API Reference

Monitoring JS API

5/1/2025

Monitoring JS API

Contents

Description

You can use the Monitoring JS API in your web pages to send events (read more about how these events are structured here) to the Genesys Web Engagement Frontend Server. This is independent from the set of events and conditions that are defined in the DSL files that are loaded by the browser's Monitoring Agents. The design model for the Monitoring JS API is highly flexible, and its use can be extended well beyond the common model of user-triggered events — the design decision is up to you. You can submit `UserInfo`, `SignIn`, `SignOut`, and your own custom business events using this API.

> **Important**
> All commands and options are **case sensitive**.

The entry point for this API is the global `_gt` (Genesys Tracker) object which implements the `push()` method. The `_gt.push()` method takes an array as a parameter, which can contain any type of information, so longs as it follows this format:

```
_gt.push(['<commandName>', <options>])
```

- `<commandName>` — name of the event command.
- `<options>` — options for the command.

_gt.push(['event', options])

Sends business events to the Frontend Server.

Parameters

Parameter name	Type	Mandatory	Description
options	object	yes	A set of key/value pairs that represent event information.

Possible key/value pairs in "options"

Parameter name	Type	Mandatory	Description
eventName	string	yes	The identification name of the business event. This field is equivalent

Parameter name	Type	Mandatory	Description
			to the name parameter of the DSL <event> element.
<customParameter>	object	no	An object of additional key/value pairs to send along with the event.

Example with mandatory parameters

```
_gt.push(['event', {
        eventName: 'AddToCart'
}])
```

Example with additional parameters

```
_gt.push(['event', { eventName: 'AddToCart',
        productName : 'Sony',
        productModel:  'JVB72',
        productPrice:  '1000$'
}])
```

_gt.push(['sendUserInfo', options])

Genesys Web Engagement relies on your website to trigger the transitions between visitor states (see Visitor Identification).

This event sends the system "UserInfo" event with customer information. You should send this event when a user visits your website after closing the browser window on an authenticated session. For details, see Recognized Visitors.

Parameters

Parameter name	Type	Mandatory	Description
options	object	yes	A set of key/value pairs that represents event information.

Possible key/value pairs in "options"

Parameter name	Type	Mandatory	Description
userID	string	yes	The identification ID for the user. For instance, the user account name

Parameter name	Type	Mandatory	Description
			or the e-mail address.
<customParameter>	object	no	An object of additional key/value pairs to send along with event.

Example with mandatory parameters

```
_gt.push(['sendUserInfo',  {
        userID: 'user@genesyslab.com'
}])
```

Example with additional parameters

```
_gt.push(['sendUserInfo',  {
        userID: 'user@genesyslab.com',
        name:   'Bob',
        sex:    'male',
        age:    30
}])
```

_gt.push(['sendSignIn', options])

This event creates and sends the system "SignIn" event. Send this event when the user is authenticated by the website. This allows the system to identify the user and creates a new "session" with a `sessionID` that is unique to a visit and will last the duration of the visit. Only Authenticated visitors have an associated sessionId.

Parameters

Parameter name	Type	Mandatory	Description
options	object	yes	A set of key/value pairs that represents event information.

Possible key/value pairs in "options"

Parameter name	Type	Mandatory	Description
userID	string	yes	The identification ID for the user. For instance, the user account name or the e-mail address.
<customParameter>	object	no	An object of additional key/value pairs to send

Parameter name	Type	Mandatory	Description
			along with event.

Example with mandatory parameters

```
_gt.push(['sendSignIn',  {
        userID: 'user@genesyslab.com'
}])
```

Example with additional parameters

```
_gt.push(['sendSignIn',  {
        userID: 'user@genesyslab.com',
        name:   'Bob',
        sex:    'male',
        age:    30
}]);
```

_gt.push(['sendSignOut', options])

This event creates and sends the "SignOut" system event for the current user. This event should be sent if the user performs a logout on the website or as soon as possible after the logout action was done. **Note:** The `sessionId` lasts for the duration of the authenticated user's visit to your website. It is stored in a cookie and sent with every event that occurs between "SignIn" and "SignOut", and is changed automatically after every "SignIn" event.

Parameters

Parameter name	Type	Mandatory	Description
options	object	no	A set of key/value pairs that represents event information.

Possible key/value pairs in "options"

Parameter name	Type	Mandatory	Description
userID	string	no	The identification ID for the user. For instance, the user account name or the e-mail address. **Note:** Genesys recommends using this parameter even though it is not mandatory.

Parameter name	Type	Mandatory	Description
<customParameter>	object	no	An object of additional key/value pairs to send along with event.

Example with no parameter

```
_gt.push(['sendSignOut'])
```

Example with additional parameters

```
_gt.push(['sendSignOut', {
        userID: 'user@genesyslab.com'
}])
```

_gt.push(['getIDs', callback])

Gets visit identification information from the Web Engagement application. The callback contains an object with the `visitID`, `globalVisitID`, `pageID`, and `alias` fields.

Parameters

Parameter name	Type	Mandatory	Description
callback	function(IDs)	yes	A function that is called if the request succeeds. The function is passed one argument.

"IDs" parameter

Parameter name	Type	Mandatory	Description
IDs	object { `globalVisitID`, `visitID`, `pageID`, `alias` }	yes	An object that contains visit identification information.

Example

```
_gt.push(['getIDs', function(IDs) {
     console.log('IDs: ', IDs);
}])
```

How To — Enable a trigger after another trigger

The DSL is a great tool to create business events on your website without requiring programming knowledge, but it's not a JavaScript representation in XML. There are some use cases when DSL is not enough; instead, you should use the Monitoring JS API.

Let's look at this example use case:

You have a web page with a text field and a submit button. If a user starts typing in the text field and, for example, 100 seconds pass with no "sumbit" — you want to make sure that is reported to Web Engagement.

You can implement the functionality that's described in the use case with the following approach:

```
...
<p><input class="comment" type="text"></p>
<p><input class="submit" type="button" value="submit"></p>

<script>
    var timeout;

    $('.comment').focus(function() {
        if (!timeout) {
            console.log('timer started');
            timeout = setTimeout(function () {
                console.log('send event');
                _gt.push(['event', {eventName: 'myEvent'}])
            }, 100 * 1000)
        }
    });

    $('.submit').click(function() {
        if (timeout) {
            console.log('clean timeout');
            window.clearTimeout(timeout);
            timeout = undefined;
        }
    });
</script>
...
```