API Reference

Configuration API

5/6/2025

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 and the Chat API.

> **Important**
>
> Co-browse JavaScript is included in the Integrated JavaScript Application. You can configure Co-browse by modifying the JavaScript included in your webpages, also known as instrumentation. Before continuing, make sure you have read and understood Website Instrumentation#Basic Instrumentation and Integrated JavaScript Application#Configuration

> **Important**
>
> Configuration is **optional**. If any configuration options are not present, Co-browse will use pre-defined defaults.

> **Tip**
>
> To configure integration with chat, see Integrated JavaScript Application#Configuration The `integration` section of the configuration API is deprecated and support may be discontinued in later version of the integrated JavaScript Application.

Accessing the Co-browse and Chat APIs

> **Tip**
>
> This section is a quick start guide. To better understand API access, read Integrated JavaScript Application#Obtaining Chat and Co-browse APIs.

Since the main Co-browse JavaScript file is added to the page asynchronously, you cannot instantly access the Co-browse and Chat 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
        APIs.chat     // Chat widget API
    }
};
</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>
```

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
    APIs.chat     // Chat widget API
});
```

> **Tip**
>
> For more information on the <INSTRUMENTAITON_SNIPPET>, see Web Site Instrumentation#Basicinstrumentation.

Co-browse Configuration Options

Example configuration with default values:

```
var _genesys = {
    // defaults:
    cobrowse: {
        disableBuiltInUI: false,
        primaryMedia: undefined,
        css: {
          server: true
        },
        onReady: function(cobrowseApi) {}
    }
};
```

> **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`:

`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.

`primaryMedia`

Default: Built-in chat

Used to pass an object implementing an external media adapter interface. By default, the built-in chat is used.

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.

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;
}
```

There are two strategies for CSS synchronization, `server` and `browser`.

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.

Browser strategy is also available but it is a legacy setting and has the following limitations:

- Browser strategy cannot synchronize CSS hover effects if the CSS was loaded from another domain or sub-domain.
- Browser strategy cannot properly handle browser specific CSS in `:hover` rules.

  Example:

  ```
  #menu {
    background: #FFFFFF;
  }
  #menu:hover {
    background-image: -webkit-linear-gradient(top, #444444, #999999); /* Chrome 10-25, iOS
  5+, Safari 5.1+ */
    background-image:         linear-gradient(to bottom, #444444, #999999); /* Chrome 26,
  Firefox 16+, IE 10+, Opera */
  }
  ```

  If you have the CSS above while using browser CSS synchronization and the agent and user have different browsers, the menu will be kept white (first CSS rule) for the slave when the consumer moves his or her mouse over the menu.

- Browser strategy does not support the **remote execution** mode of applying hover effects. Consider the case where the agent moves his or her mouse over a menu item and a sub-menu is shown by CSS behavior. In server mode, the sub-menu will be shown on the master side **first** and **only then** propagated to the slave. The agent *will not see anything the user has not already seen.* In browser mode, the agent will see the change **immediately** and only then will it be propagated to the master side.

Example browser strategy configuration:

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

Both strategies may be activated at the same time. For a few cases, this may result in better content synchronization.

Example:

```
<script>
var _genesys = {
```

```
    cobrowse: {
        css: {
          browser: true,
          server: true
        }
    }
};
</script>
```

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.

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.