Here's an example of these common fields in raw JSON:

```
{
  "anonymousId": "507f191e810c19729de860ea",
  "context": {
    "active": true,
    "app": {
      "name": "InitechGlobal",
      "version": "545",
      "build": "3.0.1.545",
      "namespace": "com.production.segment"
    },
    "campaign": {
      "name": "TPS Innovation Newsletter",
      "source": "Newsletter",
      "medium": "email",
      "term": "tps reports",
      "content": "image link"
    },
    "device": {
      "id": "B5372DB0-C21E-11E4-8DFC-AA07A5B093DB",
      "advertisingId": "7A3CBEA0-BDF5-11E4-8DFC-AA07A5B093DB",
      "adTrackingEnabled": true,
      "manufacturer": "Apple",
      "model": "iPhone7,2",
      "name": "maguro",
```

```
      "type": "ios",
      "token": "ff15bc0c20c4aa6cd50854ff165fd265c838e5405bfeb9571066395b8c9da449"
    },
    "ip": "8.8.8.8",
    "library": {
      "name": "analytics.js",
      "version": "2.11.1"
    },
    "locale": "en-US",
    "network": {
      "bluetooth": false,
      "carrier": "T-Mobile US",
      "cellular": true,
      "wifi": false
    },
    "os": {
      "name": "iPhone OS",
      "version": "8.1.3"
    },
    "page": {
      "path": "/academy/",
      "referrer": "",
      "search": "",
      "title": "Analytics Academy",
      "url": "https://segment.com/academy/"
    },
    "referrer": {
      "id": "ABCD582CDEFFFF01919",
      "type": "dataxu"
    },
    "screen": {
      "width": 320,
      "height": 568,
      "density": 2
    },
    "groupId": "12345",
    "timezone": "Europe/Amsterdam",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0
.0.0 Safari/537.36",
    "userAgentData": {
      "brands": [
        {
          "brand": "Google Chrome",
          "version": "113"
        },
        {
          "brand": "Chromium",
          "version": "113"
        },
        {
          "brand": "Not-A.Brand",
          "version": "24"
        }
      ],
      "mobile": false,
      "platform": "macOS"
    }
  },
  "integrations": {
    "All": true,
    "Mixpanel": false,
    "Salesforce": false
  },
  "event": "Report Submitted",
  "messageId": "022bb90c-bbac-11e4-8dfc-aa07a5b093db",
  "receivedAt": "2015-12-10T04:08:31.909Z",
  "sentAt": "2015-12-10T04:08:31.581Z",
  "timestamp": "2015-12-10T04:08:31.905Z",
  "type": "track",
  "userId": "97980cfea0067",
  "version": 2
}
```


In more detail these common fields for every API call are:

FIELD	TYPE	DESCRIPTION

FIELD		TYPE	DESCRIPTION
`anonymousId`	*required; optional if `userID` is set instead*	String	A pseudo-unique substitute for a User ID, for cases when you don't have an absolutely unique identifier. A `userId` or an `anonymousId` is required. See the Identities docs for more details.
`context`	*optional*	Object	Dictionary of extra information that provides useful context about a message, but is not directly related to the API call like `ip` address or `locale` See the Context field docs for more details.
`integrations`	*optional*	Object	Dictionary of destinations to either enable or disable See the Destinations field docs for more details.
`messageId`	*implicit*	String	Automatically collected by Segment, a unique identifier for each message that lets you find an individual message across the API. This field is limited to 100 characters.
`receivedAt`	*implicit*	Date	Automatically set by Segment, the timestamp of when a message is received by Segment It is an ISO-8601 date string. See the Timestamps fields docs for more detail.
`sentAt`	*optional*	Date	Timestamp of when a message is sent to Segment, used for clock skew correction It is set automatically by the Segment tracking libraries. It is an ISO-8601 date string. See the Timestamps fields docs for more detail.
`timestamp`	*optional*	Date	Timestamp when the message itself took place, defaulted to the current time by the Segment Tracking API, as a ISO-8601 format date string. If the event just happened, leave it out and we'll use the server's time. If you're importing data from the past, make sure you to provide a `timestamp`.See the Timestamps fields docs for more detail.
`type`	*implicit*	String	Type of message, corresponding to the API method: `'identify'`, `'group'`, `'track'`, `'page'`, `'screen'` or `'alias'`.
`userId`	*required; optional if `anonymousID` is set instead*	String	Unique identifier for the user in your database. A `userId` or an `anonymousId` is required. See the Identities docs for more details.
`version`	*implicit*	Number	Version of the Tracking API that received the message, automatically set by Segment.

Beyond this common structure, each API call adds a few specialized top-level fields.

Context

Context is a dictionary of extra information that provides useful context about a datapoint, for example the user's `ip` address or `locale`. You should **only use** Context fields for their intended meaning.

FIELD	TYPE	DESCRIPTION
`active`	Boolean	Whether a user is active. This is usually used to flag an `.identify()` call to just update the traits but not "last seen."
`app`	Object	dictionary of information about the current application, containing `name`, `version`, and `build`. This is collected automatically from the mobile libraries when possible.
`campaign`	Object	Dictionary of information about the campaign that resulted in the API call, containing `name`, `source`, `medium`, `term`, `content`, and any other custom UTM parameter. This maps directly to the common UTM campaign parameters.
`device`	Object	Dictionary of information about the device, containing `id`, `advertisingId`, `manufacturer`, `model`, `name`, `type`, and `version`. **Note:** If you collect information about iOS devices, note that the `model` value set by Apple might not exactly correspond to an iPhone model number. For example, an `iPhone 15 Pro Max` has a `model` value of `iPhone16,2`.

FIELD	TYPE	DESCRIPTION
`ip`	String	Current user's IP address.
`library`	Object	Dictionary of information about the library making the requests to the API, containing `name` and `version`.
`locale`	String	Locale string for the current user, for example `en-US`.
`network`	Object	Dictionary of information about the current network connection, containing `bluetooth`, `carrier`, `cellular`, and `wifi`. If the `context.network.cellular` and `context.network.wifi` fields are empty, then the user is offline.
`os`	Object	Dictionary of information about the operating system, containing `name` and `version`.
`page`	Object	Dictionary of information about the current page in the browser, containing `path`, `referrer`, `search`, `title` and `url`. This is automatically collected by Analytics.js.
`referrer`	Object	Dictionary of information about the way the user was referred to the website or app, containing `type`, `name`, `url`, and `link`.
`screen`	Object	Dictionary of information about the device's screen, containing `density`, `height`, and `width`.
`timezone`	String	Timezones are sent as tzdata strings to add user timezone information which might be stripped from the timestamp, for example `America/New_York`, but in some cases, this may be unavailable due to browser limitations, privacy settings, or missing API support.
`groupId`	String	Group / Account ID. This is useful in B2B use cases where you need to attribute your non-group calls to a company or account. It is relied on by several Customer Success and CRM tools.
`traits`	Object	Dictionary of `traits` of the current user. This is useful in cases where you need to `track` an event, but also associate information from a previous Identify call. You should fill this object the same way you would fill traits in an identify call.
`userAgent`	String	User agent of the device making the request.
`userAgentData`	Object	The user agent data of the device making the request. This always contains `brands`, `mobile`, `platform`, and may contain `bitness`, `model`, `platformVersion`,`uaFullVersion`, `fullVersionList`, `wow64`, if requested and available. This populates if the Client Hints API is available on the browser. This may contain more information than is available in the `userAgent` in some cases.
`channel`	String	where the request originated from: server, browser or mobile

Context fields automatically collected

Below is a chart that shows you which context variables are populated automatically by the iOS, Android, and analytics.js libraries.

Other libraries only collect `context.library`, any other context variables must be sent manually.

CONTEXT FIELD	ANALYTICS.JS	ANALYTICS-IOS	ANALYTICS-ANDROID
app.name		≣	≣
app.version		≣	≣
app.build		≣	≣
campaign.name	≣		

CONTEXT FIELD	ANALYTICS.JS	ANALYTICS-IOS	ANALYTICS-ANDROID
campaign.source	▤		
campaign.medium	▤		
campaign.term	▤		
campaign.content	▤		
device.type		▤	▤
device.id		▤	▤
device.advertisingId		▤	▤
device.adTrackingEnabled		▤	▤
device.manufacturer		▤	▤
device.model		▤	▤
device.name		▤	▤
library.name	▤	▤	▤
library.version	▤	▤	▤
ip*	▤	▤	▤
locale	▤	▤	▤
network.bluetooth			▤
network.carrier		▤	▤
network.cellular		▤	▤
network.wifi		▤	▤
os.name		▤	▤
os.version		▤	▤
page.path	▤		
page.referrer	▤		
page.search	▤		
page.title	▤		
page.url	▤		
screen.density			▤
screen.height		▤	▤
screen.width		▤	▤
traits		▤	▤
userAgent	▤		▤
userAgentData*	▤		

CONTEXT FIELD	ANALYTICS.JS	ANALYTICS-IOS	ANALYTICS-ANDROID
timezone	≣	≣	≣

- IP Address isn't collected by Segment's libraries, but is instead filled in by Segment's servers when it receives a message for **client side events only**.

> **IPv6**
>
> Segment doesn't support automatically collecting IPv6 addresses.

- The Android library collects `screen.density` with this method.
- userAgentData is only collected if the Client Hints API is available on the browser.
- Segment doesn't collect or append to the context of subsequent calls in the new mobile libraries (Swift, Kotlin, and React Native).

To pass the context variables which are not automatically collected by Segment's libraries, you must manually include them in the event payload. The following code shows how to pass `groupId` as the context field of Analytics.js's `.track()` event:

```
analytics.track("Report Submitted", {}, {
  context: {
    groupId: "1234"
  }
});
```

To add fields to the context object in the new mobile libraries, you must utilize a custom plugin. Documentation for creating plugins for each library can be found here:

- React Native
- Swift
- Kotlin

Integrations

A dictionary of destination names that the message should be sent to. `'All'` is a special key that applies when no key for a specific destination is found.

Integrations defaults to the following:

```
{
  All: true,
  Salesforce: false,
}
```

This is because Salesforce has strict limits on API calls.

Sending data to the rest of Segment's destinations is opt-out so if you don't specify the destination as false in this object, it will be sent to rest of the destinations that can accept it.

Timestamps

Every API call has four timestamps, `originalTimestamp`, `timestamp`, `sentAt`, and `receivedAt`. They're used for very different purposes.

All timestamps are ISO-8601 date strings, and are in the UTC timezone. To see the user's timezone information, check the `timezone` field that's automatically collected by client-side libraries.

You must use ISO-8601 date strings that include timezones when you use timestamps with Engage. If you send custom traits without a timezone, Segment doesn't save the timestamp value.

Timestamp overview

TIMESTAMP	CALCULATED	DESCRIPTION
`originalTimestamp`	Time on the client device when call was invoked **OR** The `timestamp` value manually passed in through server-side libraries.	Used by Segment to calculate `timestamp`. **Note:** `originalTimestamp` is not useful for analysis since it's not always trustworthy as it can be easily adjusted and affected by clock skew.
`sentAt`	Time on client device when call was sent. **OR** `sentAt` value manually passed in.	Used by Segment to calculate `timestamp`. **Note:** `sentAt` is not useful for analysis since it's not always trustworthy as it can be easily adjusted and affected by clock skew.
`receivedAt`	Time on Segment server clock when call was received	Used by Segment to calculate `timestamp`, and used as sort key in Warehouses. **Note:** For max query speed, `receivedAt` is the recommended timestamp for analysis when chronology does not matter as chronology is not ensured.
`timestamp`	Calculated by Segment to correct client-device clock skew using the following formula: `receivedAt - (sentAt - originalTimestamp)`	Used by Segment to send to downstream destinations, and used for historical replays. **Note:** Recommended timestamp for analysis when chronology does matter.

originalTimestamp

The `originalTimestamp` tells you when call was invoked on the client device or the value of `timestamp` that you manually passed in.

Note: The `originalTimestamp` timestamp is not useful for any analysis since it's not always trustworthy as it can be easily adjusted and affected by clock skew.

sentAt

The `sentAt` timestamp specifies the clock time for the client's device when the network request was made to the Segment API. For libraries and systems that send batched requests, there can be a long gap between a datapoint's `timestamp` and `sentAt`. Combined with `receivedAt`, Segment uses `sentAt` to correct the original `timestamp` in situations where a user's device clock cannot be trusted (mobile phones and browsers). The `sentAt` and `receivedAt` timestamps are assumed to occur at the same time (maximum a few hundred milliseconds), and therefore the difference is the user's device clock skew, which can be applied back to correct the `timestamp`.

Note: The `sentAt` timestamp is not useful for any analysis since it's tainted by user's clock skew.

Segment now adds `sentAt` to a payload when the batch is complete and initially tried to the Segment API for the Swift, Kotlin, and C# mobile libraries

This update changes the value of the Segment-calculated `timestamp` to align closer with the `receivedAt` value rather than the `originalTimestamp` value. For most users who are online when events are sent, this does not significantly impact their data. However, if your application utilizes an offline mode where events are queued up for any period of time, the `timestamp` value for those users now more closely reflects when Segment received the events rather than the time they occurred on the users' devices.

receivedAt

The `receivedAt` timestamp is added to incoming messages as soon as they hit the API. It's used in combination with `sentAt` to correct clock skew, and also to aid with debugging libraries and systems that deliver events in batches.

The `receivedAt` timestamp is most important as the sort key in Segment's Warehouses product. Use this for max query speed when retrieving data from your Warehouse.

Note: Chronological order of events is not ensured with `receivedAt`.

timestamp

The `timestamp` timestamp specifies when the data point occurred, corrected for client-device clock skew. This is the timestamp that is passed to downstream destinations and used for historical replays. It is important to use this timestamp for importing historical data to the API.

If you are using the Segment server Source libraries, or passing calls directly to the HTTP API endpoint, you can manually set the `timestamp` field. This change updates the `originalTimestamp` field of the Segment event. If you use a Segment Source in device mode, the library generates `timestamp` and you cannot manually set one directly in the call payload.

Segment calculates `timestamp` as `timestamp = receivedAt - (sentAt - originalTimeStamp)`.

For client-side tracking it's possible for the client to spoof the `originalTimeStamp`, which may result in a calculated `timestamp` value set in the future.

FAQ

Why Are Events Received with Timestamps Set in the Past or Future?

If you're using one of Segment's client-side libraries, please note that several factors can cause timestamp discrepancies in your event data.

1. **Overriding Timestamp Value:**
   When a manual timestamp is set in the payload with a date in the past, it can cause events to appear as if they were sent earlier than they actually were.
2. **Analytics.js Source with Retries Enabled:**
   The Retries feature supports offline traffic by queuing events in Analytics.js. These events are sent or retried later when an internet connection is available, keeping the original timestamp intact.
3. **Mobile App Backgrounded or Closed:**
   If a user closes the app, events may be queued within the app. These queued events won't be sent until the app is re-opened, potentially in the future, leading to timestamp discrepancies.
4. **Inaccurate Browser/Device Clock Settings:**
   Timestamps can be incorrect if the client's device time is inaccurate, as the `originalTimestamp` relies on the client device's clock, which can be manually adjusted.
5. **Traffic from Internet Bots:**
   Internet Bots can sometimes send requests with unusual timestamps, either intentionally or due to incorrect settings, leading to discrepancies.

This page was last modified: 06 Feb 2025

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

Privacy

Terms

Website Data Collection Preferences