Journey JavaScript SDK

8/8/2020

Table of Contents

Search the table of all articles in this guide, listed in alphabetical order, to find the article you need.

Article	Description
Article	Description

"> About Events methods Events methods allow developers to build functionality that reacts to state changes in the SDK."> About modules Learn about optional JavaScript files that enhance the functionality provided by the Journey JavaScript SDK."> About the tracking snippet Learn what happens after you deploy the tracking snippet on your web pages. Not what you're looking for? Try About tracking with Altocloud instead."> api.session.getCustomerCookieId Learn how to obtain a customer's cookie Id."> api.session.getData Learn how to obtain the data such as the short ID for a particular customer's session."> api.session.getId Learn how to get the Id of a particular session."> autotrackClick Learn how to configure which click events tracks on your websites, so you have more accurate page tracking information for use in segments and outcomes."> autoTrackClick Learn how to configure which click events Altocloud tracks on your websites, so you have more accurate page tracking information for use in segments and outcomes."> autotrackClick Learn how to configure which click events Altocloud tracks on your websites, so you have more accurate page tracking information for use in segments and outcomes."> autoTrackIdle Learn how to configure when Altocloud detects inactivity on a webpage, so you have more accurate page tracking information for use in segments and outcomes."> autotrackIdle Learn how to configure when Altocloud detects inactivity on a webpage, so you have more accurate page tracking information for use in segments and outcomes."> autotrackInViewport Learn how to configure which element Altocloud tracks on your websites as they appear and disappear from the viewport, so you have more accurate page tracking information for use in segments and outcomes."> autotrackScrollDepth Learn how to configure which scroll milestones Altocloud tracks on your websites, so you have more accurate page tracking information for use in segments and outcomes."> autoTrackUrlChange Learn how to customize tracking for Single Page Application (SPA) websites. Alternatively, use your preferred tag manager to customize and deploy the SPA tracking snippet. For more information, see About event tracking with tag managers. If you are just getting started, read Tracking snippet."> autotrackURLChange Learn how to customize tracking for Single Page Application (SPA) websites. Alternatively, use your preferred tag manager to customize and deploy the SPA tracking snippet. For more information, see About event tracking with tag managers. If you are just getting started, read Tracking snippet."> Advanced tracking with cookies Learn how to use the Journey JavaScript SDK to refine how Altocloud tracks customer data. Alternatively, you can refine tracking using your preferred tag manager. For more information, see About event

tracking."> Content offers lifecycle See the lifecycle of a content offer and the metrics that we capture at each state along the way. Lifecycle states are used in reporting and in determining the triggering behavior of action maps that use content offers."> Cookie usage Configure how Altocloud uses cookies to store customer data."> debug Learn how to send helpful messages to the console when you are troubleshooting tracking behavior."> destroy Learn how to use the `destroy` method to stop all Journey JavaScript SDK activity and remove all tracking information."> Display icons in the Journey gadget Learn how to use the SDK to display icons for tracked user behavior on the Journey map ."> dom - ready Learn how to make a function handler to run when the DOM has fully loaded."> Exclude URL query parameters Exclude irrelevant URL query parameters to improve analytics."> Form Tracking API Learn how to use an API to track form submission and abandonment events, including what visitors enter in your forms."> forms:track Learn how to use the`forms:track` method to capture when users complete web-based forms."> Get started Get started using the Journey JavaScript SDK."> identify Learn how to use the `identify` method to add information to a customer record."> Initialization methods View the methods that initialize and destroy the Journey JavaScript SDK."> initialized Learn how to use the `initialized` method to be notified when the Journey JavaScript SDK has fully initialized."> init Learn how to use the `init` method to initialize the Journey JavaScript SDK."> load Learn how to add the functionality of a module to to the Journey JavaScript SDK."> Method reference View a list of all methods in the Journey JavaScript SDK."> off To unsubscribe from receiving notifications about a particular type of SDK activity, use ac('off')."> once To receive a notification for only the first SDK event for a particular type of activity, use ac('once')."> on To subscribe to receive notifications about a particular type of SDK activity, use ac('on')."> pageview Learn how to use the `pageview` method to track when users view your webpages."> record Learn how to use the `record` method to capture website events."> Retrieve visitor information Use the Visit API and Customer API to retrieve visitor information."> serialize Learn how to use the `serialize` method to submit serialized data from a form."> Session methods View the methods that allow you to obtain data about a specific session."> Track the #hash portion of the URL fragment Track the #hash portion of URL fragments to include relevant information in analytics."> Tracking methods Learn about the methods available in the Journey JavaScript SDK that you can use to extend and enhance how Altocloud tracks customer activity on your website."> Map traits to link customer records Learn how to see more complete customer profiles in Live Now by mapping multiple records for the same customer."> Track a page's canonical URL Track the canonical URL of a page for precise pageview counts."> Examples: Events methods with content offers See examples how how to use Events methods with content offers."> Use Events methods with web actions Use Events methods to subscribe to events that occur during the lifecycle of Altocloud web actions such as web chats and content offers. This raw data can be streamed to third-party analytics platforms or to tag management platform data layers for use in analytics and reporting platforms."> Examples: Events methods with web chats See examples of how to use Events methods with web chats."> Utility methods Learn

how to simplify calls to the Journey JavaScript SDK collection methods."> Web chat lifecycle See the lifecycle of a web chat and the metrics that we capture at each state along the way. Lifecycle states are used in reporting and in determining the triggering behavior of action maps that use web chats."> Web Tracking API Learn how to track user activity via an API.

Use the Journey JavaScript SDK to customize how Altocloud tracks and manages customer activity on your website.

Get started

Get started using the Journey JavaScript SDK.

When you deploy the Altocloud tracking snippet on your website, you initialize the Journey JavaScript SDK. You can enhance and refine the tracking snippet by using the Journey JavaScript SDK:

- Copy and deploy the tracking snippet.
- Track activity in a GDPR-compliant manner.
- Configure advanced tracking.
- Understand how Altocloud uses cookies.
- Use the Web Tracking API.
- Method reference

About the tracking snippet

Contents

Learn what happens after you deploy the tracking snippet on your web pages. Not what you're looking for? Try About tracking with Altocloud instead.

Related pages:

-
-

How the tracking snippet works

Altocloud provides a traditional tracking snippet and an SPA tracking snippet to track activity on your webpages. For information about these different types of snippets, see Types of snippets.

When you add the snippet to a webpage, the tracking snippet loads the Journey JavaScript SDK whenever a visitor accesses a tracked page. To ensure that the process of loading the Journey JavaScript SDK does not cause the visitor to wait for the page to load. Instead, we cache the Journey JavaScript SDK in the visitor's browser and load it asynchronously.

> **Important**
>
> - The Altocloud tracking snippet loads JavaScript asynchronously without slowing down page loading.
> - The Altocloud SDK does not block the loading of any other resources.
> - This snippet represents the minimum configuration needed to add the Altocloud customer support widget. To start sending tracking data from your visitors back to the Altocloud servers, see Web Tracking API.

Once loaded, the Journey JavaScript SDK:

- Creates a new script HTML element
- Sets the source attribute to the Altocloud SDK's URL
- Sets the async attribute to 1 (truthy)
- Adds the script element to the DOM
- Sets the name of the only global function exposed by the Altocloud SDK to `'ac'`
- Calls the `ac` function and executes the following commands:
  - `init` to set the organization ID and region, and to specify which account to send the data to

- `pageview` to record an event when a page with the tracking snippet is loaded, allowing Altocloud to track the visitor's journey across the website.

For more information about the the tracking method, see Web Tracking API.

Cookie usage

Contents

Configure how Altocloud uses cookies to store customer data.

Related pages:

-
-

The cookie that identifies customers

To identify customers, Altocloud uses a first-party cookie named **_actmu** to store the visitor ID. This is a unique, randomly generated string that is stored in the browser. This cookie is sent to the Altocloud APIs to determine whether a **tracked event** is associated with a particular customer and to associate subsequent visits to the same site with the same customer.

Cookies that expire after 1 year

> **Important**
> You can change the expiration time for all cookies that expire after 1 year. For information on how to do this, see Configure advanced tracking.

Name	Description
_actcc	Distinguishes visitor's `beacon` and `pageview` counts for the current session and all sessions collectively.
_actmi	Distinguishes logged in visitors. This cookie is set to the user ID passed when calling the `identify` method.
_actmu	Distinguishes visitors. The cookie is created when the Journey JavaScript SDK library executes and no existing __actmu cookies exists.
_actvc	Distinguishes the visit count for an individual visitor. This cookie is created and updated on each separate visit.
_actts	Distinguishes timestamps of the visitor's first, previous, and current session.

Cookies that expire after 30 minutes

Name	Description
_actmm	Distinguishes UTM information.
_actmr	Distinguishes the session referrer.
_actms	Distinguishes session ID.

Cookies that expire at varying times

Name	Description	Expiration details
_ac_test	Altocloud uses this cookie to check whether first-party cookies are supported by the browser and whether Altocloud can successfully set the tracking cookies.	Immediately after it is set.
_actmf	Stores data submitted in a form and sends it on the next page load.	If this cookie is not set before user leaves the site, the cookie expires when the session expires.
_actmh	Stores a hash of the visitor information that is passed when calling `identify` to minimize the number the times this information is sent to the Altocloud servers.	

Advanced tracking with cookies

Contents

Learn how to use the Journey JavaScript SDK to refine how Altocloud tracks customer data. Alternatively, you can refine tracking using your preferred tag manager. For more information, see About event tracking.

Related pages:

-
-

About advanced tracking

Use `init` options to change how Altocloud sets cookies and how you track your visitors across subdomains or multiple domains.

Cookie options

By default, the Journey JavaScript SDK sets the cookie expiration date and determines the cookie domain. To customize these settings, use the following parameters with the `init` method.

me	Description	Default
`allowedLinkers`	An array of domains that are allowed to link into the current domain for cross-domain tracking	Null
`autoLink`	All links to the specified domains on the site will be augmented to contain information that allows the linked page to continue the current tracking session	Null
`cookieDomain`	Determines the domain on which the cookies are set	The highest level domain possible
`cookieExpires`	Specifies the expiration time in seconds for the `actmi`, `_actmu` and `_actvc cookies`. For example, 1 year = 365 days * 24 hours * 60 minutes * 60 seconds = 31536000 seconds.	1 year
`cookiePrefix`	Adds a prefix to the names of the Altocloud cookies	"_" (underscore)

Example

```
ac('init', 'YOUR-ORGANIZATION-ID', {
  region: 'YOUR-REGION',
  cookieDomain: 'YOUR-DOMAIN',
  cookieExpires: 31536000,
  cookiePrefix: 'YOUR-PREFIX'
});
```

Track a domain and its subdomains

By default, to simplify cross-domain tracking implementations, Altocloud will automatically write cookies to the highest level domain possible. If you manage both a domain and one or more subdomains such as `www.example.com`, `blog.example.com` and `store.example.com`, the cookie domain used to store cookies will be `.example.com`.

Tracking subdomains in 2 different accounts

For tracking subdomains separately in 2 different accounts, the Altocloud tracking snippet needs to be customized to specify the desired domain in the `init` call:

```
ac('init', 'YOUR-ORGANIZATION-ID', {
  region: 'YOUR-REGION',
  cookieDomain: 'subdomain.example.co.uk'
});
```

Track multiple domains

A default setup will track traffic to each domain (for example: `example.com` and `example.co.uk`) independently. Therefore, a visitor arriving in one domain who then proceeds to another domain that is set up with the same tracking account is counted as two separate users with two separate visits/sessions. Each of these comprises the activities (pages visited, and so on) that occurred on each domain.

To enable cross domain tracking, also known as *site linking*, to bundle together the traffic to both domains, modify the `init` call in the Altocloud tracking snippet to allow auto linking to another domains. Suppose you have a site, `example.com` that links to `example.co.uk` and vice versa.

To enable tracking across both of these domains, make the following modifications to the Altocloud tracking snippet in `example.com`:

```
ac('init', 'YOUR-ORGANIZATION-ID', {
  region: 'YOUR-REGION',
  allowedLinkers: ['example.co.uk'],
  autoLink: ['example.co.uk']
});
```

Then, modify the Altocloud tracking snippet in `example.co.uk` to also allow linking. Specifically, to accept the user tracking cookies from another site, and to enable auto linking to decorate all links pointing to example.com:

```
ac('init', 'YOUR-ORGANIZATION-ID', {
  region: 'YOUR-REGION',
  allowedLinkers: ['example.com'],
  autoLink: ['example.com']
});
```

Asymmetric site linking

You can set up asymmetric site linking. In this case, you can carry cookies from `example.com` to `example.co.uk` but not the other way around. In this case, the Altocloud tracking snippet in `example.com` should remove `example.co.uk` from `allowedLinkers`, and `autoLink` should be removed from the Altocloud tracking snippet added to `example.co.uk`.

Method reference

Contents

View a list of all methods in the Journey JavaScript SDK.

Initialization methods

For complete information about a method, including signature and arguments, click the method's name.

Method	Description
init	Initializes the Journey JavaScript SDK
initialized	Notifies when the Journey JavaScript SDK is fully initialized.
destroy	Stops all Journey JavaScript SDK activity and removes all tracking information

Session methods

For complete information about a method, including signature and arguments, click the method's name.

Method	Description
api.session.getData	Returns an object that contains session information
api.session.getCustomerCookieId	Returns a string that contains the customer cookie Id
api.session.getId	Returns a string containing session Id

Tracking methods

For complete information about a method, including signature and arguments, click the method's name.

Method	Description
pageview	Tracks page views
record	Records custom website events
identify	Adds information to a customer
forms:track	

Events methods

For complete information about a method, including signature and arguments, click the method's name.

Method	Description
on	Subscribes to notifications about a particular type of Journey JavaScript SDK activity.
once	Subscribes to notifications for first event for a particular type of Journey JavaScript SDK activity.
off	Unsubscribes from notifications about a particular type of Journey JavaScript SDK activity

Utility methods

For complete information about a method, click the method's name.

Method	Description
serialize	Used for passing serialized data from a form.
dom - ready	Allows you specify a function handler to run when the DOM has fully loaded.
debug	Sends helpful messages to the console.

Initialization methods

Contents

View the methods that initialize and destroy the Journey JavaScript SDK.

Initialization methods

- init
- initialized
- destroy

init

Contents

Learn how to use the `init` method to initialize the Journey JavaScript SDK.

Related pages:

-
-

Description

The `init` method initializes the Journey JavaScriipt SDK.

> **Important**
> For GDPR compliance, be sure to obtain a customer's consent before you call the `init` method. For more information about how to use Altocloud in a GDPR-compliant manner, see GDPR.

Signature

```
ac('init', organisationId, options)
```

Arguments

- organizationId
- options

organizationId

- Description: your organization's unique ID. Find your organization ID

- Type: string
- Status: required

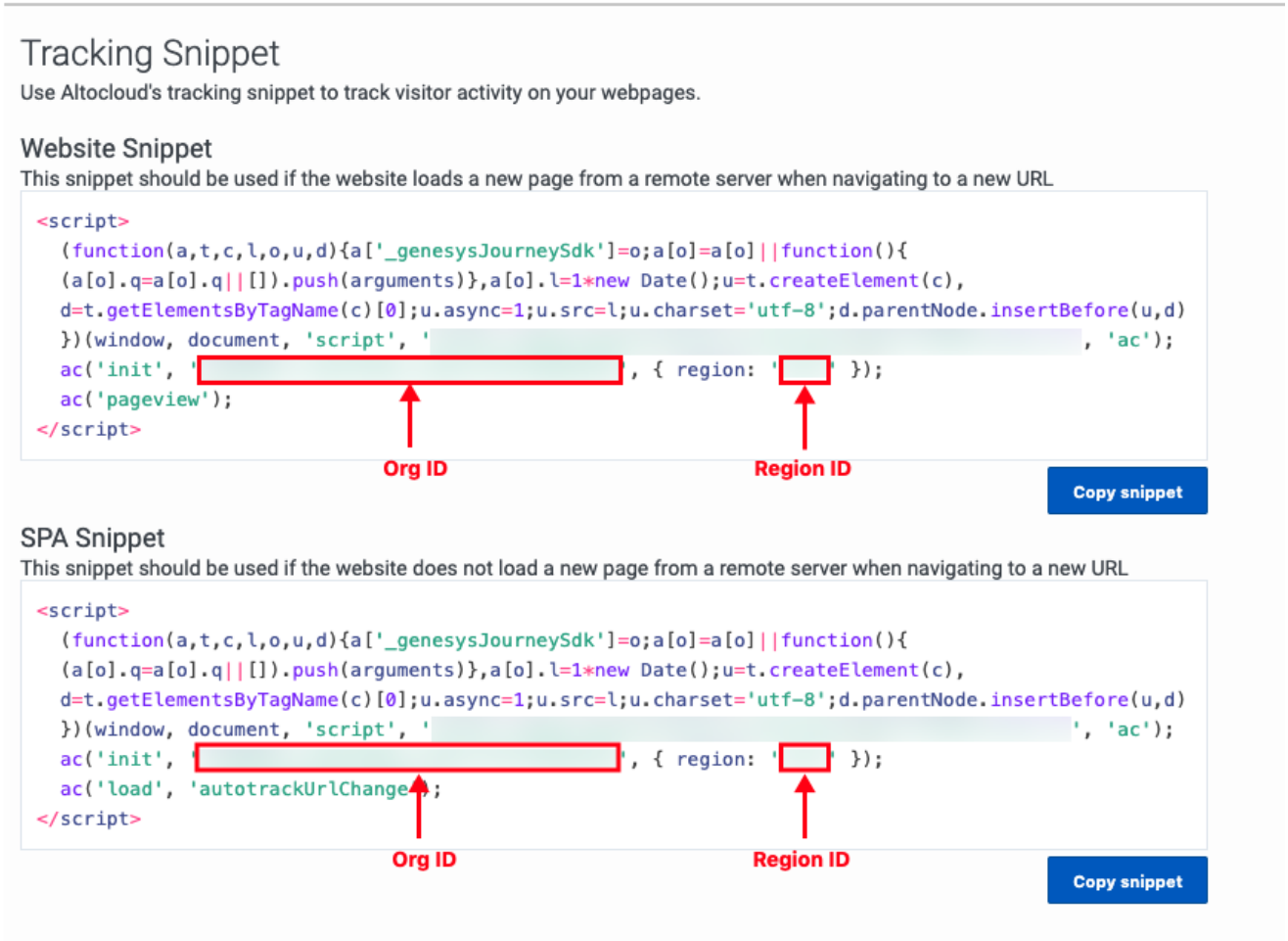
options

> **Tip**
> For detailed explanations of how you can use these options to configure tracking, see Advanced tracking with cookies.

- Description: configures the Journey JavaScript SDK with its region and other known default options.
- Type: object
- Status: required
- Properties:

Name	Description
region	Your organization's region. Find your region ID
cookieDomain	Sets a custom cookie domain
cookieExpires	Sets a time in seconds for a customer's cookie to expire
cookiePrefix	Sets a custom cookie prefix
autoLink	An array of domains whose outbound links will be modified to contain tracking information. For
allowedLinkers	An array of domains whose inbound referral will set tracking information. For an example, see C
canonicalLink	Uses canonical links
globalTraitsMapper	Maps custom attributes to traits For more information, see Traits Mapper.

Find your org ID and region ID



Go to **Genesys Cloud** > **Admin** > **Tracking Snippet**.

Region names and IDs

The following table lists the available region names and corresponding Ids.

Region name	ID
Americas (US West)	usw2
Americas (US East)	use1
Americas (Canada)	cac1
EMEA (Dublin)	euw1
EMEA (London)	euw2

Region name	ID
EMEA (Frankfurt)	euc1
Asia Pacific (Tokyo)	apne1
Asia Pacific (Seoul)	apne2
Asia Pacific (Sydney)	apse2

initialized

Contents

Learn how to use the `initialized` method to be notified when the Journey JavaScript SDK has fully initialized.

Related pages:

-
-

Description

Use the `initialized` method to be notified when the Journey JavaScript SDK has fully initialized. This may be useful in situations where tracking data from the Journey JavaScript SDK is required, but because SDK initialization takes place after the page has loaded, tracking has not yet begun.

For example, some businesses require GDPR consent before they begin tracking user activity. In this case, a business may present a GDPR consent dialog box to a user when the user arrives at the webpage. Until the user has agreed to allow tracking, the Journey JavaScript SDK remains in an uninitialized state, and calls to SDK methods, such as the api.session methods, will fail.

After the user provides their consent, the SDK may be initialized and can begin tracking visitor activity. Once initialized, callbacks registered using the initialized method will be invoked and can begin to use the Journey JavaScript SDK's other methods.

Signature

```
ac('initialized',  eventHandler);
```

Example

```
ac('initialized', () => {
  console.log('Tracking SDK initialized');
  ac('api.session.getData', (session) => {
    console.log('Session data', session);
  });
});
```

destroy

Contents

Learn how to use the `destroy` method to stop all Journey JavaScript SDK activity and remove all tracking information.

Related pages:

-
-

Description

The `destroy` method stops all Altocloud SDK activity and removes all tracking information.

Signature

```
ac('destroy')
```

Arguments

None.

GDPR and destroyed data

The `destroy` method stops all Altocloud tracking immediately. In addition to ceasing tracking, the `destroy` method also removes all cookies set by the SDK; no more proactive engagements will be offered. Use the `destroy` method if a customer requests that you stop tracking their activities on your website. For more information about using Altocloud in a GDPR-compliant manner, see GDPR.

Session methods

Contents

View the methods that allow you to obtain data about a specific session.

Session methods

Altocloud session methods allow you to obtain session-specific data from web sessions.

- api.session.getCustomerCookieId
- api.session.getData
- api.session.getId

api.session.getCustomerCookieId

Contents

Learn how to obtain a customer's cookie Id.

Related pages:

-
-

Description

api.session.getCustomerCookieId returns a string that contains the customer's cookie Id.

Signature

```
ac('api.session.getCustomerCookieId', (err, cookieInfo) => {
  if (err) {
    // handle error
    return;
  }

  return cookieInfo;
});
```

Callback

The callback takes err as the first parameter.

api.session.getData

Contents

Learn how to obtain the data such as the short ID for a particular customer's session.

Related pages:

-
-

Description

api.session.getData returns an object that contains the session Id, short Id, and customer cookie Id.

Signature

```
ac('api.session.getData', (err, sessionInfo) => {
  if (err) {
    // handle error
    return;
  }

  return sessionInfo;
});
```

Example

The following is an example of an object that is returned by api.session.getData:

```
{
  id:
  shortId: 12345
  customerCookieId:
}
```

Callback

The callback takes err as the first parameter.

api.session.getId

Contents

Learn how to get the Id of a particular session.

Related pages:

-
-

Description

api.session.getData returns an object that contains session Id.

Signature

```
ac('api.session.getId', (err, IdInfo) => {
  if (err) {
    // handle error
    return;
  }

  return IdInfo;
});
```

Callback

The callback takes err as the first parameter.

Tracking methods

Contents

Learn about the methods available in the Journey JavaScript SDK that you can use to extend and enhance how Altocloud tracks customer activity on your website.

Related pages:

-
-
-
-
-

Tracking methods

- pageview
- record
- identify
- forms:track

pageview

Contents

Learn how to use the `pageview` method to track when users view your webpages.

Related pages:

-
-
-
-
-

Description

The `Pageview` method tracks page views.

Signature

```
ac('pageview', [pageOverrides], [customAttributes], [options])
```

Arguments

- pageOverrides
- customAttributes
- options

pageOverrides

- Description: sets custom page view location and title
- Type: object
- Status: optional
- Properties:

Name	Description	Type	Status	Default
location	Page URL	string	optional	
title	Page title	string	optional	

customAttributes

- Description: adds extra information to `pageview` event
- Type: object
- Status: optional
- Restrictions: a flat object with properties of type string, number, or boolean

options

- Description: used for additional configuration
- Type: object
- Status: optional
- Properties:

Name	Description	Type	Status	Default
traitsMapper	Used to map custom attributes to traits. For more information, see Traits Mapper.	traitsMapper		
callback	Called once beacon is sent	function	optional	
callbackTimeout	ms to wait for beacon to send	number	optional	

record

Contents

Learn how to use the `record` method to capture website events.

Related pages:

-
-
-
-
-

Description

The `record` method records custom website events.

Signature

```
ac('record', eventName, [customAttributes], [options])
```

Arguments

- eventName
- customAttributes
- options

eventName

- Description: name of the custom event
- Type: string
- Status: required

customAttributes

- Description: adds extra information to `pageview` event
- Type: object
- Status: optional
- Restrictions: a flat object with properties of type string, number, or boolean

Example

```
ac('record', 'product_added', { price: 15.99, code: 'CDE-123', name: 'Product', hasBatteries: false });
```

options

- Description: Used for additional configuration
- Type: object
- Status: optional
- Properties:

Name	Description	Type	Status	Default
traitsMapper	used to map custom attributes to traits. For more information, see Traits Mapper.	traitsMapper		
callback	called once beacon is sent	function	optional	
callbackTimeout	ms to wait for beacon to send	number	optional	

identify

Contents

Learn how to use the `identify` method to add information to a customer record.

Related pages:

-
-
-
-
-

Description

The `identify` method adds information to a customer record.

> **Important**
> `identify` will affect the next web event sent via `pageview` or `record` method.

Signature

```
ac('identify', [loginId], [traits], [callback])
```

Arguments

- loginId
- traits
- callback

login

- Description: an Id that can be used to stitch customer identities together
- Type: string/null
- Status: required
- Note: If loginId is null, the current identity will be cleared.

traits

- Description: information about a customer
- Type: object
- Status: optional
- Restrictions: A flat object with properties of type string, number, or boolean

For more information about how to link different customer records together, see Traits mapper.

callback

- Description: callback
- Type: function
- Status: optional

forms:track

Contents

Learn how to use the `forms:track` method to capture when users complete web-based forms.

Related pages:

-
-
-
-
-

Description

The `forms:track` method tracks form submission and abandonment events. By default, forms tracking captures form data when a user submits or abandons a form.

- Recorded form data **includes** the values of all input, select, and textarea fields.
- Recorded form data **excludes** the values of hidden, submit, and password fields, along with any fields that contain any of the sensitive input strings.

> **Important**
> In order for Altocloud SDK forms tracking to capture form data, each input needs to have a properly defined `name` attribute.

See also Form Tracking API.

Signature

```
ac('forms:track', [selector], [options]);
```

Arguments

- selector
- options

selector

- Description: The CSS selector for the element or elements you want to track.
- Type: string
- Status: optional
- Default: form

options

- Description: The activity or behavior that you want to track.
- Type: object
- Status: optional
- Default: {}
- Properties:

Name	Description	Type	Status	Default	Arguments
captureFormDataOnAbandon		boolean	optional	true	
captureFormDataOnSubmit		boolean	optional	true	
transform		function	optional		formDataObject
traitsMapper		traitsMapper			

Display icons in the Journey gadget

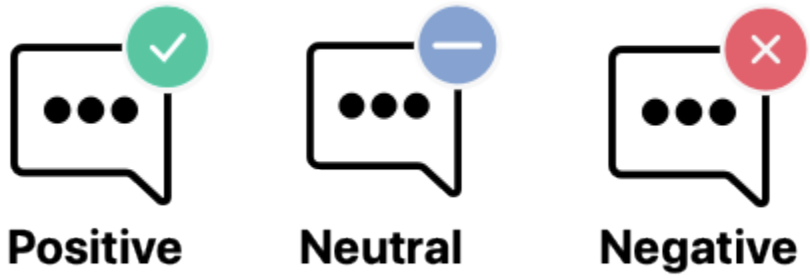
Contents

Learn how to use the SDK to display icons for tracked user behavior on the Journey map .

Related pages:

-
-
-
-
-

About the icons

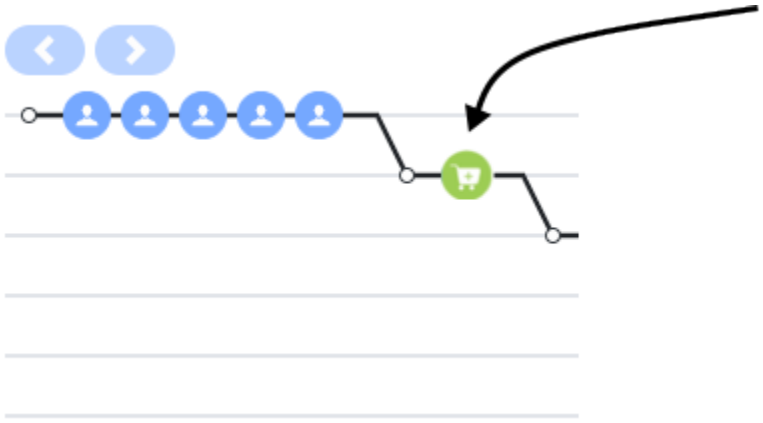


Use the `ac('record')` method to display an Altocloud icon on the Visit journey map (admin view) when a user completes a tracked behavior:

Available icons

- Purchase-related icons
- Form-related icons
- Journey-related icons

Code example



This code example shows how to use `ac('record')` to display the Product added icon in the Visit journey map (admin view) when a user adds a t-shirt to their shopping cart.

```
ac('record', 'product_added', [optionalExtraDataObject]
ac('record', 'product_added', { name: 't-shirt', id: 'hkds9d8j', price: '$45.45' });
```

Purchase-related icons

Icon	Tooltip text	Description	Name
	Product added to cart	The user added a product to their shopping cart.	"product_added"
	Product removed from cart	The user removed a product from their shopping cart.	"product_removed"
	Checkout complete	The user completed the purchase of the items in their shopping cart.	"product_purchased"

Form-related icons

> **Important**
> You can display the icons in this section via `ac('record')` or via auto form tracking.

Icon	Tooltip text	Description	Name
	Form submitted	The user submitted a form.	"form_submitted"
	Form abandoned	The user navigated away from a form before completing it.	"form_abandoned"

Miscellaneous icons

Icon	Tooltip text	Description	Name
	Searched	The user searched for the string shown in the tooltip.	"search_performed"

About Events methods

Events methods allow developers to build functionality that reacts to state changes in the SDK.

Events methods

Events methods allow you to receive notifications about activities that are tracked by the Journey JavaScript SDK.

- Methods:
  - on
  - once
  - off

Use Events methods with content offers

You can use Events methods to capture data related to content offers.

- Use Events methods with web actions
- Content offers lifecycle
- Examples: Events methods with content offers

Use Events methods with web chats

You can use Events methods to capture data related to web chats.

- Use Events methods with web actions

- Content offers lifecycle
- Examples: Events methods with web chats

on

Contents

To subscribe to receive notifications about a particular type of SDK activity, use ac('on').

Related pages:

-
-

Description

Use ac('on') to subscribe to SDK events of a given event type and state.

Signature

```
ac('on', '', function eventHandler(evt) { /* do stuff */ });
```

Examples

For examples, see:

- Examples: Events methods with content offers
- Examples: Events methods with web chats

once

Contents

To receive a notification for only the first SDK event for a particular type of activity, use ac('once').

Related pages:

-
-

Description

Use ac('once') to subscribe to the first SDK event of a given event type and state.

Signature

```
ac('once', '', function eventHandler(evt) { /* do stuff */ });
```

Example

See:

- Examples: Events methods with content offers
- Examples: Events methods with web chats

off

Contents

To unsubscribe from receiving notifications about a particular type of SDK activity, use ac('off').

Related pages:

-
-

Description

Use ac('off') to unsubscribe from receiving future SDK events of a given event type and state.

Signature

```
ac('off', '', function eventHandler(evt) { /* do stuff */ });
```

Example

See:

- Examples: Events methods with content offers
- Examples: Events methods with web chats

Use Events methods with web actions

Contents

Use Events methods to subscribe to events that occur during the lifecycle of Altocloud web actions such as web chats and content offers. This raw data can be streamed to third-party analytics platforms or to tag management platform data layers for use in analytics and reporting platforms.

Related pages:

-
-
-
-

Event methods for web actions

To capture information about events that occur during the lifecycle of a web action, use the Events methods shown in the following table.

Method	Description
on	Subscribe to start receiving events for a given type of web action in a given state.
once	Receive events for only the first occurrence of a given type of web action in a given state.
off	Unsubscribe to stop receiving events for a given type of web action in a given state.

Media types, lifecycle states, and code examples using Events methods

Media type	Lifecycle states	Code examples
contentoffer	Content offers lifecycle	Examples: Events methods with content offers
webchat	Web chat lifecycle	Examples: Events methods with web chats

Event types for web actions

The following table lists the events that you can use with Events methods for web actions. Event information returned includes the action state, customer ID, session ID, and action map ID.

Event	Data type	Description
actionId	UUIDv4	Unique Id for a specific Predictive Engagement action.
actionState	String	Current state of the action. For example, offered.
actionMediaType	String	The engagement type. For example, webchat.
actionMapId	UUIDv4	Id of the action map that qualified/ triggered this action.
actionMapVersion	Integer	Version of the action map.
customerId	UUIDv4	Stable identifier of the customer. For example, a cookie Id.
customerIdType	String	The specific type of customer identifier (always "cookie").
sessionId	UUIDv4	Identifier of the customer's current web session.
errorCode	Integer	Status code for any exceptions caught during presentation of the action.
errorMessage	String	Error message for any exceptions caught during presentation of the action.

Capture more data with Genesys widgets

The Web Action Events API can be used with Genesys Widgets commands to enrich events with more data that may be useful.

For example, the Genesys Cloud `conversationId` may be useful in an analytics context. For more information on the Widgets API, see API Commands.

```
ac('on', 'webchat:all', (evt) => {
  _genesys.widgets.bus.command('WebChatService.getSessionData').then((data) => {
    if (data.conversationId) {
      evt.conversationId = data.conversationId;
    }

    someAnalyticsProvider.send(evt);
  })
});
```

> **Important**
>
> The data returned by WebChatService.getSessionData will differ based on your Genesys platform.

Content offers lifecycle

Contents

See the lifecycle of a content offer and the metrics that we capture at each state along the way. Lifecycle states are used in reporting and in determining the triggering behavior of action maps that use content offers.

Related pages:

-
-
-
-

Content offer lifecycle states

The following table provide details about the lifecycle of a content offer, including the events that can occur and the data that is available for use with the Events methods.

State	Event	Description	Data available
offered	Web Actions Offered	The customer's activity has qualified an action map, and a proactive invitation is offered.	See Event types for web actions.
accepted	Web Actions Accepted	The customer accepts the invitation by clicking a	See Event types for web actions.

State	Event	Description	Data available
		button like **Book now!** This is a terminal state.	
rejected	Web Actions Rejected	The customer rejects the invitation by either clicking **X** or a button like, **No, but thank you**. This is a terminal state.	See Event types for web actions.
errored	Web Actions Errored	An error occurred in the widget that prevented the engagement from occurring. **Note:** This event does not have a corresponding metric in the Action Map Performance Report.	See Event types for web actions. In addition, the errorMessage field is available.
ignored	Web Actions Ignored	The customer has ignored the invitation by navigating away from or around it. This is a terminal state. **Note:** This event does not have a corresponding metric in the Action Map Performance Report.	See Event types for web actions.

Terminal states for content offers

In the content offer lifecycle, certain states are considered *terminal*, or final, states. If a customer navigates to a web page where an action map is set to trigger a content offer, the action map will not offer the content offer if it is already in a terminal state. This ensures that a customer does not receive the same content offer after the customer has already accepted the offer or has indicated that they are not interested in that particular content offer.

Terminal states for content offers are:

- Accepted
- Rejected
- Ignored

For more information, see Trigger an action map.

Report metrics and events

The metrics used in the Action Map Performance report correlate directly with the event types for web actions. For more information about metrics for content offers, see Monitor a content offer's performance.

Examples: Events methods with content offers

Contents

See examples how how to use Events methods with content offers.

Related pages:

-
-
-
-

Subscribe to offered events

```
ac('on', 'contentoffer:offered', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to accepted events

```
ac('on', 'contentoffer:accepted', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to rejected events

```
ac('on', 'contentoffer:rejected', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to ignored events

```
ac('on', 'contentoffer:ignored', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to errored events

```
ac('on', 'contentoffer:errored', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to all content offer events

```
ac('on', 'contentoffer:all', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Unsubscribe from content offers in the offered state

```
ac('off', 'contentoffer:offered', eventHandler); // unsubscribes `eventHandler` from 'contentoffer:offered' ever
```

Web chat lifecycle

Contents

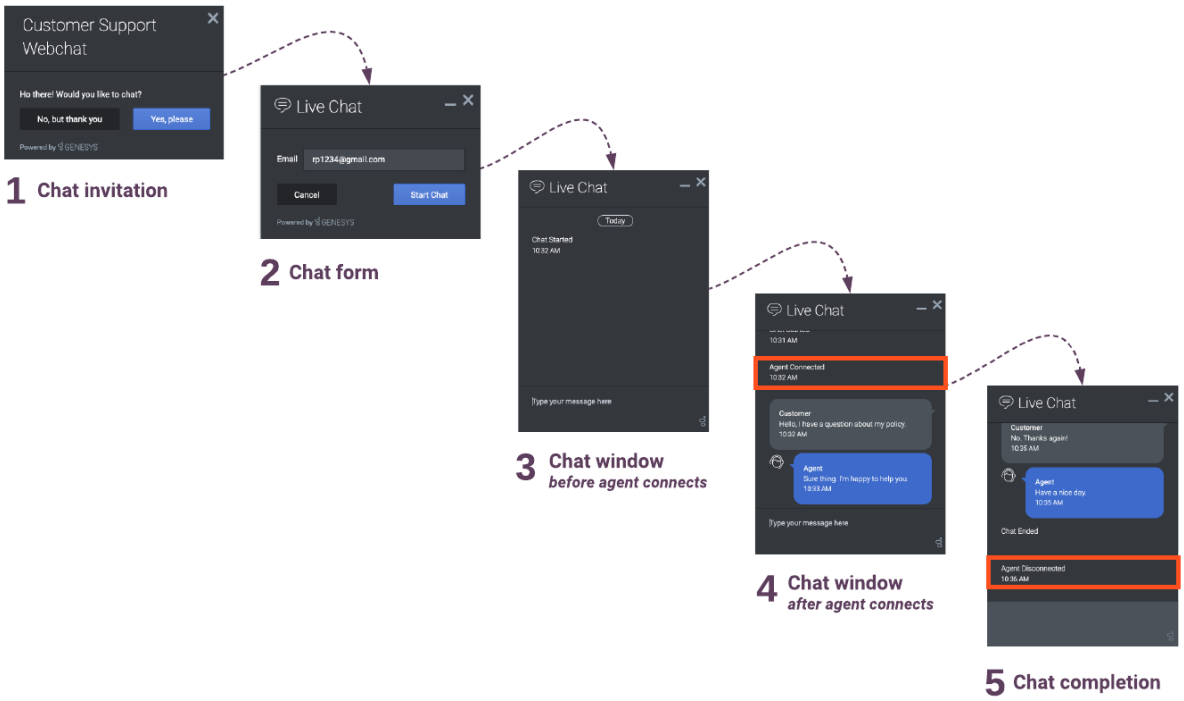
See the lifecycle of a web chat and the metrics that we capture at each state along the way. Lifecycle states are used in reporting and in determining the triggering behavior of action maps that use web chats.

Related pages:

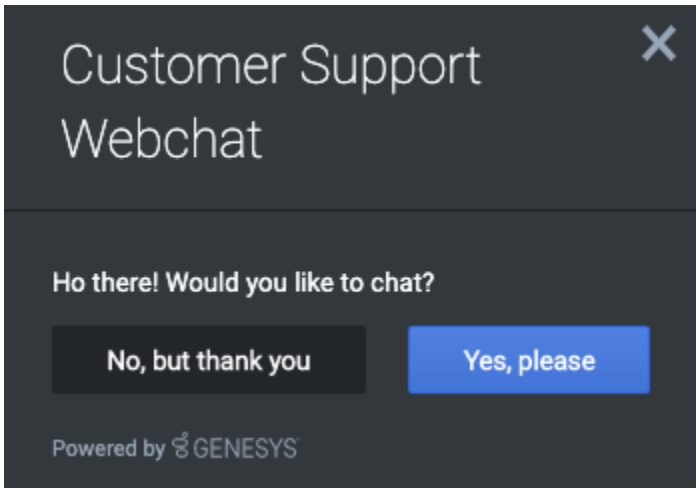
-
-
-
-

Web chat lifecycle

The following diagram shows the stages that occur during the lifecycle of web chats. Subsequent sections provide details about specific states, including the events that can occur and the data that is available for use with the Events methods for web actions. The section, Terminal states, explains how states ensure that customers do not repeatedly see the same offer to chat.



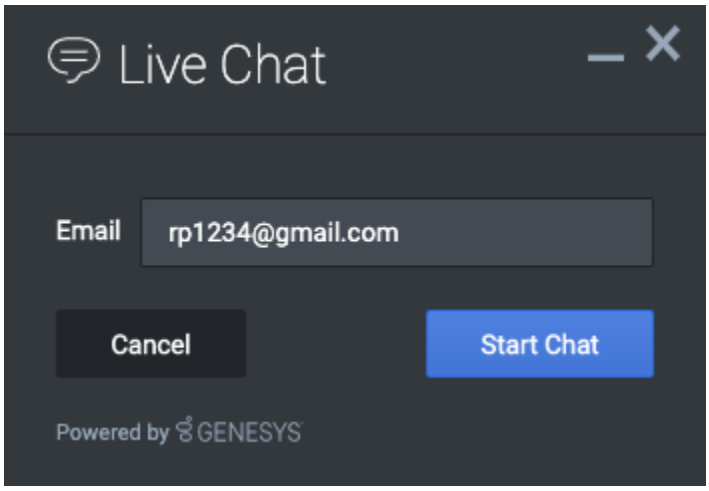
1. Web chat invitation



State	Event	Description	Data collected
offered	Web Actions Offered	The customer's activity has qualified an action map, and a proactive invitation is offered.	See Event types for web actions.
accepted	Web Actions Accepted	The customer accepts the invitation by clicking a button like, **Yes, please**.	See Event types for web actions.
rejected	Web Actions Rejected	The customer rejects the invitation by either clicking **X** or a button like, **No, but thank you**. This is a terminal state.	See Event types for web actions.
errored	Web Actions Errored	An error occurred in the widget that prevented the engagement from occurring. **Note:** This event does not have a corresponding metric in the Action Map Performance Report.	See Event types for web actions. In addition, the errorMessage field is available.
ignored	Web Actions Ignored	The customer has ignored the invitation by navigating away from or around it. This is a terminal state. **Note:** This event does not have a corresponding metric in the Action Map Performance Report.	See Event types for web actions.
timed out	Web Actions Timed Out	The timeout period has been reached and the invitation has been	See Event types for web actions.

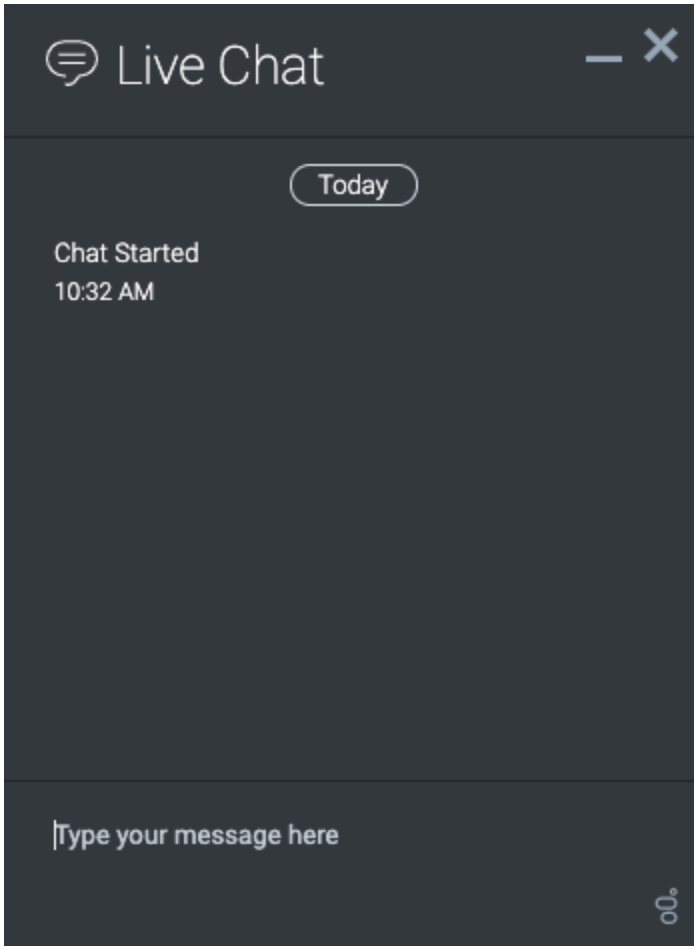
State	Event	Description	Data collected
		rescinded. This is a terminal state. **Note:** This event does not have a corresponding metric in the Action Map Performance Report. The timeout period is configurable via the widget.	

2. Web chat form



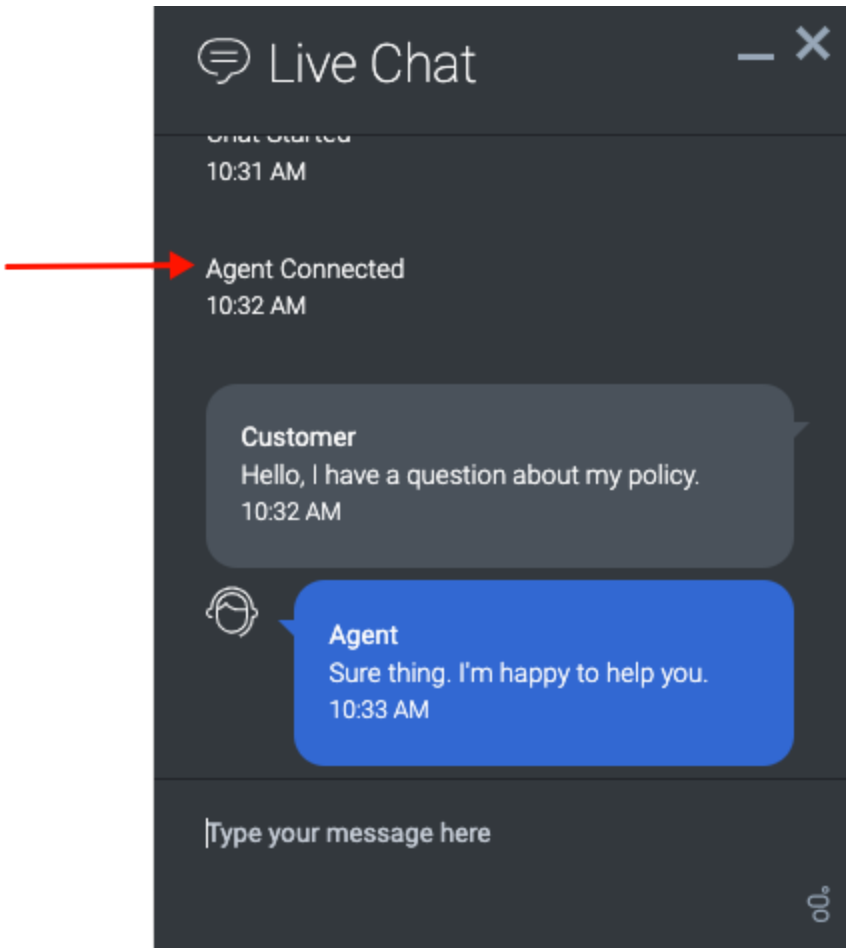
State	Event	Description	Data collected
rejected	Web Actions Rejected	The customer cancels the form by either clicking **X** or a button like **Cancel**. This is a terminal state.	See Event types for web actions.

3. Web chat window: before agent connects



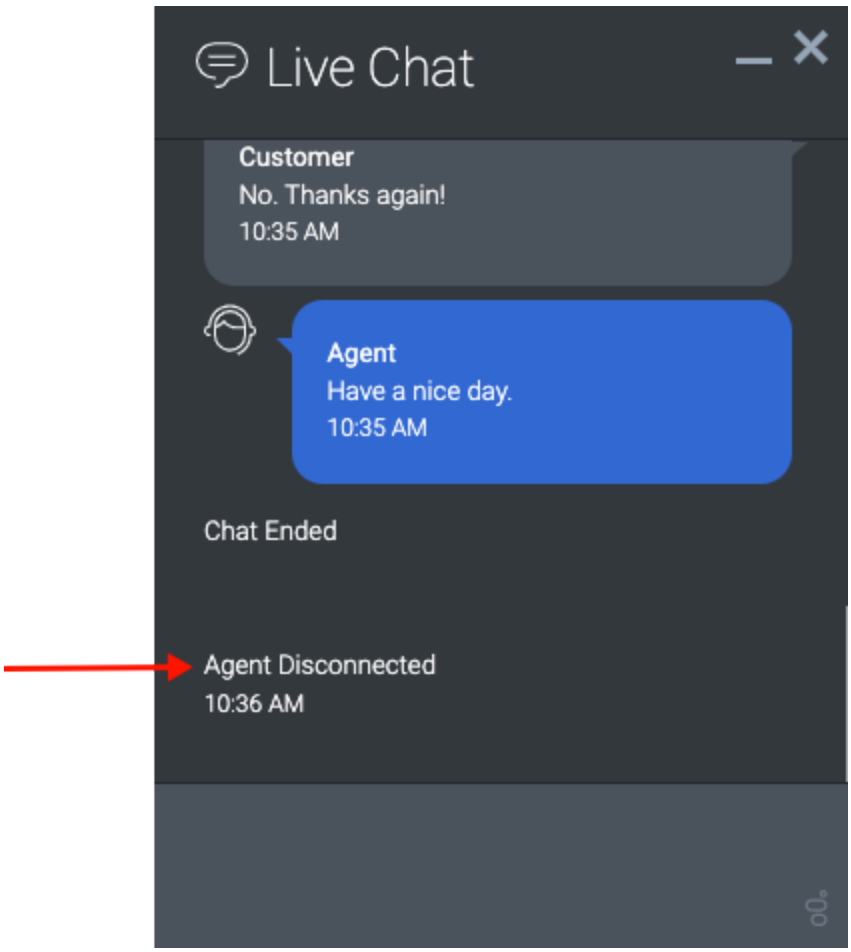
State	Event	Description	Data collected
started	Web Actions Started	After the customer submits the form, a chat interaction is initated.	See Event types for web actions.
abandoned	Web Actions Abandoned	The customer closes the chat window before an agent connects. This is a terminal state.	See Event types for web actions.

4 Web chat window: after agent connects



State	Event	Description	Data collected
engaged	Web Actions Engaged	An agent accepts the chat and is connected with the customer. This is a terminal state.	See Event types for web actions.

5. Web chat completion



State	Event	Description	Data collected
Not applicable/Not tracked	Not applicable/Not tracked	Either the customer or the agent ends the chat. **Note:** This event does not have a corresponding metric in the Action Map Performance Report.	See Event types for web actions.

Terminal states for web chats

In the web chat lifecycle, certain states are considered *terminal*, or final, states. If a customer navigates to a web page where an action map is set to trigger a web chat, the action map will not offer the web chat if it is already in a

terminal state. This ensures that a customer does not receive the same offer to chat after the customer has already accepted the offer or has indicated that they are not interested in that particular chat offer.

Terminal states for web chats are:

- Engaged
- Rejected
- Timed out
- Ignored
- Abandoned

For more information, see Trigger an action map.

Report metrics and events

The metrics used in the Action Map Performance report metrics correlate directly with the event types for web actions. For more information about metrics for web chats, see Monitor a web chat's performance.

Examples: Events methods with web chats

Contents

See examples of how to use Events methods with web chats.

Related pages:

-
-
-
-

Subscribe to offered events

```
ac('on', 'webchat:offered', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to accepted events

```
ac('on', 'webchat:accepted', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to started events

```
ac('on', 'webchat:started', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to engaged events

```
ac('on', 'webchat:engaged', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to rejected events

```
ac('on', 'webchat:rejected', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to ignored events

```
ac('on', 'webchat:ignored', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to errored events

```
ac('on', 'webchat:errored', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to timedout events

```
ac('on', 'webchat:timedout', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Subscribe to all web chat events

```
ac('on', 'webchat:all', (event) => {
  console.log('received %s:%s event', event.actionMediaType, event.actionState, event);
});
```

Unsubscribe from web chats in the offered state

```
ac('off', 'webchat:offered', eventHandler); // unsubscribes `eventHandler` from 'webchat:offered' events
```

Utility methods

Contents

Learn how to simplify calls to the Journey JavaScript SDK collection methods.

Related pages:

-
-
-

Utility methods

Altocloud utility methods simplify calls to the Journey SDK collection methods.

For more information, see Collect tracking data in Web Tracking API.

- serialize
- dom - ready
- debug

serialize

Contents

Learn how to use the `serialize` method to submit serialized data from a form.

Related pages:

-
-
-

serialize

The `serialize` method creates a JSON object where the keys are the form elements names, and the value for each key is the value(s) of the corresponding form elements. This method is useful in scenarios such as submitting a form with AJAX, since submit won't be able to capture the form submission, but instead a record can be invoked passing the serialized form data as its last argument. The serialize method takes the following parameters:

- A unique identifier of the HTML form element
- An (optional) array with the names of the elements to be serialized
- A callback function with the serialized form data into a JSON object

```
ac('serialize', 'my-form', function(data) {
  console.log(data['contact-email']);
});
```

dom - ready

Contents

Learn how to make a function handler to run when the DOM has fully loaded.

Related pages:

-
-
-

dom - ready

The `ready` method of the `dom` module allows you to specify a function handler to run when the DOM has fully loaded. A webpage cannot be altered safely until the document is ready, Journey SDK starts executing regardless of the DOM readiness, and may even fire events before that. Any code reacting to these events that need to modify the DOM should be placed inside the ready handler.

```
ac('dom', 'ready', function () {
  // safely modify the DOM here
});
```

debug

Contents

Learn how to send helpful messages to the console when you are troubleshooting tracking behavior.

Related pages:

-
-
-

debug

To turn on debug mode, invoke the `debug` method. This mode logs helpful messages to the console.

This example shows how to enable debug mode

```
ac('debug');
```

This example shows how to disable debug mode.

```
ac('debug', false);
```

Web Tracking API

Contents

Learn how to track user activity via an API.

About the Web Tracking API

The Web Tracking API lets you track what visitors do on your website. Tracking data is collected through a series of interactions occurring on your website such as pageviews, button clicks, and custom events. These interactions are grouped into visits and act as a container for the actions taken on your website by a specific visitor.

Visits do not have a predefined duration. Depending on your visitor, the visit may be a few seconds or a couple of hours long. A new visit will be created when the visitor has been idle for 30 minutes or more, but they all will be linked to the same visitor.

Obtain consent before tracking visitors

> **Important**
>
> To be compliant with GDPR requirements, consider if you need to obtain a visitor's consent before tracking their data. For more information on using Altocloud in a GDPR-compliant manner, see GDPR.

To implement tracking after receiving consent, modify the tracking snippet so that the `` `ac('init')` `` and `` `ac('pageview')` `` are only called if consent is given, as shown in the following example:

```
 (function(a,t,c,l,o,u,d){a['_genesysJourneySdk']=o;a[o]=a[o]||function(){
  (a[o].q=a[o].q||[]).push(arguments)},a[o].l=1*new Date();u=t.createElement(c),
  d=t.getElementsByTagName(c)[0];u.async=1;u.src=l;u.charset='utf-8';d.parentNode.insertBefore(u,d)
  })(window, document, 'script', 'https://apps.inindca.com/journey/sdk/js/web/v1/ac.js', 'ac');
if (consentGiven) {
// Call the ac('init') function to enable tracking
  ac('init', 'a061a3fe-7a80-4b50-9d3b-df88c0f9efad', { region: 'use1' });
  ac('pageview');
}
```

You are responsible for setting the value for the `` `consentGiven` `` variable based on the visitor's choice.

Enable web tracking

To enable Web tracking on your website, initialize the Tracking SDK and then call the `pageview` method when a visitor navigates to a new page.

Stop tracking if a visitor revokes consent

If a visitor revokes consent at any point, invoke the `destroy` command to stop tracking and remove all cookies, as shown in the following example.

```
// to disable tracking and delete Altocloud cookies
ac('destroy');
```

Types of tracked data

The Journey JavaScript SDK lets you customize how you collect tracking data for your website. The most basic form of tracking is page view tracking. For page view tracking, Altocloud records each page a visitor visits. You can also use the Journey JavaScript SDK to record custom visitor activities such as button clicks. For more information see *Track custom events*, below.

Track pages viewed by your customers

The `pageview` method tracks the pages viewed by your visitors. To send a `pageview`, call the `ac` function and pass `pageview` as the first argument.

```
ac('pageview');
```

> **Important**
>
> By default, the Altocloud tracking snippet contains the `ac` function.

Track custom events

The `record` method allows you to track custom events, usually as a result of a person interacting with an element or control in your website, for example, the click of a button. The `record` method takes two parameters:

- The name of the event to record as a string. **Note:** See *Guidelines for custom event names* below.

- An (optional) key-value hash of properties for the event.
- An (optional) callback function that will be invoked when the request is completed.
- An (optional) callback timeout, in milliseconds, to configure how long to wait if the event is taking too long to complete. This is useful to capture events (such as file downloads) before navigating to a different page/URL, making sure the visitor is always redirected.

```
ac('record', 'section_opened');
```

You may also provide extra metadata with your custom event:

```
ac('record', 'button_clicked', {
  companyName: 'Acme Inc,',
  employees: '100-500'
});
```

The following code sample shows how to record a file downloaded and navigate to the file `URL /files/pricing.pdf` when the event is captured, or after 400 milliseconds:

```
ac('record', 'file.downloaded', {
  name: 'Pricing',
  fileType: 'pdf'
}, function () {
  navigateTo('/files/pricing.pdf');
}, 400);
```

Guidelines for custom event names

> **Important**
>
> Event names must be a maximum of 255 characters. You can use any combination of lowercase alphanumeric characters and an underscore. Keep event names short and descriptive.

The suggested format for custom event names is: *`object-delimiter-action`*, where object is the object that was interacted with, and action is the type of interaction that occurred.

Examples of custom event names include:

- `"detected_errors"`
- `"section_failed"`
- `"section_submitted"`
- `"product_added"`

- `"product_removed"`
- `"address_selected"`

Form Tracking API

Contents

Learn how to use an API to track form submission and abandonment events, including what visitors enter in your forms.

About the Form Tracking API

In order to track customer activities on your website, you must first deploy the tracking snippet. After you do that, you can track form-level activity with the Form tracking API.

In order to be tracked, a form must have an ID, name, or action field that uniquely identifies it. By default, the values of the input, select, and textarea elements entered in the form (fields with personal information such as name, email, phone number and company name) are sent with the form submission and abandoned events. The customer profile is updated accordingly upon form submission.

> **Important**
> You can use Altocloud to track visitor activity in a GDPR compliant-manner, however, you must modify the tracking snippet in order to be compliant. For more information on how to be compliant with the GDPR requirements, see GDPR.

Enable form tracking

To enable form tracking, call `forms:track` and specify the CSS selector of the form(s) that you want to track. For example:

```
ac('forms:track', '.hs-form');
```

To enable tracking for all forms call the function without specifying the selector. For example:

```
ac('forms:track');
```

To enable tracking for multiple forms based on the CSS selector specify a group of selectors as a comma-separated list. For example:

```
ac('forms:track', '#firstFormId, #secondFormId');
```

> **Important**
> Where the same selector is applied to multiple forms, each form with that selector will be tracked separately.

```
ac('forms:track', '#firstFormId, #secondFormId');
```

Example: Track a webpage with 3 forms

Sample webpage with multiple forms

Tracking Forms Example

Get Name Form

First name:

Last name:

Submit

Get Account Information Form

Account No.:

Account branch:

Submit

Register for Newsletter Form

First name:

Last name:

Email address:

Submit

Track all forms in the HTML page

The following code tracks all of the form tags in the HTML page:

```
ac('forms:track');
```

or

```
ac('forms:track', 'form');
```

Track by element IDs

The following code tracks only the Get Account Information form:

```
ac('forms:track', '#get-account-information');
```

The following code tracks both the Get Account Information Form and Get Name Form:

```
ac('forms:track', '#get-account-information, #get-name');
```

Track by element classes

The following code tracks all forms with the `target-class` class. Both the Get Account Information form and Register for Newsletter form are tracked:

```
ac('forms:track', '.target-class');
```

Track by a combination of element IDs and element classes

The following code tracks a form with the `get-name` ID and all forms with the `target-class` class. As a result, all forms shown in the sample are tracked.

```
ac('forms:track', '#get-name, .target-class');
```

Manage form field data

To manage how form field data is sent with form submission and abandonment events, use the `captureFormDataOnAbandon` and `captureFormDataOnSubmit` options with the forms:track method. Set these options to `true` or `false`, depending on the result you want to obtain.

For example, to record a form abandonment event, but exclude any form field data, set `captureFormDataOnAbandon` to `false`.

```
ac('forms:track', '.hs-form', {
  captureFormDataOnAbandon: false
});
```

> **Important**
>
> If you do not set the `captureFormDataOnAbandon` option, or you set it to anything other than `false`, the option assumes the value is `true`, and the recorded form abandonment event will contain serialized form data. This data is subject to custom transformation; sensitive fields are automatically excluded. For more information, see Transform data before sending it and Sensitive form fields that are never tracked.

To capture serialized form data for form submission events, set the `captureFormDataOnSubmit` option to `true`.

You can set both options at the same time, as shown in the following example.

```
ac('forms:track', '.hs-form', {
  captureFormDataOnAbandon: false,
  captureFormDataOnSubmit: false
});
```

Transform data before sending it

To configure the format of the data that is sent with the form submission and abandonment events, specify a custom `transform` function in the `forms:track` options. This function receives an object with the form data; it should return an object with the data to be sent with the form events. You can exclude certain fields, rename fields that are not meaningful, or make any other transformations to the form data before it is sent. The data object is a JSON object of "name-value" pairs, where:

- "name" is the value of the element's id or name attribute
- "value" is the visitor's input

```
ac('forms:track', 'form', {
  transform: function (data) {
    return {
      deliveryOption: data['radioButton_3'],
      acceptedLicense: data['checkbox_license']
    }
  }
});
```

Sensitive form fields that are never tracked

> **Important**
>
> Passwords, hidden fields, and sensitive fields are never tracked.

To denote sensitive fields, use the following regular expression, after removing non-alphanumeric characters from the field name.

```
/pass|billing|creditcard|cardnum|^cc|ccnum|exp|seccode|securitycode|securitynum|cvc|cvv|ssn|socialsec|socsec|csc
```

If we track a field that you consider to be sensitive, please contact customercare@genesys.com.

Map traits to link customer records

Contents

Learn how to see more complete customer profiles in Live Now by mapping multiple records for the same customer.

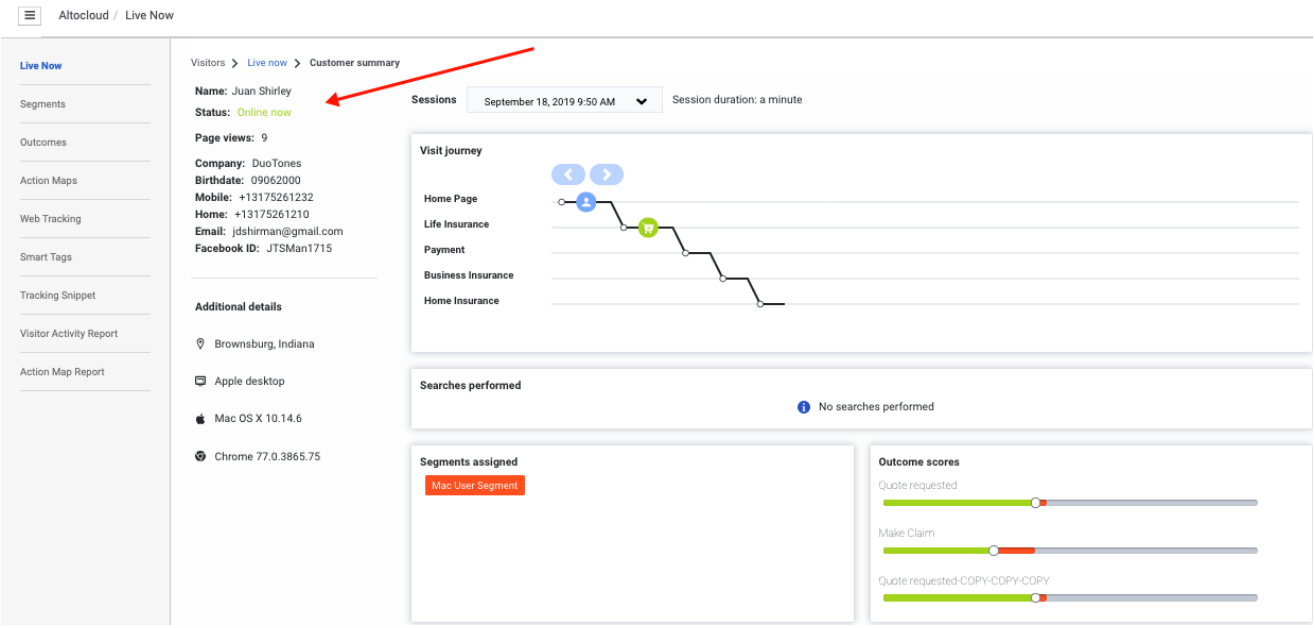
About traits mapping

Traits are properties, such as "email" and "gender" for a customer. Altocloud gathers customer traits every time a customer visits a website that you track with the Altocloud tracking snippet. In some cases, you may have multiple customer records for the same person. For example, if a customer visits your website multiple times and uses a different browser each time. Because Altocloud creates a separate record for each instance, the separate customer records may contain only a subset of all of the traits information that is actually available for the customer. You can link these separate customer records by mapping the traits information they contain. After you do this, you'll be able to see the complete customer information in Live Now.

> **Important**
>
> When customer records are linked, separate customer records are still preserved. They are not consolidated into a single customer record. Instead, all linked customer reords are updated with the current traits information.

After customer records are linked, the traits mapper updates all of the records when new trait information becomes available. Existing or duplicate traits are overwritten with the most current trait information.

View mapped traits in the user interface



After you map traits, they appear here:

- Genesys Cloud > Admin menu > Live Now > Customer summary (admin view)
- Agent user interface > Journey gadget > Customer summary for agents

Map traits globally

To start mapping traits, define a global traits mapper when you deploy the Altocloud tracking snippet on your website. Specifically, when you call init to initialize the Journey JavaScript SDK, identify which attributes you want to treat as traits. See the following code example. For more information, see Methods that track events and Mappable traits.

Whenever Altocloud gathers values for these attributes, they are automatically mapped as traits.

You can also map traits based on specific events.

Example

Map traits for a specific event

In addition to mapping traits globally, you can map specific traits locally for specific events. For more information, see Methods that track events and Mappable traits.

The complete set of map traits for a customer is the union of globally mapped traits and locally mapped traits. For example, suppose you map the email address field via the global traits mapper, but on one page, you ask for the customer's Facebook ID. In this case, both the email address and the Facebook ID are mapped to the customer and both appear in the customer's Live Now profile.

If the same data is captured in two places, the most recent trait mapped appears in Live Now. Previous values for mapped traits are not preserved.

Examples of mapped traits

The following examples show how attributes that are mapped traits. Specifically:

- The attributes, "email" and "emailAdddress" are mapped to the trait "email."
- The attributes, "gender" and "sex" are mapped to the trait "gender."

Attributes	traitsMapper	Traits

{
"email": "firstname.lastname@somemail.com",
"gender": "male",
"comment": "This is great",
"section": "support"
}
	[
	{"fieldName": "email"},
	{"fieldname”: "gender"}
	]
		{
		"email": "firstname.lastname@somemail.com”,
		"gender": "male"
		}

{
"emailAddress": "firstname.lastname@somemail.com",
"sex": "male",
"comment": "This is great",
"section": "support"
}
	[
	{
	"fieldName": "emailAddress",
	"traitName": "email"
	}, {
	"fieldname”: "sex",
	"traitName": "gender"
	}
	]
		{
		"email": "firstname.lastname@somemail.com”,
		"gender": "male"
		}

Methods that track events

Traits mapping can occur whenever there is a tracked event on your website. Specifically, events are tracked when you use the following methods:

- init
- identify
- pageview
- record
- forms:track

Mappable traits

Demographic traits

Trait	Example
gender	male
birthDate	01012001

Company traits

Trait	Example
companyName	Genesys
dunsNumber	622286318
industry	Technology
numberofEmployees	5025

ID traits

Trait	Example
email	JTS1715@gmail.com
facebookId	JTSmith1715
messengerId	John.Smith .1715
twitterId	@JTS1715
telegramId	123456789

Phone traits

Trait	Example
homePhone	3179871234
cellPhone	3179871235
otherPhone	3179712356
workPhone	8179874321

Name traits

Trait	Example
salutation	Mr.
jobTitle	Manager
givenName	John

middleName	Thomas
familyName	Smith
displayName	John

About modules

Contents

Learn about optional JavaScript files that enhance the functionality provided by the Journey JavaScript SDK.

Related pages:

-
-
-
-
-
-
-
-
-
-

About modules

The Journey JavaScript SDK has optional functionality provided as a set of modules. Use the `load` function to load whatever modules you need.

Available modules

- autotrackClick
- autotrackIdle
- autotrackInViewport
- autotrackScrollDepth
- autotrackURLChange

load

Contents

Learn how to add the functionality of a module to to the Journey JavaScript SDK.

Related pages:

-
-
-
-
-
-
-
-
-
-

Description

Use the `load` function to load modules. When you load a module, its functionality is immediately added to the Journey JavaScript SDK.

Signature

```
ac('load', moduleName, ,[userOptions], [callback])
```

Arguments

moduleName

- Description: name of the module to load. The module must be on the list of valid modules.
- Type: string
- Status: required

userOptions

- Description: configuration for the loaded module
- Type: object
- Status: module dependent
- Properties: module dependent

callback

- Description: callback that triggers when the module has finished loading
- Type: function
- Status: optional (default implementation does nothing)
- Arguments
  - On failure, `load` passes an error (like `InvalidModuleError`)
  - In other cases, `load` passes module-dependent values to the callback

Example

autotrackClick

Contents

Learn how to configure which click events Altocloud tracks on your websites, so you have more accurate page tracking information for use in segments and outcomes.

Related pages:

-
-
-
-
-
-
-
-

Description

The `autotrackClick` module tracks when and where a user clicks on a webpage. To use `autotrackClick`, use the following options to configure the click events that you want to track:

- Config
- Callback

Signature

```
ac('load', 'autotrackClick', config, [callback]);
```

Example

```
ac('load', 'autotrackClick', {
  clickEvents: [
    { selector: 'button.bg-green', eventName: 'green_button_clicked' },
    { selector: 'footer *', eventName: 'footer_clicked' }
  ]
}, function () {
    console.log('"autotrackClick" has been loaded');
});
```

Config (required)

Description: identifies an array of click events that should be tracked.

Type: object

Properties:

Name	Description	Type	Status
selector	A string that selects elements. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/Element/matches	String	required
eventName	is a string that will be used as the event name when an element matching the selector is clicked.	String	required

Example

```
{
  clickEvents: [
    { selector: 'button.bg-green', eventName: 'green_button_clicked' },
    { selector: 'footer *', eventName: 'footer_clicked' }
  ]
}
```

Callback (optional)

When a module fully loads, callback is executed. No arguments are passed to the callback.

autotrackIdle

Contents

Learn how to configure when Altocloud detects inactivity on a webpage, so you have more accurate page tracking information for use in segments and outcomes.

Related pages:

-
-
-
-
-
-
-
-
-

Description

The `autotrackIdle` module tracks when and where a user becomes inactive on a webpage. To use `autotrackIdle`, use the following options to configure the idle events that you want to track:

- Config
- Callback

Signature

```
ac('load', 'autotrackIdle', [config], [callback]);
```

Example

```
ac('load', 'autotrackIdle'); // This is for all one idle event config with defaults

ac('load', 'autotrackIdle', {
  idleEvents: [
    {}, // This is for all defaults
    { eventName: 'stuck_on_page' },
    { idleAfter: 60 },
    { eventName: 'idle_for_2_min', idleAfter: 120 },
  ]
}, function () {
    console.log('"autotrackIdle" has been loaded');
});
```

Config (optional)

Description: identifies an array of idle events that should be tracked.

Type: object

Properties:

Name	Description	Type	Status
idleAfter	The number of seconds of inactivity after which an event will fire. The default is 60 seconds. **Note:** The minimum is 30 seconds. If you specify less than 30, then 30 seconds is used.	Number	optional
eventName	A string that will be used as the event name when an element matching the selector is clicked.	String	optional

Example

```
{
  idleEvents: [
    {}, // This is for all defaults
    { eventName: 'stuck_on_page' },
    { idleAfter: 60 },
    { eventName: 'idle_for_2_min', idleAfter: 120 },
  ]
}
```

Callback (optional)

When a module fully loads, callback is executed. No arguments are passed to the callback.

autotrackInViewport

Contents

Learn how to configure which element Altocloud tracks on your websites as they appear and disappear from the viewport, so you have more accurate page tracking information for use in segments and outcomes.

Related pages:

Description

The `autotrackInViewport` module tracks when an element becomes visible in the viewport or disappears from the viewport. To use `autotrackInViewport`, use the following options to configure the elements that you want to track:

- Config
- Callback

Signature

```
ac('load', 'autotrackInViewport', config, [callback]);
```

Example

```
ac('load', 'autotrackInViewport', {
  inViewportEvents: [
    { selector: 'button.bg-green', eventName: 'green_button_element' },
    { selector: 'footer *', eventName: 'footer_element' }
  ]
}, function () {
    console.log('"autotrackInViewport" has been loaded');
});
```

Config (required)

Description: identifies an array of elements that should be tracked.

Type: object

Properties:

Name	Description	Type	Status
selector	A string that selects elements. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/Element/matches	String	required
eventName	is a string that will be used as the event name when an element matching the selector moves into or outside the viewport.	String	required

Example

```
{
  inViewportEvents: [
    { selector: 'button.bg-green', eventName: 'green_button_element' },
    { selector: 'footer', eventName: 'footer_element' }
  ]
}
```

Callback

When a module fully loads, callback is executed. No arguments are passed to the callback.

autotrackScrollDepth

Contents

Learn how to configure which scroll milestones Altocloud tracks on your websites, so you have more accurate page tracking information for use in segments and outcomes.

Related pages:

Description

The `autotrackScrollDepth` module tracks when the user scrolls to see a specific percentage of a webpage. To use `autotrackScrollDepth`, use the following options to configure the click events that you want to track:

- Config
- Callback

Signature

```
ac('load', 'autotrackScrollDepth', config, [callback]);
```

Example

```
ac('load', 'autotrackScrollDepth', {
  scrollDepthEvents: [
    { percentage: 75, eventName: 'scroll_depth_75' },
    { percentage: 100, eventName: 'scroll_depth_100' }
  ]
}, function () {
    console.log('"autotrackScrollDepth" has been loaded');
});
```

Config (required)

Description: identifies an array of scroll depths that should be tracked.

Type: object

Properties:

Name	Description	Type	Status	Default
percentage	A string that selects elements. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/Element/matches	Number (0 - 100)	required	NA
eventName	is a string that will be used as the event name when an element matching the selector is clicked.	String	optional	scroll_depth_ eg if "percentage" = 75, eventName defaults to "scroll_depth_75"

Example

```
{
  scrollDepthEvents: [
    { percentage: 50 },
    { percentage: 100, eventName: 'viewed_full_page' },
  ]
}
```

Callback (optional)

When a module fully loads, callback is executed. No arguments are passed to the callback.

autotrackURLChange

Contents

Learn how to customize tracking for Single Page Application (SPA) websites. Alternatively, use your preferred tag manager to customize and deploy the SPA tracking snippet. For more information, see About event tracking with tag managers. If you are just getting started, read Tracking snippet.

Related pages:

Description

The `autotrackURLChange` module tracks activity on an SPA webpage when a user clicks through relative links or when software-driven activity changes the URL or browser history.

> **Important**
>
> The `autotrackURLChange` module is automatically loaded when you load the SPA snippet.

To customize how `autotrackURLChange` tracks user activity, use these options:

- isUrlChange
- onUrlChange
- callback
- delay

isUrlChange

Description: checks whether the URL has changed since the last check.

Type: function

Status: default implementation available; can be overwritten

Returns: Boolean

Arguments:

Name	Description	Type	Status
oldUrl	the previous URL before the change	string	required
newUrl	the possibly changed URL	string	required

Example

This example shows how to exclude tracking when a user clicks relative links on a webpage.

```
ac('load', 'autotrackUrlChange', {
  isUrlChange: function(oldUrl, newUrl) {
    if (oldUrl !== newUrl) {
      if (newUrl.includes(oldUrl)) {
        return false;
      }
      return true;
    }
    return false;
  }
}, function() { });
```

onUrlChange

Description: executes when the URL changes via a relative link or SPA routing functionality.

Type: function

Status: default implementation available; tracks pageviews via `ac('pageview')`; can be overwritten

Returns: void

Arguments:

Name	Description	Type	Status
newURL	the URL the user has changed to	string	required

> **Important**
>
> By default, the tracking snippet tracks page views via `ac('pageview')`. If you override the default behavior using `onUriChange`, remember to manually include this call in the new function.

callback

When a module fully loads, callback is executed. No arguments are passed to the callback.

Example

Your SPA page may use routers or relative links to change the page URL without changing the page title. In this case, the default `ac('pageview')` call tracks all page views as occurring on the same page. Visit journey information that appears in Live Now and the agent UI does not reflect actual user behavior. To change this, customize the onUrlChange option as shown in the following example.

```
ac('load', 'autotrackUrlChange', {
  onUrlChange: function(newUrl) {
    let customTitle = document.title;
    if (newUrl.includes('marketing')) {
      customTitle = customTitle + 'marketing';
    }
    if (newUrl.includes('contact-us')) {
      customTitle = customTitle + 'contact-us';
    }
    // supply pageOverrides with location and custom title field
    ac('pageview', {
      location: window.location.href,
      title: customTitle
    });
  }
}, function() { });
```

delay

Use this option to set a delay between the time when the SDK realizes there is a URL change and when it sends the `onUrlChange` function.

Example

This example sets a 1-second delay. This gives the application time to change the page title to match the current URL.

```
ac('load', 'autotrackUrlChange', { delay: 1000 });
```

Exclude URL query parameters

Contents

Exclude irrelevant URL query parameters to improve analytics.

Exclude URL query parameters

Websites often use URL query parameters that are not relevant; they do not influence the content that is shown to the user. For example, in web analytics, parameters such as session IDs, campaign IDs, and so on are not informative. We recommend that you use the Journey JavaScript SDK to exclude these query parameters when you register unique page views. Excluded query parameters are not tracked and are not included in the analytics reports.

> **Important**
> Be careful when you exclude parameters. For example, if you want to know which products visitors view on your site, do not exclude a query string parameter for a product ID.

To configure Altocloud to skip certain query parameters, login as an administrator, go to **Settings > Tracking Settings** and enter a comma-separated list of the unwanted parameters in the **Exclude URL Query Parameters** field (for example: sessionid,var1,var2).

Track the #hash portion of the URL fragment

Contents

Track the #hash portion of URL fragments to include relevant information in analytics.

Track the #hash portion of the URL fragment

By default, Altocloud removes the fragment #hash part of the URL. However, you may want to track the URL fragment to analyze relevant information that it contains.

To include the URL fragments in the URLs tracked, login as an administrator, go to **Settings > Tracking Settings** and enable/disable the Keep Page URL fragments setting.

Before you enable this setting, beware that - by default - pageviews are recorded on page load. If your website renders different content based on the URL fragment (without issuing a page reload) and you want to track these interactions as individual pageviews, you must call the `ac('pageview')` method each time the URL fragment changes, passing in the desired page location.

Track a page's canonical URL

Contents

Track the canonical URL of a page for precise pageview counts.

Canonical URLs eliminate duplicate content

Canonical URLs help webmasters and site administrators eliminate duplicate content from analytics reports.

For example

```
http://www.example.com/blog
```

would be treated as a different page than

```
http://www.example.com/blog?sidebar=0
```

even though those URLs display the same content. The side-effect of this is that one page view to each page will be reported instead of two pageviews to a single page.

Canonical URLs in links elements

The canonical URL of a page is often specified in a canonical link element, which can be inserted into the section of a web page.

For example:

How to track a canonical URL

To use your canonical URLs and override the URL that the ac SDK will track, configure the init call to use the canonical link element value when available.

```
ac('init', 'YOUR-ORGANIZATION-ID', {
  region: 'YOUR-REGION',
  canonicalLink: true
});
```