API Reference

Configuration API

4/28/2025

Contents

Configuration API

This API configures Co-browse and its integration with other media. It is also used to subscribe to the main Co-browse JavaScript API.

Co-browse is configured via a global _genesys variable. To configure Co-browse, create a <script> such as the following example and add it to your instrumentation:

```
<script>
var _genesys = {
    cobrowse: {
        /* Co-browse configuration options */
    }
};
</script>
<INSTRUMENTATION_SNIPPET>
```

> **Important**
>
> Co-browse is designed to make configuration optional. If any configuration options are not present, Co-browse will use the pre-defined default values.

> **Warning**
>
> _gcb has been discontinued.

Accessing the Co-browse APIs

Since the main Co-browse JavaScript file is added to the page asynchronously, you cannot instantly access the Co-browse APIs. Instead, you must create a function that will accept the APIs as an argument. There are two approaches to creating this function.

You can assign the function to the special property of a global configuration variable:

```
<script>
var _genesys = {
    onReady: function(APIs) {
        APIs.cobrowse // Co-browse API
    }
};
</script>
<INSTRUMENTATION_SNIPPET>
// or
<script>
```

```
var _genesys = {
    cobrowse: {
        onReady: function(cobrowseApi) { ... }
    }
};
</script>
<INSTRUMENTATION_SNIPPET>
```

Alternatively, you can modify configuration to make the APIs accessible at any point in your application through a _genesys global variable.

To do this, you must first assign an array to the onReady property:

```
<script>
var _genesys = {
    onReady: []
};
</script>
<INSTRUMENTATION_SNIPPET>
// or
<script>
var _genesys = {
    cobrowse: {
        onReady: []
    }
}
</script>
<INSTRUMENTATION_SNIPPET>
```

You can then obtain the APIs at any point in your application using the following code snippet:

```
_genesys.onReady.push(function(APIs) {
    APIs.cobrowse // Co-browse API
});
// or
_genesys.cobrowse.onReady.push(function(cobrowseApi) { ... });
```

> **Tip**
> For more information on the <INSTRUMENTAITON_SNIPPET>, see Basic instrumentation.

Disabling Co-browse

You can disable Co-browse using JS instrumentation. To do that, pass the value `false` to the respective configuration subsection:

```
<script>
var _genesys = {
    cobrowse: false
};
</script>
```

Configuring the Co-browse Button

You can hide the Co-browsing button. This might be useful if you want to start co-browsing from your own custom button (or from any other element or event), using the Co-browse API.

```
 <script>
var _genesys = {
buttons: {
cobrowse: false
}
};
</script>
```

You can change the position of the Co-browsing button:

```
<script>
var _genesys = {
buttons: {
position: 'right'
}
};
</script>
```

By default, the position is left and the button is visible.

Providing Custom HTML for Buttons

You can also pass functions that return HTML Element to `buttons.cobrowse`. In this case the output of the function will be used to render the button instead of using default image.

> **Tip**
> In this case your custom button(s) will inherit the positioning of the default button(s).

Here's a simple example that makes use of jQuery library to generate HTML Elements:

```
function createCustomButton() {
    return jQuery('<div class="myButtonWrapper"><button class="myButton">Co-
browse!</button></div>')[0];
}

var _genesys = {
    buttons: {
        cobrowse: createCustomButton
    }
};
```

> **Important**

> Note that is NOT mandatory to use jQuery in order to provide a custom HTML element. The example above does return an HTML element out of a jQuery object by retrieving the first element from jQuery collection via `[0]`.

Localizing the Co-Browsing Button

By default the buttons are images and therefore they cannot be localized in the same way as the rest of the interface. To localize these buttons, you can use one of the two following methods:

- Provide custom localized buttons instead of the default ones, as explained in Providing Custom HTML for Buttons.
- Override the appearance of the buttons using CSS.

For more information about localizing Co-browse, see Localization.

Co-browse Configuration Options

> **Tip**
> For backward compatability with previous versions of Co-browse, the name of the global configuration variable can also be `_gcb`. The use of `_gcb` is deprecated and may be discontinued in later versions. If you are using `_gcb`, we recommend that you switch to `_genesys`.

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

enableStaticResourceService

Default: `true`

Set to `true` to enable the Static Resources Behind Authentication feature, which will cache resources. Setting to `false` disables the feature.

Example:

```
<script>
var _genesys = {
      cobrowse: {
            enableStaticResourceService: true;
      }
};
</script>
```

debug

Default: `false`

Set to `true` to enable debugging console logs. You can enable full Co-browse logs, using this example:

```
<script>
var _genesys = {
      debug: true;
};
</script>
```

Or reduced Co-browse logs, as shown below:

```
<script>
var _genesys = {
      cobrowse: {
            debug: true;
      }
};
</script>
```

For debugging purposes, we recommend the first example. However, this example can also turn on logging from other Genesys tools, if you are using any and if you configure them using the _genesys variable.

disableBuiltInUI

Default: `false`

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

Example:

```
var _genesys = {
    cobrowse: {
        disableBuiltInUI: true
    }
};
```

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

> **Important**
>
> Co-browse is always trying to ensure that the customer is either on the phone or on a chat with the agent. If Co-browse cannot automatically detect this, it asks the customer via the UI. If you want to disable these UI dialogs before the start of a Co-browse session, you can implement an external media adapter with the Co-browse JavaScript External Media API.

primaryMedia

Default: null

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

Example:

```
<script>
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 */ }
};
</script>

<script>
var _genesys = {
    cobrowse: {
        primaryMedia: myPrimaryMedia
    }
};
</script>
<INSTRUMENTATION SNIPPET>
```

See External Media Adapter API for more details.

> **Warning**
> 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.

CSS

Default: Server synchronization strategy, `{server: true}`

This option manages the CSS synchronization strategy. Additional CSS synchronization on top of DOM synchronization allows you to **replay** style changes that occur when the user moves his or her mouse over an element with a `:hover` style rule.

[+] Additional details

For example, if you have the following CSS, Co-browse CSS synchronization makes the underlining visible to the agent when the consumer moves her mouse over a link, and vice versa, the underlining will be visible to the user when the agent moves the mouse over a link:

```
a:hover {
    text-decoration: underline;
}
```

Server strategy is the default and preferred setting. The server strategy setting allows the Co-browse server to proxy every CSS resource, including inline CSS. This strategy synchronizes CSS hover effects regardless of the domain the CSS resource is loaded from.

Example:

```
<script>
var _genesys = {
    cobrowse: {
        css: {
          server: true
        }
    }
};
</script>
```

> **Important**
> If the `css` option is not specified, the Co-browse JavaScript application behavior is equivalent to the configuration snippet above.

> **Warning**
> There are limitations on handling invalid CSS. This may lead to partial or complete loss of hover synchronization. It may also cause partial failure of general style synchronization. See Troubleshooting CSS Synchronization for details.

maxOfflineDuration

Default: `600` (seconds)

This option specifies the time in seconds that a reference to a Co-browse session is stored after page load. The default value is 600 seconds (10 minutes). If this period expires, the Co-browse session will end by time out.

> **Important**
> If you modify this option, it must match the same option on the server, maxInterval Option.

You can set this option, as shown in this example:

```
<script>
var _genesys = {
    cobrowse: {
       maxOfflineDuration: 300;
    }
};
</script>
```

> **Important**
> Setting this option using the field of the `_genesys` variable (as shown below) is deprecated. You must use the `cobrowse` section of `_genesys` variable (as shown above).
>
> Deprecated version:
>
> ```
> <script>
> var _genesys = {
> maxOfflineDuration: 300
> };
> </script>
> ```

disableWebSockets

Default: `false`

Use this option if you need to disable WebSocket communication such as when your load balancer does not support WebSockets and you do not want to wait for Co-browse to automatically switch to another transport.

> ### Important
> Due to the highly interactive nature of Co-browse, we highly recommended you do **not** disable WebSockets. We recommend that you configure your load balancers/ proxies infrastructure to support WebSockets. Disabling WebSockets may have a huge impact on Co-browse performance.

You can set this option, as shown in this example:

```
<script>
var _genesys = {
    cobrowse: {
       disableWebSockets: true;
    }
};
</script>
```

> ### Important
> Setting this option using the field of the _genesys variable (as shown below) is deprecated. You must use the `cobrowse` section of the _genesys variable (as shown above).
>
> Deprecated version:
>
> ```
> <script>
> var _genesys = {
> disableWebSockets: true;
> };
> </script>
> ```

localization

Default: null

Use this option to localize Genesys Co-browse. For a detailed description, see Localization.

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`.

Available since Co-browse JavaScript version `8.5.002.02`. For your Co-browse JavaScript version, see the VERSION property.

> **Important**
>
> 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.

Example:

```
<script>
// Turn on setting document.domain:
var _genesys = {
    cobrowse: {
        setDocumentDomain: true
    }
};
</script>
```

disableBackForwardCache

Default: `true`

Available since Co-browse **8.5.1**.

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

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

Example:

```
<script>
// Turn BackForward Cache back on:
var _genesys = {
    cobrowse: {
        disableBackForwardCache: false
```

```
    }
};
</script>
```

cookieFootprintReduce

Default: `false`

Set to `true` to enable cookie footprint reduce feature which allows to store session between page reload and relocation into site storage instead of cookies.

Example:

```
<script>
var _genesys = {
      cobrowse: {
            cookieFootprintReduce: true
      }
};
</script>
```