Developer's Guide

Integrating Web Engagement and Co-browse with Chat

4/23/2025

Integrating Web Engagement and Co-browse with Chat

Contents

The Integrated JavaScript Application provides the functionality of Web Engagement monitoring, Co-browse, and Chat in one easy to configure JavaScript application, rather than using the individual applications for each component.

The Integrated JavaScript Application is a JavaScript file that contains the Chat, Tracker, and Co-browse JavaScript applications, as well as code for their integration.

The integration consists of the following:

- For Chat and Tracker: The **pageID** and **visitID** are automatically attached to the chat session's **userData** when the chat session is started (either via the "Live Chat" button or the Chat JS API).
- For Chat and Co-browse: The application automatically detects if the agent is connected via chat and, if yes, the agent automatically joins the Co-browse session when it is started.

The physical integrated application file, named **genesys.min.js**, contains the pre-integrated Chat, Tracker and Co-browse JavaScript applications.

> ### Tip
> Another form of the app (**gcb.min.js**) is only shipped as part of the Co-browse solution and contains pre-integrated Chat and Co-browse (no Tracker).

> ### Important
> To successfully integrate Chat and Co-browse when chat is configured to operate in "popup" mode, you must host **chatWidget.html** in the same domain as the website (subdomain is also possible).

To use the Integrated Application in your Web Engagement or Co-browse solution, review the information on this page and add the instrumentation snippet to your website, along with any necessary configuration (this can vary depending on your solution — see Configuration for details.

Instrumentation Snippet

> ### Important
> The JavaScript files are obfuscated and minified. You are not allowed to decompile and/or modify them. If you do, support can not be guaranteed. Instead, you can use the public JavaScript APIs and other documented methods to customize the functionality.

You can activate the integrated functionality on a website by inserting the following snippet before the closing </head> tag:

```
<script>(function(d, s, id, o) {
  var fs = d.getElementsByTagName(s)[0], e;
  if (d.getElementById(id)) return;
  e = d.createElement(s); e.id = id; e.src = o.src;
  e.setAttribute('data-gcb-url', o.cbUrl);
  fs.parentNode.insertBefore(e, fs);
})(document, 'script', 'genesys-js', {
  src: "<SERVER_URL>/genesys.min.js",
  cbUrl: "<COBROWSE_SERVER_URL>/cobrowse" // this line is required only if Co-browse is used
});</script>
```

This script asynchronously (which means the loading won't block your site performance) loads and executes the required JavaScript file.

You should only modify the following (except for the special case of changing global variable names, described below) lines:

- `src: "<SERVER_URL>/genesys.min.js",` — This defines the **src** (the URL) of the script that is loaded and executed. You can load the script from the GWE Frontend Server, the Co-browse Server, or your own server:
  - To load the script from the Web Engagement Frontend Server, the URL format should be `http(s)://`*`FRONTEND_HOST`*`[:`*`FRONTEND_PORT`*`]/frontend/resources/js/build/genesys.min.js`
  - To load the script from the Co-browse Server, the URL format should be `http(s)://`*`COBROWSE_HOST`*`[:`*`COBROWSE_PORT`*`]/cobrowse/js/genesys.min.js`
  - To load the script from one of your own servers, use one of the above URLs to download the file and then copy it to your server. If you choose this option, make sure to configure the caching properly (see Note on Caching for details).
- `cbUrl: "<COBROWSE_SERVER_URL>/cobrowse` — This line is only required if you use Co-browse. It defines the URL that is used by the Co-browse JavaScript to get and receive Co-browse-related data.

The Co-browse URL is also used by chat to connect to the Genesys infrastructure via the Co-browse server. If you remove it, make sure to configure the serverURL option for chat, otherwise chat will not work. Also, be sure to remove the trailing comma from the `src: "<SERVER_URL>/genesys.min.js",` line so that your script looks like this:

```
<script>(function(d, s, id, o) {
  var fs = d.getElementsByTagName(s)[0], e;
  if (d.getElementById(id)) return;
  e = d.createElement(s); e.id = id; e.src = o.src;
  e.setAttribute('data-gcb-url', o.cbUrl);
  fs.parentNode.insertBefore(e, fs);
})(document, 'script', 'genesys-js', {
  src: "<SERVER_URL>/genesys.min.js"
});</script>
```

Note on Caching

> **Important**
>
> If you choose to serve static resources (JavaScript) on your servers, you should implement the analogous caching strategy to achieve best performance and minimum traffic load.

All static resources (JavaScript in our case) are served with caching HTTP headers when loaded from the Web Engagement Frontend Server or Co-browse Server. Both servers use the combination of HTTP headers that lead to the following caching workflow:

- When the client (browser) receives the resource, it stores it on disk for a configured time interval.
- During this time interval, if the resource is requested, the browser loads it from disk without sending any requests to the server (which speeds up the initialization of the scripts).
- After the time interval expires, the browser requests the resource again from the server. Then
  - if the resource has not changed since the previous request, the server replies with an empty response with 304 Not Modified status, to minimize the traffic. The browser then caches the resource on disk for yet another configured time interval.
  - if the resource has changed since the previous request, the server replies with a new version of the resource. The browser, again, caches the resource on disk for a configured time interval.

The default time interval for both servers is 30 minutes.

Configuration

Configuration for the integrated application (except for Tracker, see Configuring Tracker) is stored as a JavaScript object assigned to a global **_genesys** variable. This variable should be accessible to **genesys.min.js** when it is loaded. So the entire instrumentation might look like this:

```
<script>
var _genesys = {/* configuration goes here*/};
</script>
<INSTRUMENTATION_SNIPPET>
```

> **Important**
>
> For backwards compatibility with previous versions of Co-browse, the name of the global configuration variable can also be **_gcb**. This is deprecated and may be discontinued in later versions, so it is recommended that you switch to **_genesys** now if you're using **_gcb**.

Disabling Services

You may encounter cases where you want to disable the Integrated Application and its services based on some specific critieria. In this case, you can use a global variable to enable or disable services.

For example, if we create a global `enableGenesys` variable, we can enable Genesys services on the page when it is set to `true` and disable servies when the variable is set to false.

```
<script>
var enableGenesys = true; // or false
<script>
```

The configuration snippet would look like this:

```
var _genesys = {
  // custom options
};
if (!enableGenesys) {
  // overwrite cobrowse/chat options
  _genesys.chat = false;
  _genesys.cobrowse = false;
}
```

The idea is to disable a service by overriding its configuration with `false` when `enableGenesys` is `false`.

Changing the "_genesys" name

You can actually store the configuration in any global variable, **_genesys** is just the default convention. To tell the application that the configuration is stored in another variable, you have to modify the instrumentation snippet by adding a line there:

```
e.setAttribute('data-cfg-var', 'myCustomVariableName');
```

For example:

```
<script>
var _myCustomConfiguration =
    debug: true
};
</script>
<script>(function(d, s, id, o) {
  var fs = d.getElementsByTagName(s)[0], e;
  if (d.getElementById(id)) return;
  e = d.createElement(s); e.id = id; e.src = o.src;
  e.setAttribute('data-gcb-url', o.cbUrl);
  // Use _myCustomConfiguration variable as configuration (don't forget the quotes!):
  e.setAttribute('data-cfg-var', '_myCustomConfiguration');
  fs.parentNode.insertBefore(e, fs);
})(document, 'script', 'genesys-js', {
  src: "<SERVER_URL>/genesys.min.js"
  cbUrl: "<COBROWSE_SERVER_URL>/cobrowse"
});</script>
```

Common Options

The following options are shared between services. They are shared by Chat and Co-browse. For

Tracker, see Configuring Tracker. These options can be set as direct properties of an object assigned to the _genesys variable:

```
var _genesys = {
    <OPTION>: <VALUE>
};
```

If an option is set as in the example above, the option will be *inherited* by both Chat and Co-browse. It is also possible to set an option for only one service or to set an option *globally* and override that option for a particular service.

Examples:

```
// Set the option for all services:
var _genesys = {
    <OPTION>: <VALUE>
};

// Set the option only for Chat:
var _genesys = {
    chat: {
        <OPTION>: <VALUE>
    }
};

// Set the option for all services, but override for Co-browse:
var _genesys = {
    <OPTION>: <VALUE_1>,
    cobrowse: {
        <OPTION>: <VALUE_2>
    }
};
```

debug

The debug option is set to `false` by default. To enable debug output to the browser console log, set it to `true`.

```
var _genesys = {
    debug: true
};
```

> **Important**
>
> This option is not valid for the Tracker application. For details about configuring debug for the Tracker application, see Tracker Application Advanced Configuration.

disableWebSockets

Default: `false`

Set this option to `true` to disable Web Sockets. See corresponding Chat option and Co-browse option for more information on the purpose and impact of this option.

```
// Example: disable WebSockets for Chat and Co-browse (not recommended)
var _genesys = {
    disableWebSockets: true
};

// Example: disable WebSockets for Chat, but enable for Co-browse
var _genesys = {
    chat: {
        disableWebSockets: true
    }
};
```

> **Tip**
>
> When used with Chat, this option is automatically passed from configuration to `startChat()` and `restoreChat()`.

> **Important**
>
> This option is ineffective for Tracker. See Configuring Tracker for information on configuring Tracker.

Configuring Buttons

The **_genesys.buttons** section allows some basic configuration of the "Live Chat" and "Co-browsing" buttons. It has three optional properties:

- **position**: Can be either "left" (default) or "right"
- **cobrowse**: Defaults to true
- **chat**: Defaults to true

Note that you can override only the properties that you want to be changed. Other properties are used with their default values. For example this configuration:

```
var _genesys = {
    buttons: {
        chat: false
    }
};
```

actually means this:

```
var _genesys = {
    buttons: {
        chat: false,
        cobrowse: true, // inherited default
        position: 'left' // inherited default
    }
};
```

Disabling Buttons

As seen in the snippet above, you can pass `false` to disable the "Co-browsing" and/or "Live Chat" button. This might be useful if you want to start chat or co-browsing from your own custom button (or from any other element or event), using the Co-browse API or Chat Widget JS API.

Providing Custom HTML for Buttons

You can also pass a **function** that returns HTML elements to **buttons.cobrowse** or **buttons.chat**. In this case, the output of the function is used to render the button instead of default image.

Note that in this case your custom button(s) inherit the positioning of the default button(s).

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

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

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

> **Important**
>
> jQuery is NOT mandatory to use 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].

Configuring Tracker

In the current version of the Integrated JavaScript Application, the Genesys Web Engagement Tracker Application is configured in its traditional way, via the **global _gt** (or other, if configured) variable. See Tracker Application for details.

This means that the full instrumentation might look like this:

```
<script>
// Configure tracker:
var _gt = window._gt || [];
_gt.push(['config', {
  dslResource: <DSL_RESOURCE>,
  httpEndpoint: <HTTP_ENDPOINT>,
  httpsEndpoint: <HTTPS_ENDPOINT>
}]);

// Configure integrated application:
var _genesys = { /* Integrated application, Chat and Co-browse configuration */ };
</script>

<INSTRUMENTATION_SNIPPET>
```

Changing the "_gt" Variable Name

If you use **genesys.min.js** to include the Tracker Application onto your page, and want to modify the name of the variable that Tracker is exported to, you must add the following line to the instrumentation snippet:

```
e.setAttribute('data-gpe-var', '<NAME_OF_THE_VARIABLE>');
```

For example, let's export Tracker to the **_myTracker** variable:

```
<script>(function(d, s, id, o) {
  var fs = d.getElementsByTagName(s)[0], e;
  if (d.getElementById(id)) return;
  e = d.createElement(s); e.id = id; e.src = o.src;
  e.setAttribute('data-gcb-url', o.cbUrl);
  e.setAttribute('data-gpe-var', '_myTracker'); // note the quotes around variable name
  fs.parentNode.insertBefore(e, fs);
})(document, 'script', 'genesys-js', {
  src: "<SERVER_URL>/genesys.min.js"
  cbUrl: "<COBROWSE_SERVER_URL>/cobrowse"
});</script>
```

Using External Tracker

It is possible to use the integrated application with an external Tracker application (that is, a Tracker application loaded from another script).

This might be useful if you have configured a Tracker application and want to use it with **gcb.min.js** (provided by Co-browse solution) instead of loading Tracker from **genesys.min.js** (although this setup is not recommended).

To do that, pass a reference to the external tracker to **_genesys.tracker**:

```
var _genesys = {
    tracker: _gt
};
```

The passed external Tracker is integrated with the chat widget.

Configuring Chat

Configuration for chat is stored in the **chat** subsection of the global configuration object:

```
var _genesys = {
    chat: {/* chat configuration */}
};
```

Configuring the Server URL

The main thing you might want to configure for chat is the URL of the server.

In most cases the server here is the Web Engagement Backend. Use the template below to construct the URL:

```
var _genesys = {
    chat: {
```

```
        serverUrl: 'http(s)://<BACKEND_HOST>[:<BACKEND_PORT>]/backend/cometd'
    }
};
```

> **Important**
>
> If you use Co-browse, you can use Co-browse Server for chat. In this case, you don't have to configure the **serverUrl** option explicitly. The **cbUrl** option in the instrumentation snippet is used to automatically create the proper URL to connect chat to the Genesys infrastructure via Co-browse Server.

Disabling Chat

You can disable the built-in chat completely by passing false to **_genesys.chat**.

```
var _genesys = {
    chat: false
};
```

In this case, the "Live Chat" button is also disabled (it is not added to the page). If you want to disable chat and to enable the "Live Chat" button (for example, to bind your own chat widget to this button), you can do it by explicitly enabling the button in configuration (see Configuring Buttons):

```
var _genesys = {
    chat: false,
    buttons: {
        chat: true
    }
};
```

Now the button is added to the page, but clicking it does not open the chat widget.

> **Tip**
>
> Also see Disabling Services.

autoRestore

On every page reload / navigation, the chat widget is automatically restored if there is an ongoing chat session. You can disable this behavior with the **autoRestore** option, which is set to `true` by default. You might disable this behavior if you want more control over chat widget restoration or if you want to get access to the chat session service API.

```
<script>
var _genesys = {
    chat: {
        autoRestore: false,
        onReady: function(chat) {
            chat.restoreChat().done(function(session) {
                // Use chat session API here, e.g.:
```

```
                // session.sendMessage('hello world');
                // session.onAgentConnected(function(event) {...});
            });
        }
    }
};
</script>
```

> **Tip**
>
> See Obtaining Chat and Co-browse APIs if the **onReady** syntax above looks confusing to you.

> **Important**
>
> "Live Chat" and "Co-browsing" buttons appear only after **restoreChat** is called. So, if you set **autoRestore** to `false`, it becomes your code's responsibility to call **restoreChat**. If it is not called, buttons do not appear.

Chat Widget Options

All options (except for **autoRestore** and **onReady**) that are stored in the **_genesys.chat** object are automatically passed to chat the `startChat()`/`restoreChat()` methods. See Chat Widget JS API for the full list of options.

The integrated application provides some defaults for your convenience, so that minimal or no explicit chat configuration is required. The provided defaults are:

- **debug** is inherited from _genesys.debug
- maxOfflineDuration is aligned with Co-browse's maxOfflineDuration option and defaults to `600` seconds (10 minutes)
- serverUrl is set automatically to use the Co-browse Server (if Co-browse is used)

Configuring Co-browse

Co-browse configuration is stored in the **cobrowse** subsection of the global configuration object:

```
var _genesys = {
    cobrowse: {/* Co-browse configuration */}
};
```

See Co-browse Configuration API for the full list of options.

Disabling Co-browse

You can disable Co-browse completely by passing `false` to **_genesys.cobrowse**:

```
var _genesys = {
    cobrowse: false
};
```

In this case, the "Co-browsing" button is also disabled (not added to the page). If you want to disable Co-browse, but enable the "Co-browsing" button, you can do so by explicitly enabling the button in configuration (see Configuring Buttons):

```
var _genesys = {
    cobrowse: false,
    buttons: {
        cobrowse: true
    }
};
```

Now the button is added to the page, but clicking it does not start the Co-browse session.

> **Tip**
>
> Also see Disabling Services.

Localization of Chat and Co-browse

> **Important**
>
> The Tracker application does not have localization because it does not have a user interface.

The integrated application is shipped with English localization. You can configure localization by passing a URL to the **localization** option of the respective sub-module (**cobrowse** or **chat**). The URL must point to a valid JSON response hosted on a server with JSONP support. If localization is configured, you should implement proper caching of the localization resource to avoid delays in app initialization (see Note on Caching).

See Localization for more information.

Example:

```
var _genesys = {
    cobrowse: {
        localization: 'http://example.fr/cobrowse-fr.json'
    },
    chat: {
        localization: 'http://example.fr/cobrowse-chat-fr.json'
    }
};
```

Obtaining Chat and Co-browse APIs

> **Important**
>
> For the Tracker API, see the Tracker JS API.

> **Important**
>
> If you are using Chat as part of Web Engagement **GPE.js** (and not part of the Integrated Application), see Chat JS Application for information on Chat API.

Using onReady Callbacks

There are three "ready" events in the integration module which can be used to gain access to the APIs:

- "Main", or global, "ready" event which is fired after all the parts of the app have initialized. It provides access to both Chat and Co-browse APIs.
- Chat "ready" event.
- Co-browse "ready" event.

For each of the events, there is a dedicated **onReady** property in the configuration, which can be used to add callbacks for the event.

You can add subscriptions (callbacks) to any of these events via the mechanisms described below.

> **Tip**
>
> "ready" events are fired after the DOM is ready, so you don't have to wrap code that uses the provided APIs into jQuery(document).ready or similar constructions.

Subscribing to APIs using One Dedicated Function

Use this method if you want to provide one, and only one, subscription to a "ready" event.

To use it, simply assign a function to the **onReady** property of the configuration section:

```
<script>
    var _genesys = {
        onReady: function(APIs) {
            // Feel free to use the APIs here.
        }
    };
```

```
</script>
```

> **Tip**
> See "Main" onReady Callbacks for details about what **APIs** is in the example above.

Inside this function you can, for example, pass the provided arguments (the APIs) to your code so that it can be used multiple times there.

Also, if you need to use the APIs in different parts of your code, you can use an array as described in the next section.

Using an Array for Multiple Subscriptions to APIs

To use this method, you have to pass an array to the **onReady** property. This array may contain 0 or more subscription functions:

```
<script>
    var _genesys = {
        onReady: [function(APIs) {
            // Feel free to use the APIs here.
        }]
    };
</script>
```

Now you can add subscriptions using the **_genesys** global variable in any part of your code:

```
_genesys.onReady.push(function(APIs) {
    // Another use of the API here.
});
```

> **Tip**
> See "Main" onReady Callbacks for details about what **APIs** is in the example above.

> **Tip**
> If you **push** a callback after the respective "ready" event has already happened, the callback is called immediately.

To use the **.push(callback)** mechanism, you MUST pass an array to configuration, otherwise it is not guaranteed that the push method is always available.

For example, if you want to make use of the above push functionality for adding multiple subscriptions to each of the three **onReady** events, the minimum required configuration is this:

```
<script>
```

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

"Main" onReady Callbacks

> **Tip**
>
> See Using onReady Callbacks for detailed information about how you can add the callbacks.

These callbacks can be used to access the Co-browse API and/or the Chat API, and are also fired after the UI has been created. They can be used, for example, to attach custom handlers to the "Live chat" and "Co-browsing" buttons, add additional buttons, and so on.

All attached callbacks receive two arguments:

1. An object containing Chat (only in top context) and Co-browse APIs. APIs can be accessed via object properties:
   a. **.chat** for Chat Widget JS API
   b. **.cobrowse** for Co-browse API
2. A Boolean property indicating whether the code executes in the "top" context (`true`) or in an iframe (`false`). This is useful for Co-browse API users (see Co-browse in iframes).

Example:

```
_genesys.onReady.push(function(APIs, isTopContext) {
    // Check if we're in iframe:
    alert('We are ' + (isTopContext ? '' : 'not') + ' in an iframe');

    // Start a chat session:
    if (isTopContext) {
        APIs.chat.startSession();
    }

    // Mark an element as "service" to Co-browse (so that it won't be shown to agent):
    APIs.cobrowse.markServiceElement(document.getElementById('myCustomChatWidget'));
});
```

Chat onReady Callbacks

> **Tip**
> See Using onReady Callbacks for detailed information about how you can add the callbacks.

These callbacks are fired as soon as the Chat Widget JS API is available and they provide the same API methods the chat widget provides:

- `startChat()`
- `restoreChat()`

The only difference is that the provided methods use options from _genesys.chat configuration, so you don't have to pass options to them.

If you still need to pass options directly to `startChat()` or `restoreChat()` call, you can but the options are merged with options from configuration, and will take higher priority:

```
<!-- Suppose you have the following configuration: -->
<script>
var _genesys = {
    chat: {
        registration: false,
        embedded: false,
        onReady: []
    }
};
</script>

<!-- And then somewhere in your code: -->
<script>
_genesys.chat.onReady.push(function(chat) {
    chat.startChat({
        embedded: true
    });
});
</script>

<!-- The final options passed to startChat() will be: -->
{
    registration: false, // taken from configuration
    embedded: true // overriden by options from chat.startChat() call
}
```

Co-browse onReady Callbacks

> **Tip**

> See Using onReady Callbacks for detailed information about how you can add the callbacks.

These callbacks receive two arguments:

- **cobrowseApi**: Instance of the Co-browse API (you can name it **api**, **cobrowse** or any other name that is convenient to you).
- **isTopContext**: Boolean property indicating whether the code executes in the "top" context (`true`) or in an iframe (`false`). See Co-browse in iframes.

For example:

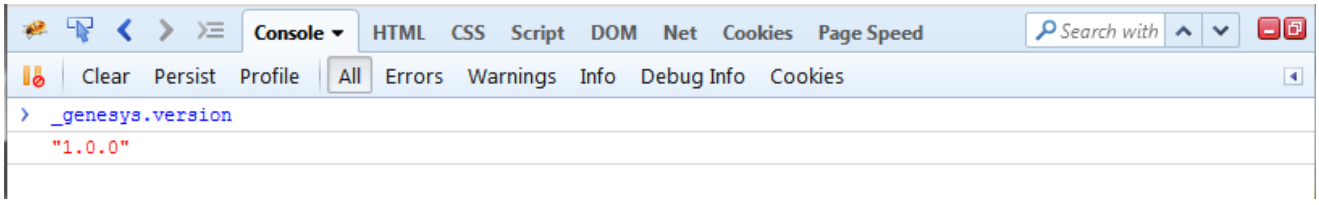
```
<script>
var _genesys = {
    cobrowse: {
        disableBuiltInUI: true,
        onReady: function(cobrowseApi, isTopContext) {
            createCustomCobrowseUI(cobrowseApi, isTopContext);
        }
    }
};
</script>
<INSTRUMENTATION SNIPPET>
```

Versions and Compatibility

The Integrated JavaScript Application has its own versioning; different versions of the application are compatible with different versions of Co-browse and Web Engagement.

The general rule is that the version of the integrated application shipped with a particular solution is compatible with that version of the solution.

To find out the version of the integrated application, see the value of `_genesys.version` (execute `_genesys.version` in the browser console) when the site is instrumented with the integrated application:



Compatibility Table

Note: The following table indicates which versions of Web Engagement and Co-browse are *compatible* with the indicated versions of the Integrated Application. It does not show which version of the Integrated Application is *shipped* with each version of Web Engagement and Co-browse.

Integrated Application version (_genesys.VERSION)	Web Engagement Server versions	Co-browse Server versions
1.0.0	8.1.200.38+	8.1.302.06+ up to, but not including 8.5 (Co-browse 8.5 is not supported)
850.0.X, 850.1.X	8.1.200.38+	8.5.XXX.XX