and if one is found, sets it as the user's ID again in the new cookie. If a user clears their cookies and `localStorage`, all of the IDs are removed, and the user gets a completely new `anonymousID` when they next visit the page.

Anonymous IDs

Analytics.js generates a universally unique ID (UUID) for the viewer during the library's initialization phase, and sets this as `anonymousId` for each new visitor to your site. This happens before Analytics.js loads any device-mode destinations, and so before these destination-libraries can generate their own user IDs.

Example:

```
ajs_anonymous_id=%2239ee7ea5-b6d8-4174-b612-04e1ef3fa952
```

You can override the default-generated anonymousID in code using the methods described below:

- Set anonymousId from the Segment snippet (before the `ready` method returns)
- Use a call to override the anonymousID
- Set `anonymousId` in the `options` object of a call

Retrieve the Anonymous ID

You can get the user's current `anonymousId` using the following call:

```
analytics.user().anonymousId();
```

If the user's `anonymousId` is `null` (meaning not set) when you call this function, Analytics.js automatically generated and sets a new `anonymousId` for the user.

If you are using the npm library, the previous call returns a promise for `user()`. As a workaround, you'll need to grab the user's current `anonymousId` in the following way:

```
analytics.instance?.user().anonymousId()
```

Refreshing the Anonymous ID

A user's `anonymousId` changes when any of the following conditions are met.

- The user clears their cookies *and* `localstorage`.
- Your site or app calls `analytics.reset()` during in the user's browser session.
- Your site or app calls `analytics.identify()` with a userId that is different from the current userId.
- Your site or app is setting `ajs_user_id` to an empty string or calling `analytics.user().id('')` before calling `analytics.identify()`. This sequence of events will result in a new anonymousId being set when `analytics.identify()` is called.

Override the Anonymous ID from the Segment snippet

You can also set the `anonymousId` immediately inside your Segment snippet, even before the `ready` method returns.

```
analytics.load('writekey');
analytics.page();
analytics.setAnonymousId('ABC-123-XYZ');
```

Use this method if you are queueing calls before `ready` returns and they require a custom `anonymousId`. Keep in mind that setting the `anonymousId` in Analytics.js does not overwrite the anonymous tracking IDs for any destinations you're using.

Device-mode destinations that load their code on your site *might* also set their own anonymous ID for the user that is separate and different from the Segment generated one. Some destinations use the Segment `anonymousId`. Read the documentation for each Destination to find out if a Destination sets its own ID.

Override the default Anonymous ID with a call

If the default generated UUID does not meet your needs, you can override it `anonymousId` for the current user using either of the following methods.

```
analytics.user().anonymousId('ABC-123-XYZ');
```

```
analytics.setAnonymousId('ABC-123-XYZ')
```

These methods behave exactly the same.

Override the Anonymous ID using the options object

Or in the `options` object of `identify`, `page`, or `track` calls, like this:

Set the anonymousId in the Options object using the format in the following examples.

The custom anonymousId persists when you use these methods, even if you do not explicitly specify the anonymousId in the calls.

For example, after the Track call below sets the anonId, any later track calls from this user will have the anonymousId of `ABC-123-XYZ`, even if it is not explicitly specified in the track call.

Override anonymousId in an Identify call

```
analytics.identify('user_123', {
  name: 'Jane Kim'
}, {
  anonymousId: 'ABC-123-XYZ'
});
```

Override anonymousId on a Page call

```
analytics.page({}, { anonymousId: 'ABC-123-XYZ' });
```

Override anonymousId on a Track call

```
analytics.track('Email Clicked', {
  callToAction: 'Signup'
}, {
  anonymousId: 'ABC-123-XYZ'
});
```

Saving traits to the context object

Traits are individual pieces of information that you know about a user or a group, and which can change over time.

The `options` dictionary contains a sub-dictionary called `context` which automatically captures data depending on the event- and source-type. See the Context documentation to learn more.

The `context` object contains an optional `traits` dictionary that contains traits about the current user. You can use this to store information about a user that you got from previous Identify calls, and that you want to add to Track or Page events.

The information you pass in `context.traits` *does not* appear in your downstream tools (such as Salesforce, Mixpanel, or Google Analytics); however, this data *does* appear in your warehouses and storage destinations.

> The `options` object described in the previous section behaves differently from the `options.context.traits` object discussed here. The `traits` object described here does not cause `anonymousId` to persist across different calls.

Consider this Identify event:

```
analytics.identify('12091906-01011992', {
    plan_id: 'Paid, Tier 2',
    email: 'grace@usnavy.gov'
});
```

The "trait" on this event is `plan_id`. You can pass these traits into `context.traits`, so you can use them in Track or Page events that the user triggers later.

The example below shows how you could pass the `plan_id` as a trait so you can use it later.

```
analytics.track('Clicked Email', {
   emailCampaign: 'First Touch'
  },
  {
    traits: {
      plan_id: 'Paid, Tier 2'
    }
  }
);
```

This appends the `plan_id` trait to this Track event. This does *not* add the name or email, since those traits were not added to the `context` object. You must do this for every following event you want these traits to appear on, as the `traits` object does not persist between calls.

Clearing Traits

You can pass an empty object to the `traits` object to clear *all* cached traits for a User or Group.

Traits are cached by default when you call the Identify and Group methods. You can clear the `traits` object for the user or group by passing `traits` an empty object:

```
analytics.user().traits({});
```

```
analytics.group().traits({});
```

Using analytics.user() and analytics.group()

You can use the `user` or `group` method as soon as the Analytics.js library loads, to return information about the currently identified user or group. This information is retrieved from the user's cookie.

Tip: You can wrap any reference to `user()` or `group()` in a ready function block to ensure that Analytics.js has fully loaded so these methods are available.

Examples:

```
analytics.ready(function() {

  var user = analytics.user();
  var id = user.id();
  var traits = user.traits();

});
```

```
analytics.ready(function() {

  var group = analytics.group();
  var id = group.id();
  var traits = group.traits();

});
```

Anonymizing IP

Segment automatically collects the user's IP address for device-based (iOS, Android, Analytics.js and Xamarin) events.

IPv6

At the moment, Segment doesn't support automatically collecting IPv6 addresses.

You can manually set the IP by passing a value for `options.context.ip` to prevent the Segment systems from recording the IP address for the request, as in the example below.

```
analytics.track("Order Completed", {}, { context: { ip: "0.0.0.0" }});
```

You must add this override to *every* Track call to explicitly override IP collection. If you reset this trait in the context object, Segment defaults to the normal IP collection behavior.

This page was last modified: 24 May 2024

Need support?

Questions? Problems? Need more info? Contact Segment Support for assistance!

Visit our Support page

Help improve these docs!

Edit this page

Request docs change

Was this page helpful?

Yes

No

Get started with Segment

Segment is the easiest way to integrate your websites & mobile apps data to over 300 analytics and growth tools.

Your work e-mail

Request Demo

or

Create free account