Developer's Guide

Genesys Co-browse 8.1.3

1/6/2022

Table of Contents

Genesys Co-Browse 8.1 Developer's Guide

Welcome to the *Genesys Co-browse 8.1 Developer's Guide*. This document introduces you to the architecture of Genesys Co-browse and provides information about how you can customize it for your website. See the summary of chapters below.

About Genesys Co-browse

Find out about the architecture of Genesys Co-browse.

Conceptual Model

Customizing Genesys Co-browse

Learn how to customize the look and feel of the Genesys Co-browse widgets.

Customize the Genesys Co-browse User Interface

Localization

DOM Restrictions

Integrating with Chat

Find procedures to integrate Genesys Co-browse with chat.

Genesys Co-browse and Chat

External Chat without Integration

Using Data Attached to the Primary Interaction

Find procedures to use Attached Data.

Attached Data Overview

Using Attached Data for Reporting

Using Attached Data to Customize Business Processes

Integrating with Agent Applications

Find procedures to integrate Genesys Co-browse with agent desktops and agent web-based applications.

Integrate with Agent Applications

Product Overview

Overview

Genesys Co-browse provides the ability for an agent and the end customer to browse and navigate the same web page at the same time. In a Genesys Co-browse session, both the agent and the customer share the same instance of the screen, as opposed to a conventional screen sharing application, where one of the parties sees an image of the other party's browser instance.

Components

Genesys Co-Browse is composed of the following components:

- **Genesys Co-browse Server** is a server-side component that is responsible for orchestrating the co-browsing activities between the end consumer and the agent.
- **Genesys Co-browse Plug-in for Interaction Workspace** provides co-browsing functionality for Interaction Workspace users.
- **Genesys Co-browse Plug-in for Workspace Desktop Edition** provides co-browsing functionality for Workspace Desktop Edition users.
- **Genesys Co-browse Sample Reporting Templates** provides configuration files and reporting templates for getting real-time and historical statistic data.
- **Integrated JavaScript Application** is a JavaScript component that includes the Chat, Co-browse, and (optionally) Web Engagement Tracker JavaScript applications. You should add this component to the pages on your website where you want to enable co-browsing.

Features

Genesys Co-browse includes the following features:

- Active participation—both the agent and the customer have the ability to take control.
- Browsing always happens on the customer side.
- Administrators are able to restrict what the agent can do and see on the web page. The customer can easily identify which fields are masked from the agent. Administrators can easily specify which DOM elements (buttons, check boxes, and so on) the agent must not be able to control.
- Support for multiple browsers, cross-browser support, and same-browser support.
  Support for scenarios in which the agent and customer are using different browsers.

  Support for scenarios in which the agent and customer are using different versions of the same browser.

- The customer can co-browse without downloading or installing any plug-ins.

Browser Support

Genesys Co-browse supports the following browsers:

- Internet Explorer 9 and above (Windows)
- Firefox 17 and above (Windows, Linux, and Solaris)
- Safari 6 and above (Mac)
- Google Chrome (Windows)

Genesys recommends that you use Internet Explorer 10 or above on agent machines for improved synchronization speed due to Web Sockets support and better JavaScript engine performance.

Warning

We strongly advise against IE Conditional Comments.

Important

Interaction Workspace uses *only* Internet Explorer as the embedded browser for working with Co-browse sessions.

Related Components

Genesys Co-browse interacts with the following Genesys Products:

- Interaction Workspace — The Genesys Co-browse Plug-in for Interaction Workspace is required to interface Genesys Interaction Workspace with Genesys Co-browse. This plug-in enables the agent to join and terminate a co-browsing session with a customer.
- Chat Server — An eServices component, Chat Server handles chat interactions between agents and web visitors.
- Genesys Web Engagement — If Genesys Web Engagement is installed, the chat widget can be integrated with Genesys Co-browse and used to initiate a co-browsing session.

Minimum Required Components

The following components are mandatory for Genesys Co-browse:

Server Name	Compliant Versions (and later)
Configuration Server	8.1.100.14
Chat Server	8.1.000.41
Interaction Workspace	8.1.401.44
Workspace Desktop Edition	8.5.100.05
Universal Contact Server	8.1.001.12
Interaction Server	8.1.000.13
Universal Routing Server	8.1.100.09
Stat Server	8.1.000.23
CCPulse+	8.0.101.34
Data Modeling Assistant	7.6.100.01

Genesys Co-browse 8.1.3 and Previous Co-Browsing Solutions

The Genesys Co-browse 8.1.3+ solution should not be associated with the Web API Cobrowse Samples. These samples work with the old KANA-based Co-Browsing Server and do not work with this new Co-browse solution; however, you may use a chat interaction started from the Web API Chat Samples to initiate a new Co-browse session from an instrumented page with an agent.

Important

Genesys strongly recommends that you use the Co-browse chat widget to initiate a Co-browse session with an agent. This chat widget is designed to work with the new Co-browse solution in the most optimal way. Agents do not even need to paste the Co-browse session ID manually into their screen - the Co-browse page is opened automatically once the agent clicks the "Co-browsing" button during a live chat.

For more information about how to work with the Co-browse chat widget, see the following sections:

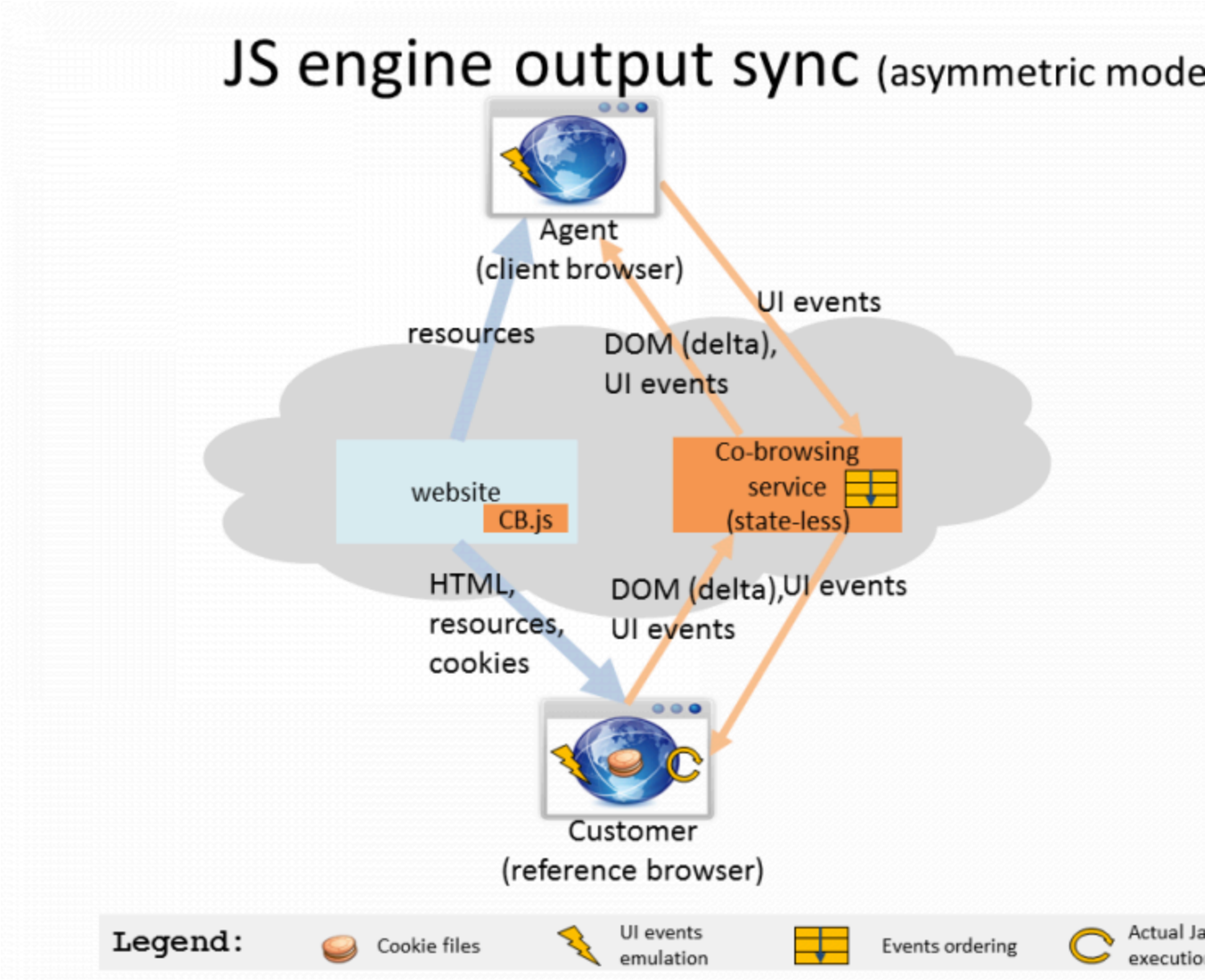
- Initiating a Co-browse session from an integrated chat
- Genesys Co-browse and Chat

For more information about how to work with external chats like the Web API Chat Samples, see:

- Initiating a co-browse session from a voice call or external chat without integration
- External Chat without Integration

Conceptual Model

Schematic Diagram



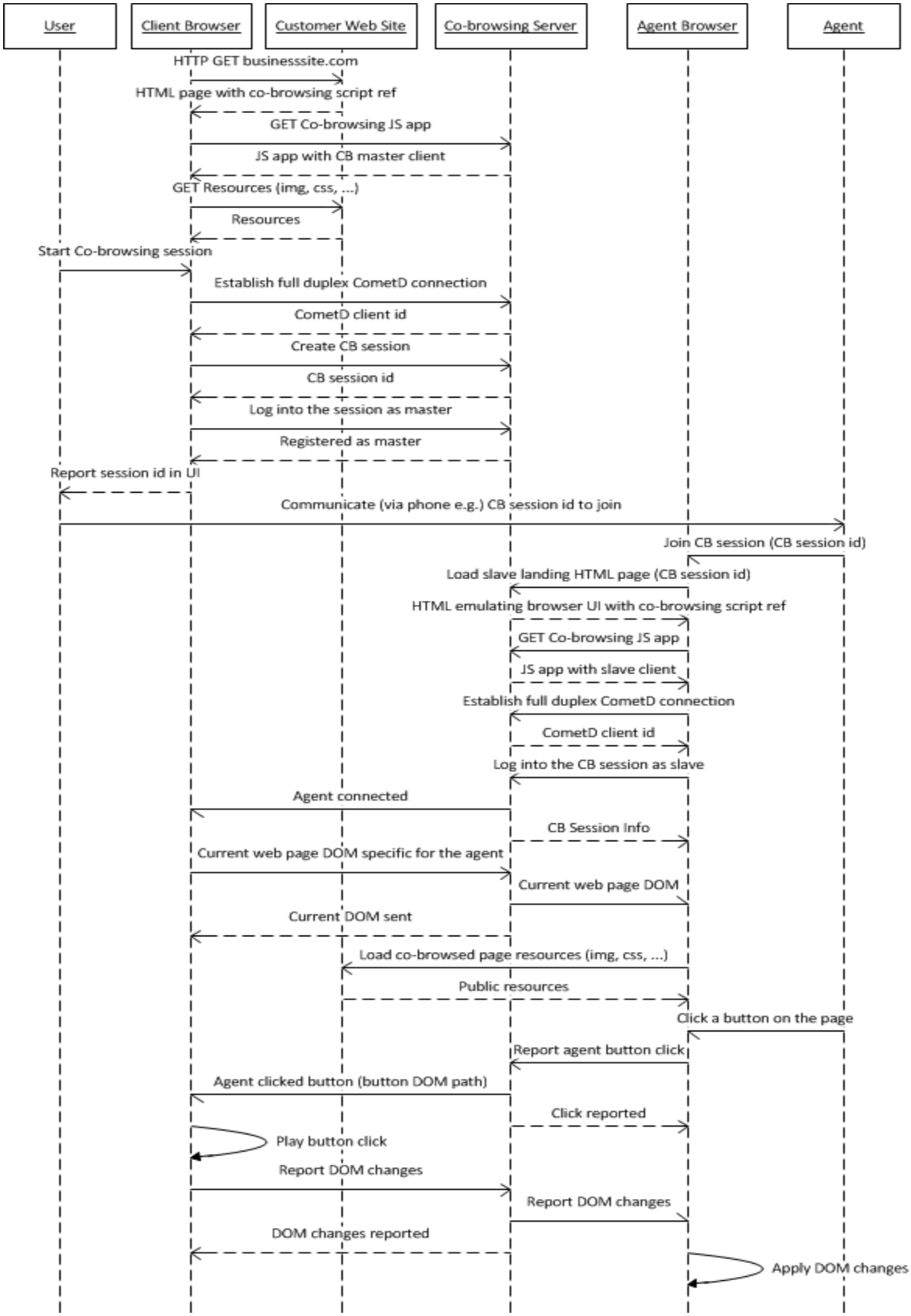
Schematic Diagram of Genesys Co-browse

After the user and an agent have been connected and a co-browsing session has been established, a number of things start synchronizing for each page.

One of the advantages of the approach shown in the image above, is that it allows for co-browsing to start at any time. For example, a co-browse session can be initiated while a user is in the middle of filling out a long form. This model also eliminates the need to worry about cookies and session timeouts on the slave side.

Sequence Diagram

The following diagram illustrates the detailed communication between the parties mentioned above.


User
Client Browser
Customer Web Site
Co-browsing Server
Agent Browser
Agent
HTTP GET businesssite.com
HTML page with co-browsing script ref
GET Co-browsing JS app
JS app with CB master client
GET Resources (img, css, ...)
Resources
Start Co-browsing session
Establish full duplex CometD connection
CometD client id
Create CB session
CB session id
Log into the session as master
Registered as master
Report session id in UI
Communicate (via phone e.g.) CB session id to join
Join CB session (CB session id)
Load slave landing HTML page (CB session id)
HTML emulating browser UI with co-browsing script ref
GET Co-browsing JS app
JS app with slave client
Establish full duplex CometD connection
CometD client id
Log into the CB session as slave
Agent connected
CB Session Info
Current web page DOM specific for the agent
Current web page DOM
Current DOM sent
Load co-browsed page resources (img, css, ...)
Public resources
Click a button on the page
Report agent button click
Agent clicked button (button DOM path)
Click reported
Play button click
Report DOM changes
Report DOM changes
DOM changes reported
Apply DOM changes

Sequence diagram

DOM Restrictions

Genesys Co-browse allows you to hide sensitive data from agents and restrict control of elements in a co-browse session by providing a map of the elements that should be restricted. For every element in this map, there should be a CSS selector which identifies it and the type of restriction applied.

You can implement two types of restrictions:

- Data masking
- DOM control

> **Important**
> Data masking and DOM controls apply to both Pointer Mode and Write Mode.

For details about DOM restrictions, see Configuring DOM Restrictions.

Data Masking

Private, critical, or sensitive customer-related data can be masked on the agent side with asterisks. This masked data does not leave the customer browser; it is not retrieved by Co-browse Server or the agent browser.

If agents try to change a masked input field, they see a notification that only the customer can access the input field.

Data masking can be applied to any visual HTML element. For input elements, the `value` attribute is masked; for all other elements, the text content is masked.

On the customers' side, if the data masked input field is in focus (for example, a customer selects an input field to enter a credit card number), customers see a notification that the information they type is not visible to agents.

> **Important**
> Data masking for all password inputs is enabled in the system and cannot be changed by Configuring DOM Restrictions.

DOM Control

DOM Control allows you to disable web page elements that agents should not be able to use. One example could be the `Sumbit` button for a web form.

> **Important**
> Buttons of type `submit` are disabled for agents by default.

Configuring DOM Restrictions

To implement DOM restrictions, you must create an XML file to store your configuration. In this XML file, you configure rules for data masking and DOM control. Sets of rules match to specific pages or groups of pages using regular expressions (regexp).

Creating an XML configuration file

Create an XML configuration with the following structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<domRestrictions>
    <restrictionsSet>
        <uriTemplate type="regexp" pattern="<URL matching regex>"/>
        <dataMasking>
            <element selector="..."/>
        </dataMasking>
        <domControl>
            <element selector="[type=submit]"/>
        </domControl>
        <dataMasking/>
    </restrictionsSet>
</domRestrictions>
```

For a detailed example, see XML configuration file example.

After you create a DOM restrictions configuration file, you must link the XML configuration file to your Co-browse server.

XML structure description

The XML configuration file contains the following elements:

- `<domRestrictions>`—root tag containing any number of `restrictionsSet` tags.

  ```
  <domRestrictions>
       <restrictionsSet>
           ...
       </restrictionsSet>
       <restrictionsSet>
  ```

```
        ...
    </restrictionsSet>
</domRestrictions>
```

- `<restricitonsSet>`—defines restriction rules applied to a web page or group of pages matching the regular expression in the `pattern` attribute. Restriction sets are cumulative. More than one restriction set can apply to a single webpage.

  ```
  <restrictionsSet>
      <uriTemplate type="regexp" pattern="..."/>
      <dataMasking>
          ...
      </dataMasking>
      <domControl>
          ...
      </domControl>
  </restrictionsSet>
  ```

- `<uriTemplate>`—defines the set of web pages the restriction applies to using a URL matching regex pattern.

  ```
  <uriTemplate type="regexp" pattern="<URL matching regex>"/>
  ```

  `pattern` value:

  - The `pattern` value is a regular expression (regex) that matches the URL of the target page. For more about regular expressions, see http://www.regular-expressions.info/.
  - Some characters have special meaning in regular expression syntax. When using these characters, you must *escape* the characters using a backslash (\). URL characters you must escape: `.:()\/?*|+{}^$[]`.
  - For example, the regex for the URI `http://www.genesys.com/about-genesys/contact-us` can be:

    ```
    http:\/\/www\.genesys\.com\/about-genesys\/contact-us
    ```

    or simply,

    ```
    www\.genesys\.com\/about-genesys\/contact-us
    ```

    > **Tip**
    > You may use online tools like Regexper to validate your regular expressions.

    `pattern` examples:

Regex	Description
`.*`	Matches any page
`login\.html`	Matches all pages that include `login.html` in the URL
`(login|registration)-page\.html`	Matches pages prefixed with `login` or `registration` such as `login-page.html` and `registration-page.html`.
`genesys\.com\/about-genesys\/contact-us`	Matches pages like `http://www.genesys.com/about-genesys/contact-us` and `https://genesys.com/about-genesys/`

Regex	Description
	contact-us.
(https\:\/\/)	Matches all HTTPS pages

- <dataMasking>—list of all web elements whose data should be masked.

```
<dataMasking>
    <element selector="..."/>
</dataMasking>
```

- <domControl>—list of all web elements that should be restricted from agent control.

```
<domControl>
    <element selector="..."/>
</domControl>
```

- <element>tag describing which element(s) to restrict.

```
<element selector="<jQuery specific selector>"/>
```

element examples:

Value	Description
<element selector="#sendRequest"/>	Element with id="sendRequest"
<element selector="[name=login]"/>	Element with name="login"
<element selector="[type=submit]"//>	Element with type="submit"
<element selector=".SendButton"/>	Elements with "SendButton" class
<element selector="[href='/about-us/contacts']"/>	Element with href="/about-us/contacts"
<element selector="[href$='.org']"/>	Elements with href attribute ending with ".org"
<element selector=":button"/>	All normal buttons
<element selector="[type=submit]:not(#uniqueSubmitId)"/>	All submit buttons without id = "uniqueSubmitId"
<element selector=".Input:not(#InputId2)"/>	All input fields in class Input and without id = "InputId2"

For more information about selecting elements from a webpage see Using browser tools to select an element.

XML structure summary

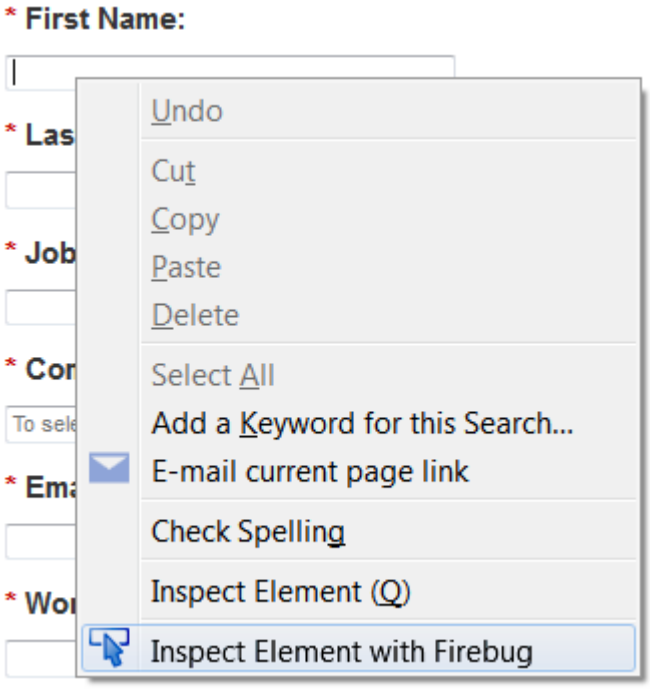
For each matched page or group of pages defined by a pattern in a <uriTemplate> tag, you can provide data masking and DOM control rules for specific page elements. Elements are represented by <element> tags. Each <element> contains a selector attribute with a jQuery selector describing the path to the element on the page.

Using browser tools to select an element

You can use browser tools to help you define an element selector in your XML configuration files. The example below uses the Firebug browser tool for Firefox.

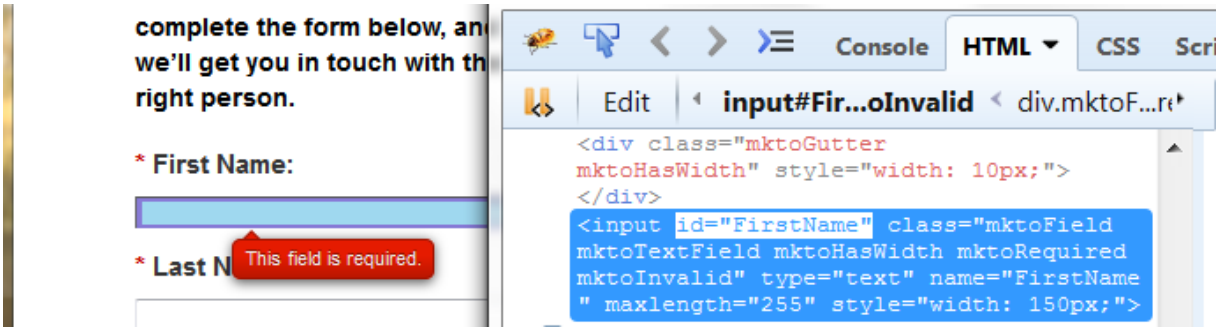
Using Firebug to find an element selector

1. Open the webpage containing the element you want to select. Right click on the element and click `Inspect Element with Firebug` to open the Firebug tool.

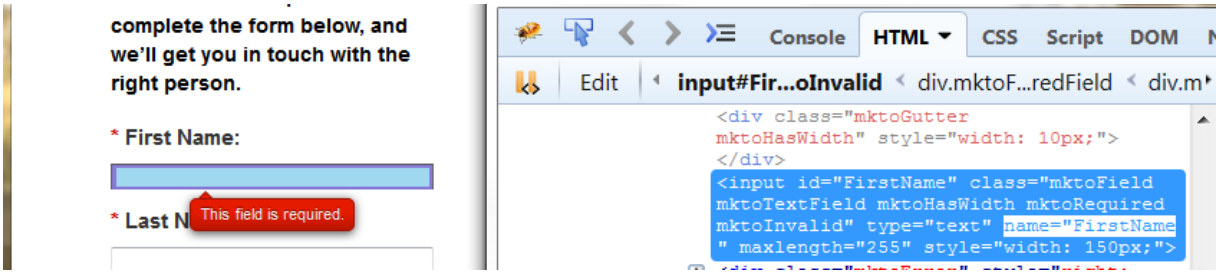


2. Firebug can help you identify the right selector for an element. For example, you could use one of the following as a selector:

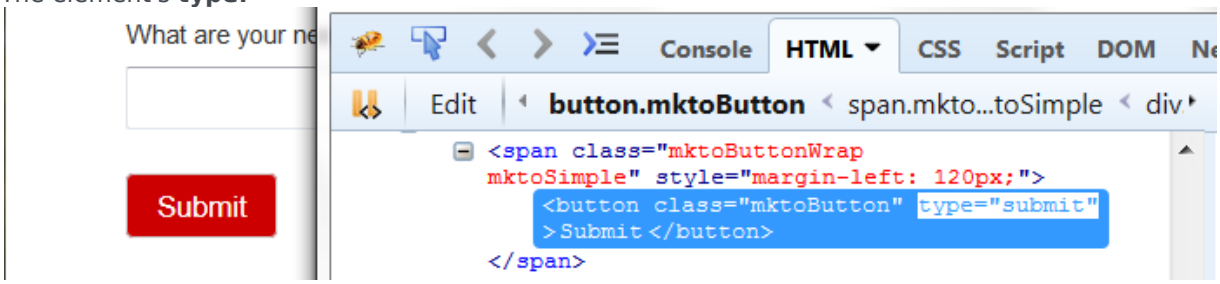
   - The element's **id:**



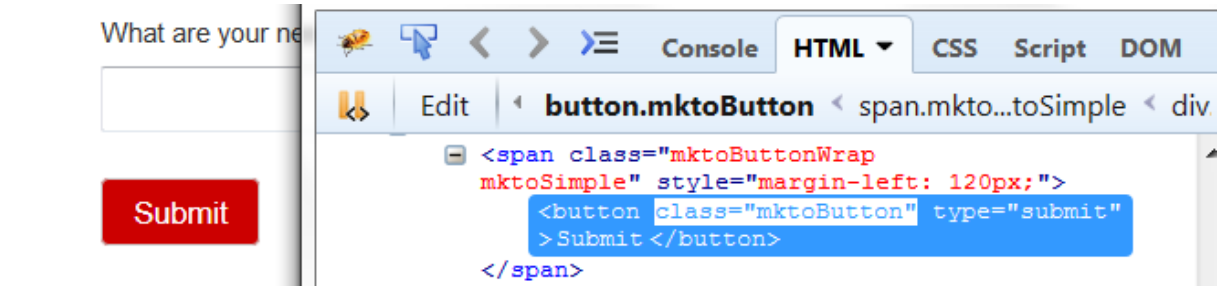
   - The element's **name:**



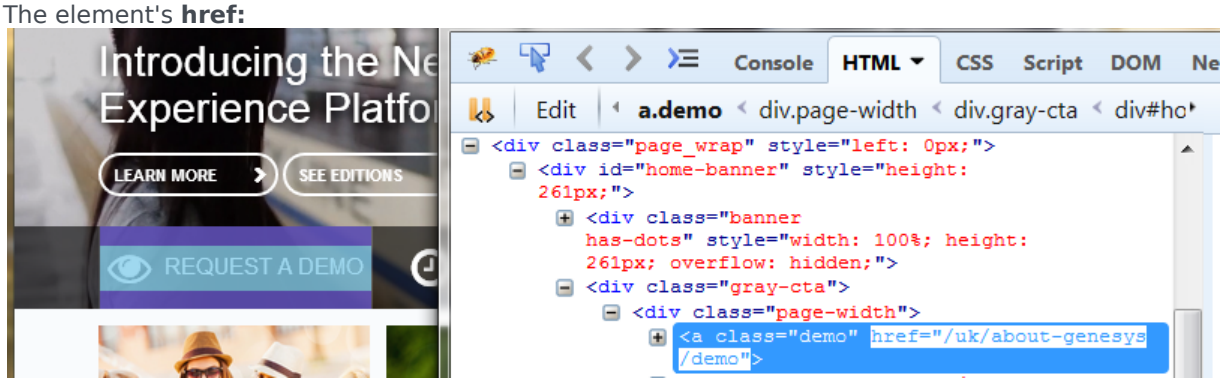
- The element's **type:**



- The element's **class:**



- The element's **href:**



> **Tip**
> For a comprehensive list of JQuery selectors, see http://api.jquery.com/category/selectors/.

Default DOM Restrictions Configuration

The default configuration ensures DOM control for all submit buttons is restricted from agents.

> **Important**
> To preserve this default behavior, create your custom configuration by extending and

not overwriting the default configuration.

```
<?xml version="1.0" encoding="UTF-8" ?>
<domRestrictions>
    <restrictionsSet>
        <uriTemplate type="regexp" pattern=".*"/>
        <domControl>
            <element selector="[type=submit]"/>
        </domControl>
        <dataMasking/>
    </restrictionsSet>
</domRestrictions>
```

Important

Data masking for all password inputs is enabled in the system and cannot be changed using DOM restrictions configuration.

XML configuration file example

The example below provides a sample configuration with comments explaining the purpose of each element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Use this configuration as a set of examples and common documentation -->
<!--
    domRestrictions contains any number of restrictionsSet elements.
    There must be only one domRestrictions element.
-->
<domRestrictions>
    <!--
        Each set defines restrictions rules matched by URL.

        Starting with including restrictionSet from default configuration,
        so that agents can never click submit buttons on behalf of customers.
    -->
    <restrictionsSet>
        <!-- Pattern ".*" matches any string and therefore any URL -->
        <uriTemplate type="regexp" pattern=".*"/>
        <domControl>
            <!-- All submit buttons (elements with type = "submit" ) will be restricted -->
            <element selector="[type=submit]"/>
        </domControl>
        <dataMasking/>
    </restrictionsSet>
    <!--
         All domControl and dataMasking rules in this restrictionSet will apply
        to pages that have "page.html" in their URL
    -->
    <restrictionsSet>
        <uriTemplate type="regexp" pattern="page\.html"/>
        <domControl>
            <element selector=":button"/> <!-- All normal buttons -->
            <element selector="#mySubmitButton"/> <!-- Concrete element with id =
```

```
"mySubmitButton" -->
            <element selector=".MySubmitButton"/> <!-- All elements that have class
"MySubmitButton" -->
            <element selector="[href='/checkout']"/> <!-- All links that lead to /checkout
page -->
        </domControl>
        <dataMasking>
            <element selector="[name=login]"/> <!-- All elements with name="login" will be
masked -->
            <element selector=".LicenseCode"/> <!-- All elements with class "LicenseCode"
will be masked -->
        </dataMasking>
    </restrictionsSet>
    <restrictionsSet>
        <!-- Range of pages pattern. -->
        <uriTemplate type="regexp" pattern="genesys\.com\/page[1-9]\.html"/>
        <domControl>
            <!--  Element with id = "uniqueSubmitId" will be excluded from restriction -->
            <element selector="[type=submit]:not(#uniqueSubmitId)"/>
        </domControl>
        <dataMasking/>
    </restrictionsSet>
</domRestrictions>
```

Linking the XML configuration file to your Co-browse Server

After you have created your configuration file, set the value of the domRestrictionsURL configuration option from the session section to point to your file. You can configure this value as:

- a URL reachable by Co-browser Server. For example, `domRestrictionsURL=http://cobrowse.com/static/dom_restrictions.xml`.
- a path to the file pre-fixed with `file`. For example, `domRestrictionsURL=file:C:\restrictions.xml`.

Integration with Agent Applications

Overview

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

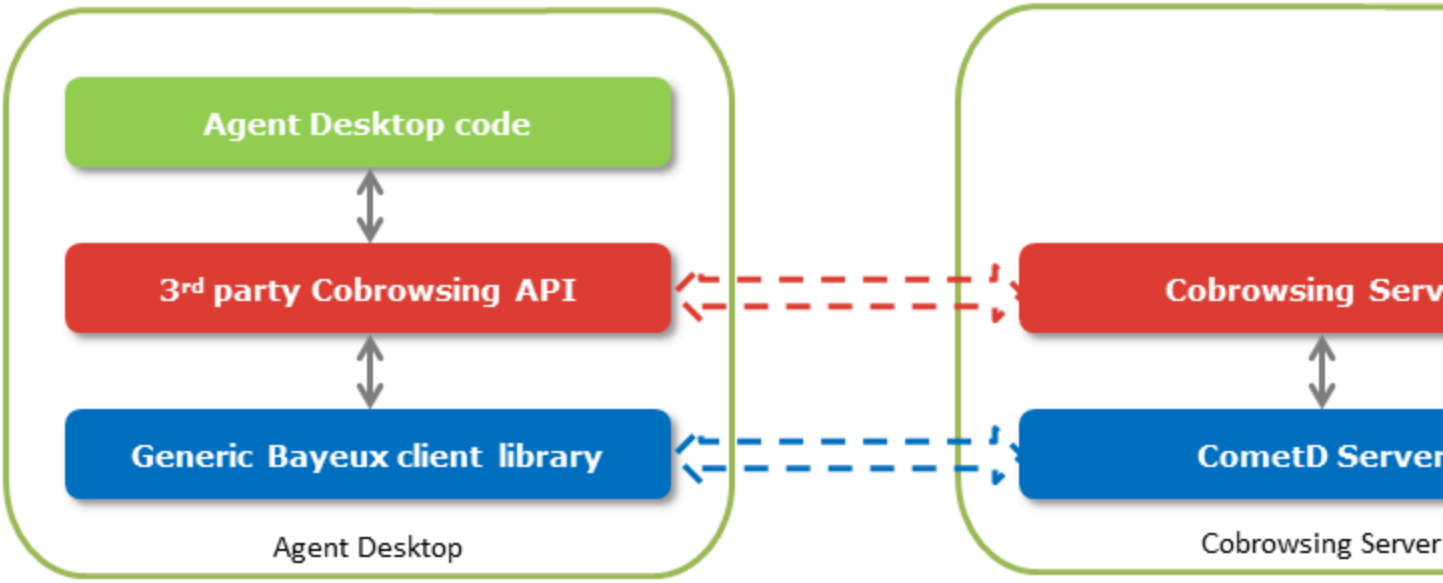
1. The Co-browse agent web UI is integrated with an agent application UI in one of two ways:
   - A browser (embedded) for desktop applications such as Interaction Workspace.
   - An iframe for web applications.
2. The agent application independently connects to the Co-browse Server in order to:
   - 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. Master — JavaScript client that shares the source web page.
2. Slave — 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 slave 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 a slave 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 Slave Page

The slave page is the agent side of the co-browse functionality. The slave 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 Slave Page) and completing the following steps:

1. Determine which server the co-browse session is hosted on by calling the REST GET session method.
2. Make sure the session server name is passed via the `gcbSessionServer` cookie in further HTTP requests (including CometD).
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 slave 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 slave 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/cobrowseSlave/ **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 slave 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 (master user'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.

Chat

Genesys Co-browse supports three levels of integration with chat functionality:

- Co-browse has its own built-in chat widget that is triggered with the "Live Chat" button. To customize the built-in chat widget, see Customizing the Genesys Co-browse UI. You can also implement your own chat widget using the Chat Service API. See Chat Service API#Advanced Usage of the Chat API.
- Co-browse can be used with any external chat without integration. In this case, the user will have to manually transfer the Co-browse session ID to the agent. See External Chat Without Integration.
- Co-browse can be integrated with an external JavaScript-based chat to use the Co-browse built-in "Live Chat" button to start a chat and to automatically transfer the Co-browse session ID to the agent. You will need to implement an External Media Adapter for your chat widget and pass your external media adapter object to the `primaryMedia` Co-browse configuration option.

External Chat Without Integration

Co-browse and non-integrated chat

If your website already has chat, you can use it with Genesys Co-browse without any other integration effort. For example, you may use the Web API Chat Samples to start a chat.

To allow users to initiate a Co-browse session from your website, you must add the Co-browse JavaScript snippet to your web pages. For information about how to enable it, see Website Instrumentation. By default, the instrumentation has a "Live Chat" button used for embedded initiation that is not needed if you already have chat enabled on your website. In this case, you should disable the "Live Chat" button. See Integration Configuration JS API for details.

If your web page is correctly instrumented, the user will see a "Co-browsing" button after loading. Now, when the user clicks the "Co-browsing" button, they will see a notification message. If the user clicks "Yes", the Co-browse session will begin and the user will be prompted to manually transfer the session ID to the agent using either chat or voice.

Overriding the "Co-browsing" button

You can also override the default "Co-browsing" button by placing a custom button on your chat widget so that chat is the explicit entry point to co-browsing.

See Integration Configuration JS API for information on overriding the "Co-browsing" button.

Customize Genesys Co-browse User Interface

The Genesys Co-browse user interface widgets that are visible on your website are based on HTML, CSS, and JavaScript. You can customize these widgets to suit the look and feel of your website.

There are three ways you can customize the UI (User Interface):

- Adding additional CSS to your website that overrides the default CSS for the widgets. See Customizing the CSS below.
- Using the JavaScript API.
- Using Localization.

See Customization Examples for several detailed examples of UI customization.

Customizing the CSS

The Co-Browse JavaScript automatically loads some CSS files that define how the widgets look. To find these files, you can do one of the following:

- Download the complete CSS file from the URL `http(s)://<COBROWSE_HOST>[:<COBROWSE_PORT>]/cobrowse/css/master-all.css`. This is a good starting point for medium to broad customizations such as changing the color scheme.
- Use Firebug, Chrome Developer Tools, or a similar browser tool to select CSS rules for particular elements.

All Co-browse CSS follows the same principles:

- Only classes are used.
- All Co-browse classes begin with the `.gcb-` prefix.

You can override the Co-browse CSS by adding your own custom rules anywhere in the <head> tag.

> **Tip**
>
> Although the Co-browse CSS is loaded dynamically, it is prepended to the beginning of the <head> tag, so your custom CSS will always have higher specificity.

Customization Examples

See Customization Examples for detailed customization examples such as how to customize the Co-browsing button and the toolbar position.

Localization

See Localization.

UI Customization Examples

This is a collection of UI customization examples. Although these examples do not cover every possible customization case, they should give you a sample of what can be achieved and how to begin.

> **Tip**
> Some examples use Firebug as a developer tool. Any other similar tool can be used instead such as Google Chrome Developer Tools, Safari Web Inspector, or Internet Explorer Developer Tools.

> **Tip**
> To customize the built-in chat widget, see Customizing the Chat Widget User Interface.

Example: Customizing Live Chat and Co-browsing Buttons

Customization type: **CSS Based**

Prerequisites

You must have basic knowledge of CSS and HTML.

> **Tip**
> The Live Chat and Co-browsing buttons are images and cannot be modified using the Localization mechanism.

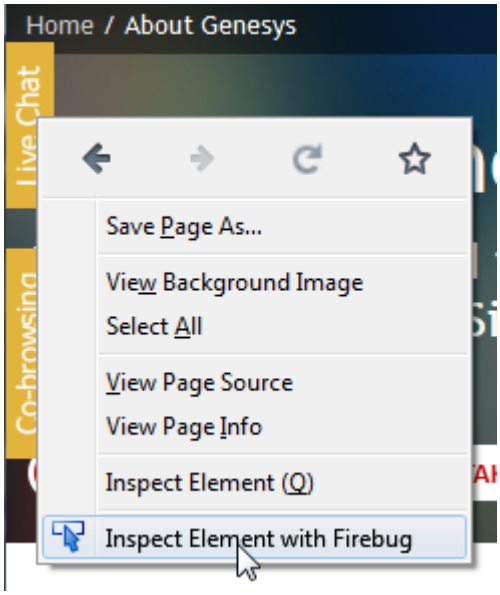
> **Tip**
> You can manage the visibility and moderately change the position of the default buttons using the JavaScript API.

Start of Procedure

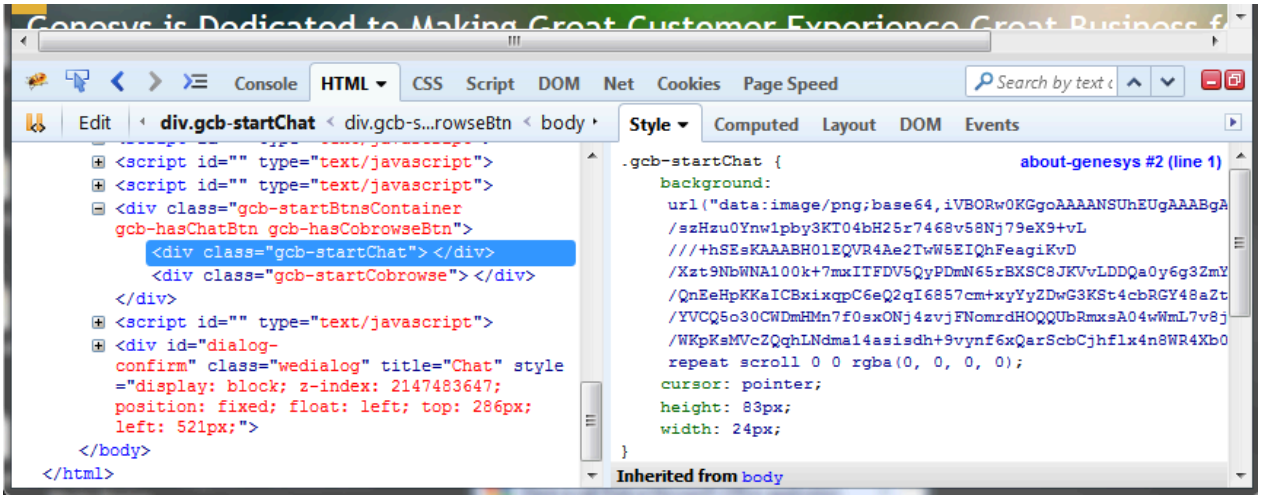
1. In Firefox, open the page instrumented with the integrated JavaScript application. Live Chat and Co-

browsing buttons should appear:

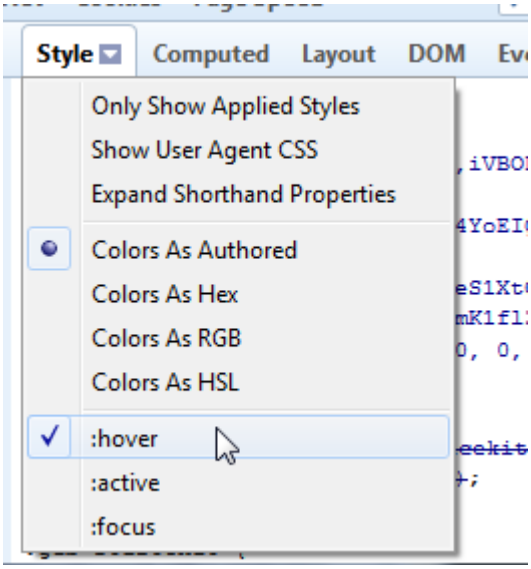
2. Right click the `Live Chat` buttons and click `Inspect Element with Firebug` to open the Firebug tool.



3. In Firebug, you can see the CSS rules responsible for styling the button:



4. Also check :hover rules to see if there are any rules applied to the button when the mouse cursor is over it. To do this, enable the `:hover` modifier in Firebug:



You can see that there is an additional :hover rule for the button:

5. Base on the rules above, you can prepare your own rules that override the defaults. In this example, we will prepare a custom image with a similar size to the default button and override the background. We will use a public service to generate an image of a kitten via an http request. Here is our CSS:

```
/* 1. copy-paste the selector of the element to override */
.gcb-startChat, .gcb-startChat:hover {
/* 2. override some rules: */
  background: url(http://placekitten.com/23/84);
}
```

6. Now we add our CSS to the <head> section of our site. The way this is done will depend on the technology your website uses. We want to add code like this to the webpage:

```
    <style>
        .gcb-startChat, .gcb-startChat:hover {
            background: url(http://placekitten.com/23/84);
        }
    </style>


</head>
```

7. Reload the page. Now, the Live Chat button is replaced with our new image:

> **Tip**
> The `Co-browsing` button can be customized in the same way.

End of Procedure

Example: Customizing the Toolbar Position

Customization Type: **JavaScript Based**

In this example, we will customize the starting position of the Co-browse toolbar:

Prerequisites

1. Experience with JavaScript, jQuery, and browser developer tools such as Firebug.
2. This example uses the the jQuery library to work with the DOM and assumes that jQuery is available on the page.

Start of Procedure

1. Obtain the Co-browse API. See Integrated JavaScript Application for more information about obtaining the Co-browse API. In this example, we use the single-function mechanism.

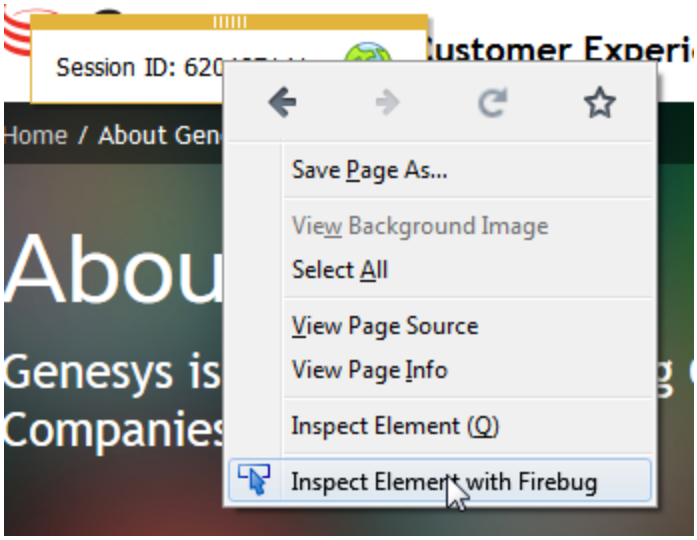
```
var _genesys = {
    cobrowse: {
        onReady: function(cobrowseAPI) {
            // ...
        }
    }
}
```

2. Use the Co-browse API to subscribe to the `onSessionSarted` Co-browse event.

```
cobrowseAPI.onSessionStarted.add(function() {
    // ...
});
```

> **Tip**
> You can subscribe to the readiness of the UI using the onReady Callback.

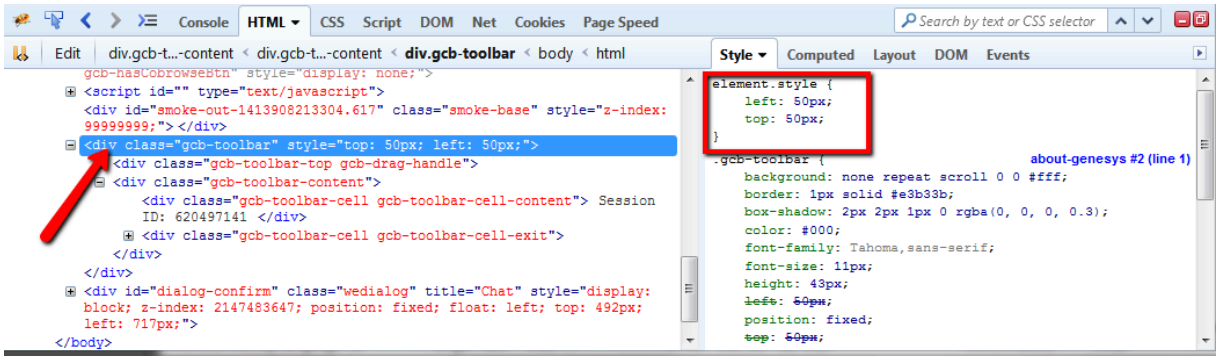
3. Use Firebug to determine which element to customize:
   a. Start a Co-browse session.
   b. Right click the Co-browse toolbar and click `Inspect Element with Firebug` to open the Firebug tool.



c. In Firebug, you can see that the element selected for inspection is not the toolbar itself but one of its sub-elements.

You can also see from the `Styles` tab that position is not set for this element.

d. You need to find the element whose position you want to change. From the DOM tree in Firebug you can see that this is the `.gcb-toolbar`.



4. Now, we will use jQuery to override the position of the toolbar. We will set the starting position to 100 pixels from the top edge and 300 pixels from the left edge.

```
jQuery('.gcb-toolbar').css({
  top: 100,
  left: 300
```

```
});
```

5. Putting it all together, we have the following:

```
// 1. Get the API
var _genesys = {
    cobrowse: {
        onReady: function(cobrowseAPI) {

            // 2. Use the API to subscribe on “session started” event
            cobrowseAPI.onSessionStarted.add(function() {

                // 3. Use jQuery to get the toolbar and reposition it
                jQuery('.gcb-toolbar').css({
                    top: 100,
                    left: 300
                });

            });

        }
    }
};
```

End of Procedure

> **Tip**
>
> You can achieve similar results using CSS:
>
> ```
> .gcb-toolbar {
>    top: 100px !important;
>    left: 300px !important;
> }
> ```
>
> This solution is less ideal because you would have to use the `!important` modifier to override the inline styles of the element and this modifier would make the toolbar non-draggable.

Localization

Genesys Co-browse localization is split into three parts:

- Master UI
- Built-in Chat UI
- Slave UI

Co-browse is localized for English by default. To modify the default English localization or add localization for other languages, you must host the localization files on your servers and configure the Co-browse application to use these files on top of the defaults.

The localization files are plain JSON files, loaded through a JSONP request. This means they can be hosted on any domain, as long as JSONP is supported by the hosting server. Co-browse uses a standard `callback` argument for the callback function name.

Localizing the Master, Integration, and Chat Widget UI

To configure the master UI of Co-browse to use custom localization files, you must pass the URL(s) of the file(s) to the `localization` option of the appropriate subsection of the JavaScript configuration API such as the `_genesys` variable. See Integrated JavaScript Application#Configuration and Integrated JavaScript Application#Localization of Chat and Co-browse.

> **Important**
>
> You must save localization JSON files with UTF-8 encoding.

For details about chat widget localization see the following:

- Chat Widget JS API#Localization for information about chat widget localization.
- Integrated JavaScript Application#Configuring Chat for information about configuring the chat widget in the Integrated JavaScript Application.

Localizing the "Live Chat" and "Co-browsing" buttons

The "Live Chat" and "Co-browsing" default buttons are images and localization changes will not affect the text on these buttons. Instead, you can localize the buttons using one of the following methods:

1. Provide custom localized buttons instead of the defaults. See Integrated JavaScript Application#Providing Custom HTML for Buttons for details.

2. Override how the buttons look using CSS. See Customizing the CSS for details.

Localizing the slave UI

The slave UI is localized by updating the `localization` configuration option to the URL of the JSON localization file in the slave section of the Co-browse Server application. Unlike the master UI, the value is changed in the Genesys Configuration server and not via the instrumentation script.

Caching and updating l10n files

Requests to the l10n files are made on every page that is instrumented with Co-browse, before the Co-browse UI is displayed. Genesys recommends that you implement a caching mechanism for these files if you host them on your servers.

For best performance, add far-future expiration headers (for example, `Expires`, `Cache-Control` or both) to your l10n files. This prevents the browser from requesting these files on each page. Instead, it will take them from the cache. This reduces the start-up time for the Co-browse UI and cuts down traffic for the end user. If you modify a localization file with a far-future expiration header, the browser must request the new version of the file from the server instead of taking it from the cache. To force the browser to do this you must change the URL of the file. You can do this by updating the corresponding `localization` parameter in Co-browse instrumentation after either putting the modified file in a new directory or updating the file's name.

For example, consider the case where you have set up your own server to host the Co-browse localization files for the master UI:

```
<script>
var _genesys = {
    cobrowse: {
        localization: '//example.com/cobrowse-l10n/2014-09-07/cobrowse-fr.json'
    }
};
</script>
<COBROWSE_INSTRUMENTATION_SCRIPT>
```

Next, you modify some of your localization files and want them to be refreshed for all users, so you create a new directory and update your Co-browse instrumentation:

```
<script>
var _genesys = {
    cobrowse: {
        localization: '//example.com/cobrowse-l10n/2014-10-08/cobrowse-fr.json'
    }
};
</script>
<COBROWSE_INSTRUMENTATION_SCRIPT>
```

Now, any browsers that had cached the files will be reload the files and re-cache them.

Built-in localization

This section lists the default localization values and keys.

You can use the code snippets in this section to create your own localization files. To do so, copy and save the code snippet as a `.json` file.

> **Important**
>
> You must save your localization JSON files using UTF-8 encoding.

> **Tip**
>
> It is not necessary to list **all** the keys in your localization JSON file. In your JSON file, you can specify only the keys you wish to override. Any key not specified will use the default localization value.

Master UI

```
{
  "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.",
  "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.<br><br>
Supported browsers are: <ul><li><a target='_blank' href='http://www.google.com/chrome'>Google
Chrome</a></li><li><a target='_blank' href='http://www.firefox.com/'>Mozilla
Firefox</a></li><li><a target='_blank' href='http://microsoft.com/ie'>Internet Explorer 9 and
above</a></li><li><a target='_blank' href='https://www.apple.com/safari'>Safari 6 and
above</a></li></ul>",
  "chatIsAlreadyRunning": "Chat is already running on another page.",
  "modalYes": "Yes",
  "modalNo": "No"
```

```
}
```

Chat Widget

See Chat Widget JS API#Localization

Slave

```
{
  "invalidSessionID": "Session ID is invalid or has expired.",
  "navRefresh": "Refresh is pressed. Reloading page.",
  "navBack": "Back is pressed. Reloading page.",
  "navForward": "Forward is pressed. Reloading page.",
  "youLeft": "You have left the session. Co-browse is now terminated.",
  "customerLeft": "Customer has left the session. Co-browse is now terminated.",
  "exitBtnText": "Exit Session",
  "sessionIdText": "Session ID: {sessionId}",
  "enterSessionIdText": "Session ID:",
  "modeWrite": "Mode: Write",
  "modePointer": "Mode: Pointer",
  "downgradeToPointer": "Downgrade to pointer mode",
  "navigationDenied": "Navigation request has failed.",
  "maskedNodeTitle": "This content is visible only to customer",
  "unsupportedNodeTitle": "Some data is missing due to Co-browse limitations"
}
```

Attached Data Overview

Co-browse does not have a dedicated type of interaction. The following Co-browse session data is appended to the primary Chat or Voice interaction when an agent joins a session:

Attached Data	Data Type	Value
CoBrowseAssociated	[str]	The initial value is "Yes".
CoBrowseSessionsQuantity	[int]	A value equal to the number of Co-browse sessions for the current interaction.
CoBrowseSessionId	[str]	A value equal to the unique session ID.
CoBrowseStartTime	[str]	An epoch time specified in milliseconds. It matches the start of the first Co-browse session for the current interaction.

When an agent exits a Co-browse session, some existing data is changed and new data is appended to the primary interaction:

Attached Data	Data Type	Value
CoBrowseAssociated	[str]	The value is changed to "No".
CoBrowseDuration	[int]	The duration in seconds is added. It summarizes the duration of all Co-browse sessions for the current interaction.
CoBrowseEndTime	[str]	An epoch time specified in milliseconds, is added. It matches the end of the last Co-browse session for the current interaction.

If there is more than one Co-browse session for the primary interaction, the following attached data is changed with each new Co-browse session:

Attached Data	Data Type	Value
CoBrowseAssociated	[str]	The value is "Yes".
CoBrowseSessionsQuantity	[int]	A value equal to the number of Co-browse sessions for the current interaction.
CoBrowseSessionId	[str]	A value equal to the unique session ID.

Using Attached Data for Reporting

Almost all attached data is used in the statistics and filters for Stat Server Application configuration. The following statistics and filters are included in the `StatProfile.cfg` file (located in the Genesys Co-browse Sample Reporting Templates root directory):

Statistics

```
[Agents_CurrentNumber]
Category=CurrentNumber
Description=Current number of agents working with interactions
MainMask=CallInternal,CallConsult,CallInbound
Objects=GroupAgents,GroupPlaces
Subject=DNStatus

[Agents_CurrentNumber_Inbound]
Category=CurrentNumber
Description=Current number of agents working with inbound interactions
MainMask=CallInbound
Objects=GroupAgents,GroupPlaces
Subject=DNStatus

[Agents_MaxNumber]
Category=MaxNumber
Description=Max number of agents worked with interactions
MainMask=CallInternal,CallConsult,CallInbound
Objects=GroupAgents,GroupPlaces
Subject=DNStatus

[Agents_MaxNumber_Inbound]
Category=MaxNumber
Description=Max number of agents worked with inbound interactions
MainMask=CallInbound
Objects=GroupAgents,GroupPlaces
Subject=DNStatus

[CurrentExAgentState]
Category=CurrentState
MainMask=*
Objects=Agent
Subject=DNAction

[Interactions_CurrentHandling]
Category=CurrentNumber
Description=Current number of interactions handling
MainMask=InteractionHandling
Objects=Agent,GroupAgents,GroupPlaces,Place
Subject=DNAction
```

```
[Interactions_Current_Inbound]
Category=CurrentNumber
Description=Current number of inbound interactions handling
MainMask=CallInbound
Objects=Agent,GroupAgents,GroupPlaces,Place
Subject=DNAction

[Interactions_TotalHandled]
Category=TotalNumber
Description=Total number of interactions handled
MainMask=InteractionHandling
Objects=Agent,GroupAgents,GroupPlaces,Place
Subject=DNAction

[Interactions_Total_Inbound]
Category=TotalNumber
Description=Total number of inbound interactions handled
MainMask=CallInbound
Objects=Agent,GroupAgents,GroupPlaces,Place
Subject=DNAction

[Interactions_TotalDuration]
Category=TotalTime
Description=Total time of interactions handled
MainMask=InteractionHandling
Objects=Agent,GroupAgents,GroupPlaces,Place
Subject=DNAction
```

Filters

```
[Filters]
Chat_Co-browse=PairExists("CoBrowseAssociated","*") & PairExists("MediaType","chat")
Chat_Co-browseAlive=PairExists("CoBrowseAssociated","Yes") & PairExists("MediaType","chat")
ChatInteraction=PairExists("MediaType","chat")
Co-browse=PairExists("CoBrowseAssociated","*")
Co-browseAlive=PairExists("CoBrowseAssociated","Yes")
VoiceCallInteraction=PairExist("MediaType", "voice") | (MediaType=voice)
Voice_Co-browse=PairExists("CoBrowseAssociated","*") & (PairExists("MediaType","voice") |
(MediaType=voice))
Voice_Co-browseAlive=PairExists("CoBrowseAssociated","Yes") & (PairExists("MediaType","voice") |
(MediaType=voice))
CoBrowseSessionId=PairExists("CoBrowseSessionId","*")
CoBrowseStartTime=PairExists("CoBrowseStartTime","*")
CoBrowseEndTime=PairExists("CoBrowseEndTime","*")
CoBrowseSessionsQuantity=PairExists("CoBrowseSessionsQuantity","*")
```

These statistics and filters are specified in the real-time and historical templates located in the "runtime" and "historical" folders. For details, see the Co-browse Reporting Templates.

Currently, the following attached data is not used for Reporting purposes:

- CoBrowseDuration

You can customize the Co-browse Reporting Templates and the statistics and filters to suit your needs. You can also create your statistics and filters using Co-browse-related data or any other data attached to the Primary interaction. The statistics and filters you create can then be used in the existing reporting templates or you can create your own templates. For details, refer to the following documents:

- Stat Server 8.1 User's Guide
- Reporting 8.0 CCPulse+ Administrator's Guide

Using Attached Data to Customize Business Processes

You can use Co-browse-related attached data to organize Business Processes against Chat or Voice interactions. For details, please refer to the following documents:

- eServices 8.1 User’s Guide
- Universal Routing 8.1 Strategy Samples
- Universal Routing 8.1 Business Process User's Guide