co-browse JavaScript

Some Co-browse UI elements, such as the toolbar, should not appear in an iframe. Common Co-browse UI elements (such as notification that an element is masked) should appear whether or not 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 in an iframe but in "top" context.

- 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

You can use the `isTopContext` variable to determine which context Co-browse is rendered in. `isTopContext` is passed to the onReady callback and equals `true` if Co-browse is rendered in the top context and `false` in iframe.

Example:

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

See Accessing the API 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 an object with the following three 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, `9.0.002.02`.

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

onSessionStarted

This signal is dispatched when a Co-browse session is successfully started 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);
```

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.
    - `sessionsOverLimit`—Agent is busy with another co-browse session and is prohibited from starting another session at the same time.
    - `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();
});
```

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`.

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

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);
    })
}
```

Static content example, without JS API usage:

```
...
```

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.title = 'Content of this elements is masked for representatives.';
});
```

Top Context API

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

isBrowserSupported()

This method checks for the presence of MutationObserver and a few other required APIs, not for browser type and version. It returns `true` when the browser supports required APIs and `false` otherwise.

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 initiates navigation such as refresh, back, forward, or enters a URL into the agent Co-browse UI. Signal is dispatched a few seconds before the navigation happens. This can be used, for example, to 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:

```
cobrowseApi.onAgentNavigated.add(function(details) {
    if (details.url) {
        alert('Representative has navigated to the page: ' + details.url);
    } else {
        alert('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:

```
cobrowseApi.onNavigationFailed.add(function() {
    alert('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
  }
});
```

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").
    }
});
```

Integrating Co-browse with Chat

External chat can be connected to Co-browse via an adapter object assigned to the `_genesys.cobrowse.primaryMedia` option. Such object may implement the following methods:

initializeAsync(done)

Use this only if your chat initializes asynchronously and you cannot be sure it is ready before Co-browse.

If the `initializeAsync` method is implemented, the Co-browse JavaScript will call the method and pass it a done callback. You must call the done callback when your media finishes initialization.

```
var myChatAdapter = {
  initializeAsync: function(done) {
    waitForChatInitialization(function() {
      // tell Co-browse chat is now ready
      done();
    });
 }
};
```

sendCbSessionToken(token)

Co-browse will use it to pass the session token to the agent. The Co-browse session token is a string consisting of nine digits.

Example:

```
myChatAdapter = {
    sendCbSessionToken: function(sessionToken) {
        myChat.sendMessage('User has started Co-browse session: ' + sessionToken);
    }
};
```

If you use Genesys Agent Workspace, wrap the Co-browse token in a `{start:}`, then the agent will join a Co-browse session as soon as he or she receives the token.

```
// For example:
myChatAdapter.sendCbSessionToken = function(token) {
  myChat.sendMessage('{start:' + token + '}');
};
```

isAgentConnected()

This method must return a `true` or `false`.

Co-browse calls this method before calling the `sendCbSessionToken`. If `isAgentConnected` returns `true`, Co-browse will call the `sendCbSessionToken` method. If `isAgentConnected` returns `false`, the user will be asked to connect with an agent via phone before starting Co-browse. If the method is absent, the user will be asked to connect with an agent via phone or chat before starting Co-browse.