Developer's Guide

Integration with Agent Applications

12/16/2025

Integration with Agent Applications

Contents

Overview

Co-browse functionality is integrated with an agent application in two steps:

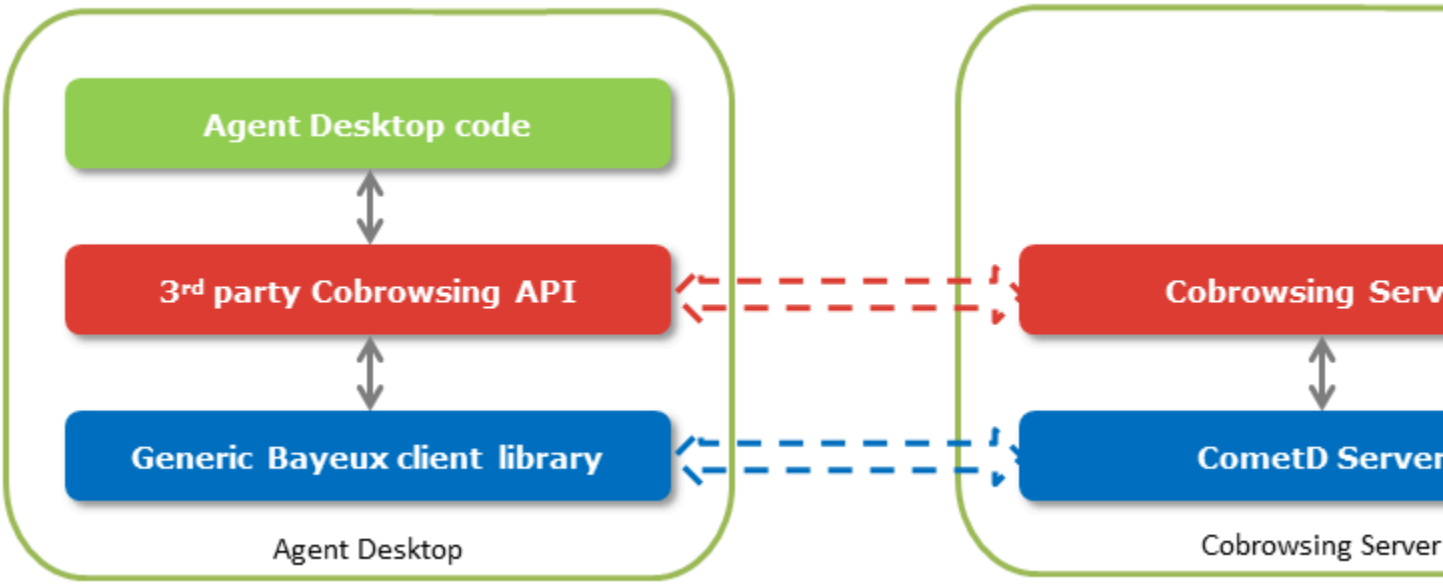
1. The Co-browse agent web UI is integrated with an agent application UI in one of two ways:
   - A browser (embedded) for desktop applications such as Interaction Workspace.
   - An iframe for web applications.
2. The agent application independently connects to the Co-browse Server in order to:
   - Control the co-browse session.
   - Integrate the co-browse session with the primary interaction (chat or voice).
   - Attach Co-browse data to the primary interaction.

Connection to Co-browse Server

Co-browse Server allows clients to inter-operate within a Co-browse session, mainly using a CometD-based API. The connected clients have one of the following roles in the session:

1. Customer — JavaScript client that shares the source web page.
2. Agent — JavaScript client that creates a remote view of the shared page.
3. Controller — Agent application that controls the Co-browse session.

Controller to Co-browse Server Connection



> **Important**
>
> Language-specific libraries for use inside agent applications to integrate with Co-browse Server (third-party Co-browsing API in the graphic above) have not been developed yet.

The agent application establishes a connection to the Co-browse Server through the third-party Co-browse API library (it will be built on top of a CometD .NET client and a Java client, for .NET and Java respectively) before loading the Co-browse agent UI. It supplies the Co-browse session access token, which is required to connect to an already created session. As a result, the agent application is connected to the session as the Controller.

Integration with Agent Desktop and Web-Based Applications

Integration with Agent Desktop Applications

In order to integrate an agent desktop application with Co-browse you must develop a plug-in that:

1. Serves the Co-browse agent page in a browser (Internet Explorer 9+/Firefox/Chrome).
2. Integrates with Co-browse Server (Controller connection).

To avoid race conditions when receiving session-activated notifications, the Co-browse plug-in should connect as Controller before opening an agent page in a browser. In future releases, the session join response may be extended to include session information (for example, session activation time).

Co-browse Web Agent Page

The agent page is the agent side of the co-browse functionality. The agent page is opened in a browser with a dynamically built URL in the following format:

```
<schema>://<host>:<port>/cobrowse/slave.html#nosidinput=1&sid=<sessionToken>
```

`<schema>` — `http` or `https`

`<host>` — Co-browse service host (from Configuration Server)

`<port>` — Co-browse service port (from Configuration Server)

`<sessionToken>` — Co-browse session token. This token is transferred from user to agent by any communication channel.

`nosidinput=1` — Specify this parameter if the plug-in UI provides its own session ID input field.

> **Important**
>
> For improved security, `slave.html` parameters are passed in the fragment identifier (also known as hash) of the URL. Formerly, `slave.html` parameters, including session ID, were passes as query parameters. The query parameter method is still supported for backwards compatibility but is subject to removal in future releases.

Co-browse Server Integration

Server integration can be split into the following functions:

1. Chat integration — Intercept the "start co-browse" chat message (optional).
2. Join a co-browse session (mandatory).
3. Close a co-browse session (mandatory).

> **Important**
>
> Co-browse Server provides CometD and REST collaboration interfaces. The agent desktop application can use both. In order for it to work correctly, the application must be subscribed to at least the JOIN, ACTIVATED, and DEACTIVATED CometD channels.

Chat Integration

This is optional functionality. It provides automatic co-browse session start by a specially formatted chat message. For this functionality, the agent desktop application must be able to:

1. "Listen" to the chat interaction (chat messages).
2. Check if a message matches a regular expression: `^\{start:\d{9}\}$`
3. If the message matches, parse the session token from the message (nine digits after "start:").
4. Join a co-browse session.

Join a Co-browse Session

A co-browse session can only be joined by having a session token (see Co-browse Web Agent Page) and completing the following steps:

1. Issue a request to the REST GET session method.
2. Use the reply to *stick* to the server that "owns" the session (see Stickiness). There are 2 ways of doing this:

- Pass the `gcbSessionServer` cookie with the server name (`sessionServerName` in response) in all further HTTP requests (including CometD).
- If the cookies are problematic, use the server URL (`sessionServerUrl` in response) for all further HTTP requests (including CometD).
  Note: For this to work you have to have the **serverUrl** option set for all Co-browse nodes in a cluster.

3. Establish a CometD connection.
4. Join a co-browse session as Controller by sending a message on the JOIN channel.
5. Wait for the async server response (notification) on the JOIN channel.
6. Attach the following to the current interaction:
   - `CoBrowseAssociated` — The co-browse session flag. This marks the interaction with an active/inactive co-browse session - "Yes" value.
   - `CoBrowseSessionsQuantity` — The number of co-browse sessions for the current interaction.
   - `CoBrowseSessionId` — The co-browse session history identifier received from the JOIN channel (`sessionHistoryId`).
7. Wait for the async server notification on the ACTIVATED channel.
8. Attach the following to the current interaction:
   - `CoBrowseStartTime` — A string with the co-browse session start time in UTC (session start time as a timestamp received from the ACTIVATED channel).

Close a Co-browse Session

A co-browse session may be closed due to the following reasons (click "[Show details]" for information about how to handle each scenario):

- The user or agent exits the co-browse session (via the Co-browse web UI) without closing the primary interaction. **[Show details]**

  In this scenario, complete the following steps:

  1. Wait for the async server notification on the DEACTIVATED channel.
  2. Attach the following to the current interaction:
     - `CoBrowseDuration` — A summary, in seconds, of the duration of all co-browse sessions (in case there were multiple co-browse session conducted within the same primary interaction).
     - `CoBrowseEndTime` — The current co-browse session end timestamp, as a string.
     - `CoBrowseAssociated` — The co-browse session flag. This marks the interaction with an active/inactive co-browse session - "No" value.

- The primary interaction is transferred (Co-Browse Server currently does not support co-browse session transfer). **[Show details]**

  In this scenario, complete the following steps:

  1. Intercept the interaction transfer event.
  2. Send a synchronous (via REST) or asynchronous (via CometD) STOP command.
  3. Attach data to the current interaction based on data received in response to the stop command (see

"The user or agent exits the co-browse session" above for details).

4. Handle the deactivated notification to clean up resources (disconnect the CometD client, for example) in a unified way.

- The inactivity timeout has expired against the primary interaction (see Interaction Server's option `settings/handling-timeout`). **[Show details]**

  When the inactivity timeout expires against the primary interaction, then the interaction is taken away from the agent and placed back into the queue and routed once more.

  The Co-browse plug-in reaction is the same as the primary interaction transferred case (since the inactivity timeout case can be thought of as a transfer initiated by the system).

- The primary interaction is closed. **[Show details]**

  In this scenario, complete the following steps:

  1. Intercept the interaction closing event.
  2. Perform steps 2-4 in the "The primary interaction is transferred" section above.

> **Important**
>
> To attach data to the interaction, the agent application must attach the data **before** the interaction is actually closed.

Integration with Agent Web-based Applications

The web agent Co-browse application can be easily integrated with any web application, such as a web-based agent desktop. To do this, simply add an iframe with the web agent app into the web application:

```
<!doctype html>
<head>...</head>
<body>
...
<iframe src="http://<CB_SERVER>/cobrowse/slave.html"></iframe>
...
</body>
```

It is important to keep `slave.html` in the URI, even if you proxy the Co-browse Server, because the Co-browse scripts rely on it. For example, the URI http://my-site.com/cobrowseAgent/ **does not** work, even though it actually points to `slave.html`.

You can enable the web browser console logs in the standard way by adding the debug=1 URL parameter.

```
<iframe src="http://<CB_SERVER>/cobrowse/slave.html#debug=1"></iframe>
```

To have the web agent immediately join a session, create an iframe with a predefined `sid` URL parameter:

```
<iframe src="http://<CB_SERVER>/cobrowse/slave.html#sid=123456789"></iframe>
```

You should use the maximum possible size for the iframe because the Co-browse area adjusts to the end customer's browser window and can be big if the user has a large monitor, for example. If the Co-browse area becomes larger than the containing iframe, an agent will see scrollbars, which may not be very convenient.

> **Important**
>
> For improved security, `slave.html` parameters are passed in the fragment identifier (also known as hash) of the URL. Formerly, `slave.html` parameters, including session ID, were passes as query parameters. The query parameter method is still supported for backwards compatibility but is subject to removal in future releases.