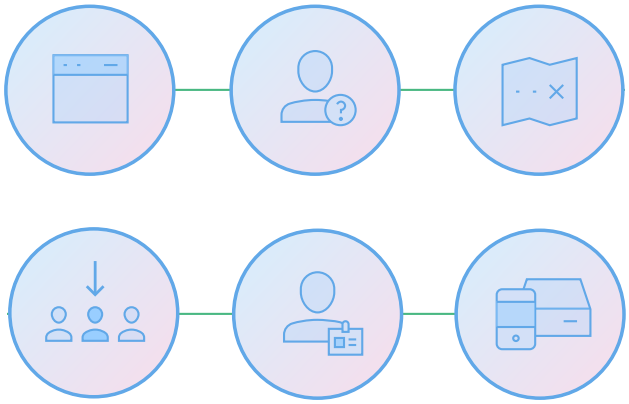
Segment



6 Data Tracking Essentials for Your App or Website

Learn Segment's key implementation strategies for tracking customer data the **right way.**

Table of Contents

CHAPTER 1

Collecting your user's data

Data tracking enables you to observe the ways people interact with your app or website so that you can use the resulting data to make your online presence better.

Tracking can help you understand the behavior of people who interact with your app or service. For example, how are customers finding their way around your site? Who's signing up for your services? What are they interested in?

Correctly answering these and other questions gives you insight into your customers that can help you drive better business outcomes. A principal question of data tracking is: When and where should you collect information to give you the most insight into customer behavior? This paper will walk you through how to implement data tracking throughout an online customer journey. We will also show you the different options and trade offs for where to collect that data.

A sample customer journey

Let's say a user visits your site for the very first time. At this point, you know nothing about them. They might visit for a few seconds and then disappear forever, or end up your most valuable customer. Either way, you should start tracking that user the second they hit your app or website.

Eventually you'll analyze this data in a tool like Mixpanel, Google Analytics, or Looker. To get your data into those tools you could load the tools' analytics libraries onto your site and add tracking calls for each tool. Or you can use Segment's open source library, analytics.js, to do all your tracking in one go and automatically send your data to your tools. This white paper assumes you're using analytics.js, but the essential steps apply to any tool.

If you'd like to track on an app instead of a website, you can use Segment's mobile SDKs.

`page` CALLS

The first step in your customer's journey, and therefore the first step in your tracking journey, is when the user first gets to your site. You want to know exactly which page drew the user to your site. So you'll start by firing a `page` call. That call is as easy as:

```
analytics.page("Home");
```

When you make that `page` call, analytics.js will do a few things behind the scenes. First, it will automatically gather information from your webpage and pass it on as properties:

title	window.location.title	**ip**	IP address request was made from
url	window.location.url	**userAgent**	userAgent request was made from
path	window.location.path	**campaign**	name, source, medium, content, term (based on UTM params in the URL)
referrer	window.document.referrer		
search	window.location.search		

Here's an example of a `page` call with some of the standard fields like timestamps and other user information removed for clarity. You can find the full call in the appendix:

```
analytics.page('Docs', {
  name: 'Docs',
  path: '/docs/sources/server/http/',
  referrer: 'https://segment.com/docs/sources/server/go/',
  search: '',
  section: 'Sources',
  title: 'HTTP Tracking API - Segment',
  topic: 'HTTP Tracking API',
  url: 'https://segment.com/docs/sources/server/http/'
});
```

ANONYMOUS IDS

When you send a `page` call (or any other analytics call for that matter), analytics.js will also check the client for a cookie. Checking for cookies is the second essential step to tracking. It tells you if this user has been to your site before and matches all of your tracking events to that user.

Since this is the user's first time on your site, they won't have a stored cookie. Instead, analytics.js will create an Anonymous ID for this user, store that ID on the user's device in a cookie and add the ID to the `page` call. As the user continues to move around your site, you'll continue to fire `page` calls which will automatically include the same Anonymous ID, so you can correlate a series of `page` calls to a specific user.

`track` CALLS

Now that your user is on the site, it's time for step three, `track` calls.

As your user navigates around your site, you'll want to track what actions they take. A lot of interactions can occur on a particular page, and it's important to track these "events" in addition to just pageviews. For example, you might fire a `product page` page call when someone hits a product page. But they might also add a product to their cart, look at different color options, or choose different sizes from the same page. That's what our `track` calls are for—discrete events your user triggers.

When you start tracking events, we recommend you choose a naming convention so that your event names are standardized. This will make it easier for new team members to understand what existing events are referring to and to code new ones consistently.

Internally at Segment, we use the "Object-Action" convention, so our events have names like `Button Clicked` or `Form Submitted`. Each event also has a set of properties that you can define. For example, `Button Clicked` could have properties like `button_name` and `button_action`.

The JavaScript for this call would look something like this:

```
analytics.track("Button Clicked", {
  button_name: "Contact Us"
  button_action: "Takes user to contact form"
});
```

From that call, analytics.js will generate the following payload:

```
{
  "type": "track",
  "event": "Button Clicked",
  "properties": {
    "button_name": "Contact Us"
    "button_action": "Takes user to contact form"
  }
}
```

Note: *The payload would also include some automatically generated fields like the timestamp and the Anonymous ID, which works the same way as it does for* `page` *calls. You can find an example of a full call in the appendix.*

`identify` CALLS

Eventually, your user may start supplying you with information about themselves. Step four of essential tracking is to use that information to build an identity for the user. That's where Segment's `identify` call comes in. `identify` lets you record traits about your users and then combines that information with all the event data you're also collecting, so you know who's doing what.

For example, let's say that there's a place on your site where a user can sign up for an email newsletter. You'll want to fire a `track` event, something along the lines of `Form Submitted`. You'll also want to add this email to the user's identity. You'll do so with the `identify` call:

```
analytics.identify({
  email: 'peter@initec.com'
});
```

At this stage, you have a choice. You can either continue using the Anonymous ID you already have for the user, or you can create a permanent User ID. If the information we're including in the `identify` call is not truly unique, like a first name, then you'll want to continue using the Anonymous ID. analytics.js will include the Anonymous ID automatically.

In this example, we have an email address which we can assume to be unique. Therefore, we'll create a User ID.

CREATING A USER ID

Creating a permanent ID for your user is step five of tracking. We'll do this for two reasons.

The first reason is that you get some unique information about the user. In the above example, the user provided an email address.

Alternatively, let's say that Peter (in this example, we're assuming his name from his email address) keeps browsing your site over a few days. Each time that he comes back, your tracking calls will continue to use the same Anonymous ID because it is stored on his client as a cookie. Peter becomes more and more impressed by your amazing site and product and decides to sign up for a user account. When that happens, you'll want to create a permanent ID for Peter.

He'll start by going to your sign-up page and you'll fire a call along the lines of `analytics.page("Sign-Up Page")`. Then he'll fill out your form that might have information like his name, title, and company. You'll use an event call to track that:

```
analytics.track("Form Submitted", {
  form_name: "user signup"
  user_name: "Peter Gibbons"
  company: "Initec"
  email: "peter@initec.com"
  title: "programmer"
});
```

The fields in this `track` call will be stored as properties of that event so in the future you can, for example, compare all the different data that that form collects. You could write a query that tells you what the most common job titles are for people who signup. However, you'll also want to store this data as information about Peter in particular. Here again we'll use an `identify` call but this time we'll use a User ID.

When Peter submits the sign-up form, you'll probably take some actions on your servers to create an account for him and enter him into your user database. When you do this, you'll create a User ID for him. You'll want to pass that same User ID into the `identify` call. That call will look like this:

```
analytics.identify("97980cfea0067", {
  user_name: "Peter Gibbons"
  company: "Initec"
  email: "peter@initec.com"
  title: "programmer"
});
```

When that call reaches Segment's servers, Segment will pass it on to your integrations, which will connect this new User ID with the existing Anonymous ID. All the data you collected about Peter (even before you knew it was Peter) will now be accessible in Peter's record. The User ID will also be stored on the client in a cookie so that all future calls will also be correlated with Peter's record.

A BRIEF NOTE ABOUT THE `alias` CALL

While Segment automatically matches Anonymous IDs with User IDs, some of the marketing tools we integrate with need to be explicitly told to make that match (KISSmetrics, Mixpanel, and Vero). We only need to do this once for each user, whenever that user is first given a User ID.

The `alias` call is very straightforward:

```
analytics.alias("97980cfea0067");
```

We just pass the new User ID (`97980cfea0067` in this example) into the call and analytics.js will automatically include the Anonymous ID. This call will go to any tools you have enabled and will make sure that they correlate these two IDs.

If you're instrumenting a website, then the Anonymous ID is generated in the browser, so you'll want to call `alias` from the client side. If you're using a server-side Session ID as the Anonymous ID, then you'll want to alias from the server side. We'll get to a full discussion of client-side and server-side data collection below. For now, what's important is that you alias from the same place you're implementing the Anonymous ID.

OTHER TIMES YOU NEED TO USE `identify`

Besides at user sign-up, there are a few other times you'll want to make `identify` calls:

- When you receive new information about a user
- After a user logs in
- Upon loading any pages that are accessible by a logged in user

Let's say at a later date, Peter signs up for a specific account plan. To add this information to his user record, you'll make this call:

```
1  analytics.identify({
2    account_plan: "premium"
3  });
```

You don't need to include any of the other information you know about Peter. analytics.js will pull his User ID from the cookie on the client and when the new information gets to Segment, we'll correlate it with his existing record.

For the second two examples, you might be asking: why would I call `identify` on every log-in and page load if I'm storing the User ID in the cookie?

Let's imagine this scenario:

> Peter's been browsing your site from his laptop and that's where your cookie is stored. Then he goes home and decides to go back to your site from his phone's browser. He could enter any page on your site. When those `page` calls fire, they won't have an existing Anonymous ID to call from. Instead, analytics.js will create a new Anonymous ID. Peter could also end up with an extra Anonymous ID if he ever clears his cookies, since this would also cause analytics.js to create a new one.
>
> This might sound bad but it's not. Eventually, Peter will either consciously go to a login page or go to a page on your site that requires him to be logged in. Once he logs in, you'll pull his User ID from your user database. Since you have `identify` calls on every logged-in page, you'll fire an `identify` with that User ID. Voila! Segment will receive both the Anonymous ID stored on his personal computer's cookie and his User ID. Segment will send that info on to your integrations where they can match those identifiers together. Now any action he takes on his personal or work computer will become part of the same user record. Some integrations handle connecting those IDs differently. Refer to our specific integration docs for more info.

WHY DON'T I JUST USE EMAIL?

Throughout this whole discussion of Peter's journey, you might be asking, "Why don't I just use Peter's email address as the User ID?" The main reason is that email addresses change from time to time, as do other public identifiers like usernames. On the other hand, User IDs never change, so they are the preferred identifier.

Since email addresses or usernames are still valuable information, you'll send them along as a trait within the `identify` call.

CHAPTER 2

Where to collect your data

The first five data tracking essentials we've gone over cover how to track your users. Our last essential is how to decide where to track your user. That is, **where** to implement your analytics calls and how that data will be sent to all the different tools you're using.

The first big choice you have is whether to add your tracking code to your site/app (the client), or to do that tracking on your servers. There are pros and cons to both options and you'll probably end up doing a mix.

Client libraries vs. Server libraries

There are two ways to track your data with Segment. You can use our client-side libraries—JavaScript, iOS, and Android—to make calls within your users' browsers or mobile devices. Or you can track data with our server-side libraries like Node, Python, etc. These calls are triggered from your own servers. (The server is a computer that returns web pages to be rendered in your browser or screens in your app.)

There are a few trade-offs between collecting data on the client or server side. Neither is better than the other, and we usually recommend you do a mix of both. Overall, more data points are easily available on the client, but server-side collection is more secure and reliable, and it's not susceptible to ad blocking.

First, let's talk about general considerations for client vs. server tracking. Then, we'll dive into nuances per platform.

WHAT ARE THE TRADE-OFFS?

The big trade-off for choosing a tracking library is context (or the amount of data you can easily access and gather) vs. reliability (or how much control you have over the actual sending of the data).

The client, being that it's the browser, has easy access to user-specific attributes, such as cookies, IP address, user agent (what browser and operating system the user is using), referrer (what sent the user to the site), and UTM parameters. This means that if you track on the client, analytics.js will automatically collect and track all of these contextual pieces of information. Some common business use cases for tracking these pieces include:

- sending targeted marketing messages based on users' locations
- learning the breakdown of your users based on mobile, desktop, tablet
- determining how marketing campaigns drive traffic and conversion via UTM parameters

However, the downside of tracking in the client is that you have less control. The end user can enable an ad blocker, which can alter your analytics data (see below). The browser also has its own unique behaviors such as page unloading (when you click a link on a site and the browser begins loading the next page) that can sometimes interrupt any "in-flight" outbound analytics requests. Therefore, we usually suggest tracking on the server for most important events. It tends to be more reliable.

Should you track on the client or the server?

*An "offsite" event means that the user is triggering the action while not necessarily interacting with your site or app, e.g. API usage.

** If you're sending triggered emails based on events, it's probably a good idea to make sure the events are sent server-side so no one gets left out or mis-mailed.

These are the kinds of events that we recommend sending via your server:

- "Offsite" events: This just means events that aren't tied to a user behavior that occurs on your website or an event that is the result of a calculation based on your production database. An example of this would be our nightly cron job that calculates API usage across our customer base, then sends server side `.track()` calls.
- Tracking revenue (or other sensitive events): Revenue figures (which should be captured by your billing system) should be sent on the server, since they're sensitive and discrepancies from ad blockers or browser mishaps would be frustrating to debug.
- Mission critical events: If you're using email or marketing automation tools that rely on events like `user created` to trigger high-value emails, it's best to keep these events on the server side. It would be a bummer to have a customer miss out on a coupon or re-engagement email due to ad block or weird browser behavior.

In addition to these considerations, you might also want to think about engineering bandwidth. Though we usually recommend tracking server-side wherever possible, the server can't easily access the information in the browser. Therefore, you'll need additional engineering resources to mimic the same call on the server.

For example, you would need client-side logic to capture the UTM parameters, send it to the server, and then have the server send it to your analytics service. Since analytics data is useful directionally and for trends, many times the convenience of tracking UTM parameters, IP address, etc. on the client outweigh the possibility of losing a small percentage of data here and there to an ad blocker or unexplained browser behavior.

AD BLOCKERS

Ad blockers generally work by preventing a site from loading content from certain domains (say, *google.adsense*) or from loading HTML content or scripts that are classified as ad-related. Ad blockers also often block tracking code, either for privacy reasons or because tracking is often used for advertising. We anecdotally estimate that around 20% of our own users deploy some form of blocking.

Segment doesn't condone gathering data from users who, through the act of using ad blockers, wish to prevent their data from being collected.

However, there are some customer events which are essential, like `Account Created` or `Payment Charged` that power vital emails and experiences. You may want to consider tracking

these types of events on the server side, to avoid exposure to ad blockers.

If there is something that must be tracked on the client, one suggestion is to nicely ask your users to disable ad blocking so you can provide them with a more personalized browsing experience.

`identify` CALLS ON CLIENT AND SERVER

When instrumenting `identify` calls, you'll want to pay extra attention to the client- and server-side decision. Unlike `track` calls, you really don't want to lose data to ad blockers. This would suggest you want to make `identify` calls from the server. On the other hand, some analytics tools only work on the client side, so you would think you should make `identify` calls from the client.

The solution we recommend, and how we implement our own tracking at Segment, is to make duplicate `identify` calls. Every time that a user triggers an action that requires an `identify` we send two calls, one from the server and one from the client. When the server call executes, we have high confidence it will make its way to our database regardless of ad blockers or the client's internet connection.

We also make sure to send the User ID from the server back to the client. This way, when the client side `identify` call executes, we know that we're using the right ID.

Connection Modes for web

In addition to where exactly you track your events with Segment libraries (on the client or the server), Segment is beta-launching a new setting called Connection Modes. You'll have the opportunity to choose where exactly Segment will transform and send data to your integrations like Mixpanel or Amplitude (device-based or cloud-based).

When you start to use Segment, you'll only have to drop in the Segment library and track "events" once for each tool, instead of needing to do this over and over for each integration you want to use. On your end, you'll just see this data being passed off your integrations. But, you actually have some options around how and where that data is connected to integrations.

DEVICE-BASED CONNECTION MODE

In your web integration settings, you'll see the option to turn on the **Device-Based** Connection Mode. When you activate this choice, Segment wraps the library for your integration into our own library and loads it onto the browser. We then translate your tracking events into the format that the

tool expects, and send that data directly from the user's device to the integration's servers.

This is traditionally how Segment's integrations have worked. It's also the best option for tools that require access to the browser like A/B testing, heat mapping, and live chat tools.

CLOUD-BASED CONNECTION MODE

Segment is currently beta-testing a new **Cloud-Based** Connection Mode to improve your site performance. When you enable the cloud-based connection mode for an integration, Segment will no longer load its library onto your site. Instead, analytics.js will send event data to Segment's cloud servers, and from there we will translate and route that data to your integrations. This will remove third-party JavaScript from your site and improve page load times.

Before you turn on this mode, consider if the integration you're using has some features that require device-based interactions. For example, if you use the Cloud-Based Connection Mode for Mixpanel, you won't be able to use their features for in-app surveys or auto-tracking. But you'll get your data in their reporting and people features just fine.

Connection Modes for mobile

Tracking on your mobile applications is pretty similar to tracking on your website, but we wanted to call out a few small differences.

First off, the client vs. server library dynamic is a bit different in mobile because in general, apps do not make as many calls to the server. Therefore, you're going to do a lot more tracking on the client. Since apps do not contain ad blockers, you don't have to worry about them as much.

On the other hand, app size and unreliable connections are larger factors than they are on web, making server-side connections attractive.

WHEN TO TRACK SERVER-SIDE

Mobile connections are not always the most reliable, and they're not always secure. That's why it's still important to do some of your tracking on the server side. The best events to track server-side are ones that interact with your databases or contain important information. Account actions and payments are examples of events that should be tracked on your servers.

WHEN TO TRACK CLIENT-SIDE

There are two ways of connecting integrations on mobile. The default is similar to the cloud-based integrations on web. You just install the Segment SDK in your app, and all of your event data will flow directly to Segment's servers. From there, we send the event data off to each integration.

The advantages of this mode are even greater on mobile than on web. Removing SDKs keeps your app size down, which can have a huge positive effect on downloads. It also reduces demand on your users' data plans and battery life.

Here's a list of available cloud-based mobile integrations on iOS and Android.

iOS

Optimizely	Mixpanel	Kahuna	AppsFlyer
Google Analytics	Taplytics	Bugsnag	Primer
Localytics	Apptimize	Crittercism	Leanplum
Amplitude	Facebook-App Events	Branch Metrics	CleverTap
Countly	Adjust	Tapstream	Urban Airship
comScore	Quantcast	Appboy	Tune
Flurry	Moengage		

Android

Moengage	Countly	Flurry	Leanplum
comscore	Wootric	Crittercism	Adjust
Localytics	Tapstream	Bugsnag	CleverTap
Mixpanel	Apptimize	Branch Metrics	Urban Airship
Amplitude	Kahuna	Appboy	Tune
Taplytics	Google Analytics	Optimizely	AppsFlyer
Quantcast			

However, like on web, some integrations are best suited to be wrapped into Segment's SDK and be available on the **device.** Services such as optimization, deep linking, error tracking, and survey tools must be included on the device to leverage their core feature set.

By default, we don't bundle any integrations to keep our SDK and your app small. If you would like to bundle integrations to access necessary features, you'll have one extra step to do than on web.

First, add the dependencies you need. You can find these in our app when you open the integration for your source. The iOS code will look like this:

```
pod 'Segment-Bugsnag'
pod 'Segment-Branch'
pod 'Segment-GoogleAnalytics'
...
```

After adding the dependency, you must register the integration with our SDK.

```
#import <Segment-GoogleAnalytics/SEGGoogleAnalyticsIntegrationFactory.h>
#import <Segment-Branch/BNCBranchIntegrationFactory.h>

SEGAnalyticsConfiguration *config = [SEGAnalyticsConfiguration
configurationWithWriteKey:@"YOUR_WRITE_KEY"];

// Add any of your bundled integrations.
[config use:[SEGGoogleAnalyticsIntegrationFactory instance]];
[config use:[BNCBranchIntegrationFactory instance]];
...

[SEGAnalytics setupWithConfiguration:config];
```

Conclusion

Collecting data is how you answer vital questions about your customer's behavior and the success of your online presence.

This white paper covered 6 essential pieces for collecting that data:

1. Use `page` calls to follow which pages your users are browsing.
2. Use `AnonymousID`s to keep track of users before they log in.
3. Use `track` calls to understand the actions a user is taking on your site or app.
4. Use `identify` calls to keep track of information about a user's identity like name or email.
5. Use a permanent `UserID` once a user creates an account to track the user's journey as a customer.
6. Decide which tracking calls to implement client-side and server-side.

To get a full view of your customers, you have to use each type of tracking call in the Segment API. Knowing where to collect your data and how to connect that data to your third-party tools lets you make decisions off of reliable, secure data without bogging down your site or app.

Appendix

Full `page` call payload

Note: Identifying information has been removed


```
{
  "anonymousId": "########-####-####-####-############",
  "category": null,
  "context": {
    "ip": "###.####.###.##",
    "library": {
      "name": "analytics.js",
      "version": "3.0.0"
    },
    "page": {
      "path": "/docs/sources/server/http/",
      "referrer": "https://segment.com/docs/sources/server/go/",
      "search": "",
      "title": "HTTP Tracking API - Segment",
      "url": "https://segment.com/docs/sources/server/http/"
    },
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Safari/537.36"
  },
  "integrations": {},
  "messageId": "ajs-#################################-clean",
  "name": "Docs",
  "properties": {
    "name": "Docs",
    "path": "/docs/sources/server/http/",
    "referrer": "https://segment.com/docs/sources/server/go/",
    "search": "",
    "section": "Sources",
    "title": "HTTP Tracking API - Segment",
    "topic": "HTTP Tracking API",
    "url": "https://segment.com/docs/sources/server/http/"
  },
  "receivedAt": "2017-01-10T23:51:49.564Z",
  "sentAt": "2017-01-10T23:52:01.570Z",
  "timestamp": "2017-01-10T23:51:49.559Z",
  "type": "page",
  "userId": "##########",
  "originalTimestamp": "2017-01-10T23:52:01.565Z"
}
```

Full `track` call payload

Note: Identifying information has been removed

```
{
  "anonymousId": "########-####-####-####-############",
  "context": {
    "ip": "###.##.##.##",
    "library": {
      "name": "analytics.js",
      "version": "3.0.0"
    },
    "page": {
      "path": "/academy/",
      "referrer": "",
      "search": "",
      "title": "Analytics Academy",
      "url": "https://segment.com/academy/"
    },
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12) AppleWebKit/602.1.50
(KHTML, like Gecko) Version/10.0 Safari/602.1.50"
  },
  "event": "Academy Subscribed",
  "integrations": {},
  "messageId": "ajs-#################################-clean",
  "properties": {
    "email": "######@berkeley.edu",
    "medium": "email"
  },
  "receivedAt": "2017-01-10T23:54:05.504Z",
  "sentAt": "2017-01-10T23:54:04.005Z",
  "timestamp": "2017-01-10T23:54:05.503Z",
  "type": "track",
  "userId": null,
  "originalTimestamp": "2017-01-10T23:54:04.004Z"
}
```

We've already built your data pipeline.

Join more than 8,000 customers who rely on Segment for their data infrastructure needs. Our customer data platform is trusted by both startups and enterprises. We have a plan to suit every data need.

Contact us

segment.com/contact

+1 (415) 649-6900

101 15th Street

San Francisco, CA 94103