API Reference

Monitoring JS API

4/29/2025

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 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', eventName, { data: options }])

Sends business events to the server.

Parameters

Parameter name	Type	Mandatory	Description
eventName	string	yes	The name of the event. This field is equivalent to the name attribute of the DSL <event> element.
options	object	no	An object of any properties you want to track with the event.

Example with mandatory parameters

```
_gt.push(['event', 'AddToCart']);
```

Example with additional parameters

```
_gt.push(['event', 'AddToCart', { data: {
    productName : 'Sony',
    productModel: 'JVB72',
    productPrice: '1000',
    productCurrency:'USD'
}}]);
```

_gt.push(['event', 'UserInfo', { data: options }])

The Tracker application relies on your website to trigger transitions between visitor states (See Visitor Identification). This method sends a **UserInfo** system event that includes customer information. You should only send this event if your website has authenticated the user. You should also send this event when an authenticated session has been ended and the user returns to the website. For more information, 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 user's ID. For instance, the user account name or email address.
<customParameter>	boolean / number / string / object	no	An object containing additional key/value pairs for sending with the event.

Example with mandatory parameters

```
_gt.push(['event', 'UserInfo', { data: {
    userID: 'user@genesyslab.com'
}}]);
```

Example with additional parameters

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

_gt.push(['event', 'SignIn', { data: options }])

Creates and sends the `SignIn` event, which allows the system to identify the user. Send this event when the user is authenticated by your website.

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 user's ID. For instance, the user account name or email address.
<customParameter>	boolean / number / string / object	no	An object containing additional key/value pairs for sending with the event.

Example with mandatory parameters

```
_gt.push(['event', 'SignIn', { data: {
    userID: 'user@genesyslab.com'
}}]);
```

Example with additional parameters

```
_gt.push(['event', 'SignIn', { data: {
    userID: 'user@genesyslab.com',
    name:   'Bob',
    sex:    'male',
    age:    30
```

```
}}]);
```

_gt.push(['event', 'SignOut', { data: options }])

Creates and sends the `SignOut` event for the current user. This event should be sent at the time the user logs out from your website or as soon as possible after logout.

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 user's ID. For instance, the user account name or email address. **Note:** Genesys recommends that you use this parameter, even though it is not mandatory.

Example without parameters

```
_gt.push(['event', 'SignOut']);
```

Example with additional parameters

```
_gt.push(['event', 'SignOut', { data: {
    userID: 'user@genesyslab.com'
}}]);
```

_gt.push(['event', 'PageEntered', { data: options }])

Creates and sends the PageEntered event. The Tracker application sends a PageEntered event automatically when a new page is loaded. This method should only be used with a Single Page Application.

Notes

- The `PageEntered` event updates the current `pageID`.
- Make sure that a `PageExited` event is sent before sending a `PageEntered` event.
- The `PageEntered` event does not affect the DSL event sequence defined on initial page load. It also does not lead to categorization or to reinitialization of the DSL.

Parameters

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

Possible key/value pairs in "options"

Parameter name	Type	Default Value	Description
title	string	document.title	The title of the current page.

Example without parameters

```
_gt.push(['event', 'PageEntered']);
```

Example with additional parameters

```
_gt.push(['event', 'PageEntered', {
    data: {
        title: 'My Page Title'
    }
}]);
```

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

Creates and sends the PageExited event. The Tracker application sends a PageExited event automatically on page unload. This method should only be used with a Single Page Application.

Example without parameters

```
_gt.push(['event', 'PageExited']);
```

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

> **Important**
> Deprecated in 8.5

Sends business events to the Web Engagement 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 to the name attribute 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]) (Deprecated)

> **Important**
> Deprecated in 8.5

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 or the email 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]) (Deprecated)

> **Important**
> Deprecated in 8.5

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 email address.
<customParameter>	object	no	An object of additional key/value pairs to send 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]) (Deprecated)

> **Important**
> Deprecated in 8.5

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 email address. **Note:** Genesys recommends using this parameter even though it is not mandatory.
<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 Tracker application. The callback contains an object with the `visitID`, `globalVisitID`, `pageID`, and `alias` fields.

> **Important**
>
> The **alias** field is deprecated in 8.5 and will be removed in 9.0

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);
}])
```

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

Gets configuration information from the Tracker application. The callback contains an object with the

`visitID`, `httpEndpoint`, `httpsEndpoint`, and other fields.

Parameters

Parameter name	Type	Mandatory	Description
callback	function(config)	yes	A function that is called if the request succeeds. A single argument is passed to the function.

"config" parameter

Parameter name	Type	Mandatory	Description
IDs	object { `httpEndpoint`, `httpsEndpoint`, ... }	yes	An object that contains configuration information.

Example

```
_gt.push(['getConfig', function(config) {
    console.log('config: ', config);
    console.log('httpEndpoint: ' + config.httpEndpoint);
    console.log('httpsEndpoint: ' + config.httpsEndpoint);
}]);
```

Events

You can use handler functions to register behaviors to take effect when the Tracker application is generating events, and to further manipulate those registered behaviors.

Here is how to bind a handler function to a Tracker. The handler will be invoked whenever the event is fired:

```
_gt.push(['on', eventName, handler]);
```

To remove a previously-bound handler function from an object:

```
_gt.push(['off', eventName, handler]);
```

beforeSendEvent

```
_gt.push(['on', 'beforeSendEvent', handler])
```

```
_gt.push(['off', 'beforeSendEvent', handler])
```

The `beforeSendEvent` event is triggered before sending an event to the server.

Handler Arguments

Argument name	Type	Description
event	object	A set of key/value pairs that represents event information.

Example

```
var myEventHandler = function (event) {
    if (event.eventName === 'Search') {
        // Send event to Google Analytics service
        ga('send', 'event', 'search', event.eventName, event.data);
    }
};

// subscribe
_gt.push(['on', 'beforeSendEvent', myEventHandler]);
// unsubscribe
_gt.push(['off', 'beforeSendEvent', myEventHandler]);
```

`myEventHandler` is a function that you can execute each time the event is triggered but before it is sent to the server.

notificationMessage

```
_gt.push(['on', 'notificationMessage', handler]);

_gt.push(['off', 'notificationMessage', handler]);
```

The `notificationMessage` event is triggered when a message is received from the server.

Handler Arguments

Argument name	Type	Description
message	object	A set of key/value pairs that represents one notification message.

Example

The following example shows how to subscribe to the default message for starting a chat:

```
var notificationMessageHandler = function (message) {
    if (message.channel === 'gpe.setVariable' && message.data.value.type === 'chat') {
        alert('Chat from notificationMessageHandler');
    }
};

_gt.push(['on', 'notificationMessage', notificationMessageHandler]);
```

How To — Enable a trigger after another trigger

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 "submit" — 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>
...
```

Tracking Single Page Applications

A Single Page Application (SPA) is a web application or website that loads all of the resources required to navigate throughout an entire site when the first page on that site is loaded.

As the user clicks links and interacts in other ways with the page, any new content is loaded dynamically. The application may update the URL in the address bar to emulate traditional page navigation, but it never requests another full page.

By default, the Tracker Application works well with traditional websites, because it sends a **PageEntered** event every single time the user loads a new page. However, for an SPA where you're loading new page content dynamically rather than as full page loads, the Tracker script only sends the first **PageEntered** event—because all subsequent page entries are *virtual*. This means you have

to track these subsequent (virtual) **PageEntered** events manually, as each piece of new content is loaded.

To track dynamically loaded content as a distinct page, you can use the Tracker API to send a **PageEntered** event. To do this, you need to specify the URL and title, as shown here:

```
_gt.push(['event', 'PageEntered', {
    url: 'http://example.com/my-page-url?id=1',
    data: {
        title: 'My Page Title'
    }
}]);
```

If you specify a URL value in a **PageEntered** event, the app will only send that URL value to the server—it will not update the URL value stored in the Tracker application itself.

This means that if you send other events and don't explicitly include the current URL value, the Tracker application will associate those events with whatever URL was stored at the time of the initial page load.

To avoid this issue, it's usually best to update the Tracker app configuration data with the URL and page title from a newly loaded "page" before you send any other events for that "page." This will ensure that these events are associated with the correct page data.

To update the Tracker configuration, use the **config** command:

```
_gt.push(['config', {
    page: {
        url: 'http://example.com/my-page-url?id=1',
        title: 'My Page Title'
    }
}]);
```

Once the Tracker configuration has been updated with the proper data for the new "page," you can send a **PageEntered** event without overriding page-related parameters. For example:

```
_gt.push(['config', {
    page: {
        url: 'http://example.com/my-page-url?id=1',
        title: 'My Page Title'
    }
}]);

_gt.push(['event', 'PageEntered']);
/*
    event.url - 'http://example.com/my-page-url?id=1'
    event.data.title - 'My Page Title'
 */
```

Here is a more complex use case:

```
_gt.push(['config', {
    page: {
        url: 'http://example.com/my-page-url?id=1',
        title: 'My Page Title'
    }
}]);

_gt.push(['event', 'PageEntered']);
```

```
/*
    event.url - 'http://example.com/my-page-url?id=1'
    event.data.title - 'My Page Title'
 */

_gt.push(['event', 'PageEntered', {
    url: 'http://example.com/new-url'
}]);
/*
    event.url - 'http://example.com/new-url'
    event.data.title - 'My Page Title'
 */

_gt.push(['event', 'MyCustomEvent']);
/*
    event.url - 'http://example.com/my-page-url?id=1'
 */
```

Note: The second PageEntered event and all subsequent PageEntered events in a single-page application do not reset the timer for timeout events defined in the DSL.