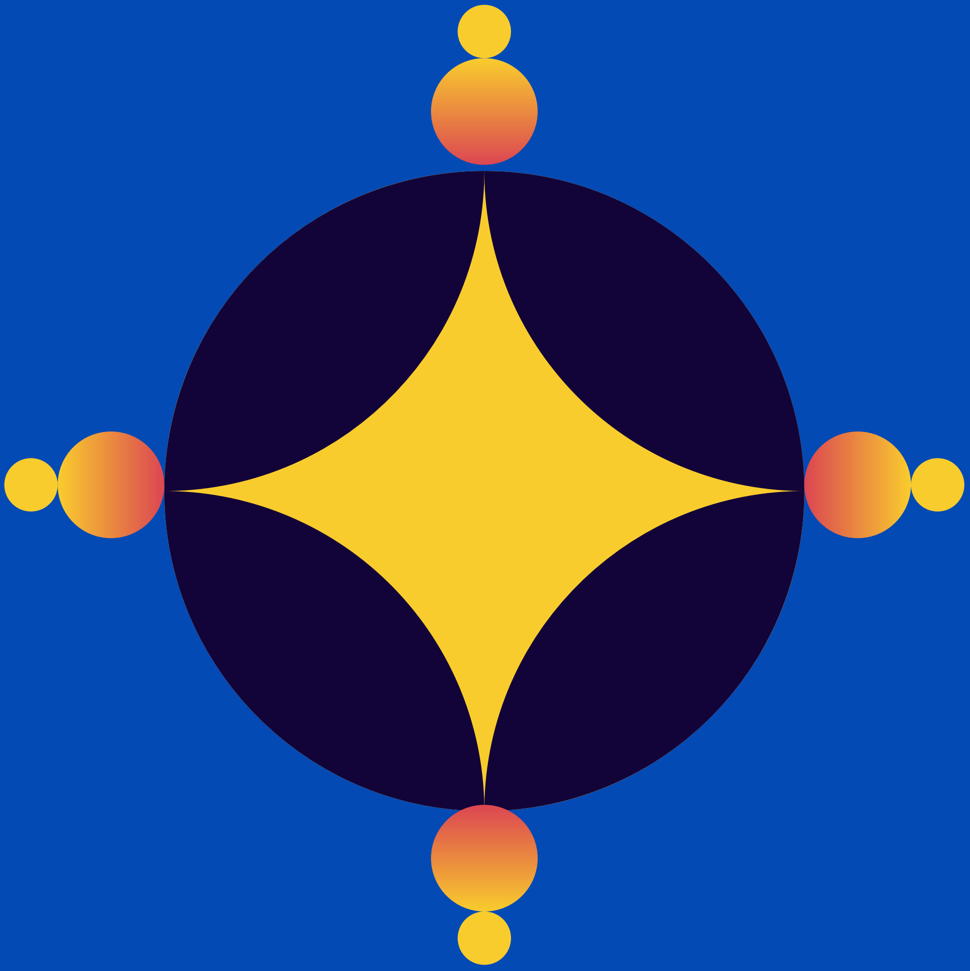
rudderstack

Whitepaper

Behavioral data collection best practices



These simple best practices will take away some of the guesswork that comes with behavioral data collection. Following them will help you avoid common pitfalls and improve data quality at the source, thereby improving the value of all downstream analysis and activation use cases.

START WITH BUSINESS OBJECTIVES, KPI's, AND REQUIREMENTS

Business value should be a top priority when planning out event tracking instrumentation. We recommend starting with the following questions:

- What is the main use case? (analysis, marketing activation, personalization, point solutions, other business decisions?)
- Who are the stakeholders that will be consuming the data? How do their needs differ?
    - Which tools or platforms will be used for storage, reporting, analysis, and activation?
- How is "success" defined for your business? What constitutes a successful user experience or customer journey through your digital ecosystem?
    - Does this definition of success change for different user types or personas?
    - Does this definition of success change depending on the different data consumers?
- What are the key touchpoints or "funnel steps" for a successful journey?
    - What information is relevant at each of these touchpoints?
- What legal or regulatory issues do we have to contend with either now or on the horizon?
    - GDPR, CCPA, HIPAA, etc.

It can be helpful to interview stakeholders across teams and obtain different answers to these questions. There is very little risk to overdoing it here – you will likely find that responses converge, but you may uncover less obvious requirements that are highly valuable to certain teams. Plus, participation from multiple teams early on in the process will pave the way for future buy-in and cooperation, and it encourages executive sponsorship and visibility.

This information should be documented clearly and agreed to by all participants. Consulting partners will almost always push for this - and for good reason. It reduces the risk of re-work in the future and creates a clear, non-technical guide to the data collection approach that all stakeholders can reference and understand.

INSTRUMENTATION APPROACH

Instrumentation at the source provides the opportunity to build out a comprehensive data collection strategy that shapes customer data for downstream tools. Generally there are two schools of thought on how to approach this:

Collect everything now, query it later (a.k.a. “Autotrack”)

This approach is favored by tools like Heap.io – you set up some basic rules to fire events every time the user loads a page, clicks a button, expands a menu, or otherwise interacts with the app in some way. Anything and everything is captured, along with available contextual information that can be easily scraped from the client. This makes implementation itself a breeze, but it results in a large volume of semi-structured data sent downstream and places the burden of the work on data analysts to make sense of the data once it hits the warehouse. It also limits you to a client-side solution, which may not be suitable for certain devices or environments.

While this may be sufficient for certain use cases (or specifically for analysis in a tool like Heap), many evenstream destinations are incompatible with this approach – or may require substantial transformation to format the data properly. Furthermore, the sheer volume of data generated by this method can be costly to collect and store, without necessarily providing much incremental value for the business. The ease-of-use advertised by this approach comes at a cost, and we don’t recommend building your first party behavioral data foundation on the whims of an automated script that may not properly capture the data you need.

Deliberate, scalable solution design to collect business-relevant events and properties

This approach requires a higher level of effort up front to architect and instrument a tracking framework, and it requires a certain amount of maintenance over time. However, the downstream effects are substantial:

- Analysts can spend more time doing analysis
- Marketers can more easily build and share targeted audiences
- Data scientists can get models up and running more quickly

Activation and analysis is much easier with data designed and formatted with specific use cases in mind, rather than trying to reconstruct user features and behavioral patterns by querying fragmented bits and pieces of scraped front-end code. This is our recommended approach, and though it requires careful planning and cross-team collaboration to succeed, the benefits to downstream tools and use cases far outweigh the initial setup cost. If that sounds daunting, let the use cases dictate prioritization – focus on instrumenting data needed to power the most valuable use cases, and take a phased approach to the rest. There will always be a tradeoff between quick time-to-value and long-term scalable solutions.

Pro Tip: If time-to-value is most important, start with a handful of key conversion events alongside a global pageview event that fires on each page or screen. This basic approach will often be enough to capture the full customer journey (akin to the basic Google Analytics tag) while still providing separate custom events for business-critical conversions.

USE A TRACKING PLAN

Proper instrumentation requires a careful approach to solution design, and the use of a tracking plan unlocks a number of powerful features that can radically improve data governance and stabilize data quality. In an ideal world, each business requirement would become a line item in the tracking plan, with specific event names defined alongside any required parameters or metrics associated with those events.

The tracking plan document, when deployed correctly, adds a ton of value to an implementation. Here are a few examples of the resources it will deliver and the teams it impacts:

- Instrumentation guide that tells the responsible dev team how to wire up events within the source
- Data dictionary/solution design document for analysts or data consumers to understand how and why data is collected at the source
- Reference schema for our Data Governance API to validate incoming events and surface any errors in a separate table in the warehouse (other observability improvements coming soon!)
- Reference table for RudderTyper, a library wrapper the dev team can use to ensure they are only instrumenting pre-approved event names and properties

Ideally, the tracking plan should be a trusted source of truth. If it falls out of date, has issues with ownership or version control, or otherwise fails to fully capture the scope of the implementation, it will start to lose its value. To avoid this, tracking plan maintenance should be incorporated into your sprint cycle process, and ownership should be clearly delineated. Incorporating tools like Avo can greatly reduce the headache of maintenance by seamlessly handling version control, facilitating collaboration across stakeholders and dev teams, and pushing schema updates to all data collection tools, including RudderStack. In the future, we plan to integrate additional tracking plan features into our own UI and provide templates for common features that will further simplify the process.

INSTRUMENTATION STRATEGY BEST PRACTICES

Developing a tracking plan provides the opportunity to build out a scalable data collection strategy that shapes your customer data for downstream tools. Here are some general best practices to consider when designing your instrumentation strategy.

If you have an e-commerce site, use a standard ecommerce tracking framework

RudderStack provides a default ecommerce tracking framework that applies to the vast majority of ecommerce websites or apps, and integrates seamlessly with downstream tools like Google Analytics or Klaviyo. Custom events or properties can be added if necessary, but when it comes to ecommerce there's no need to reinvent the wheel – the conventions of ecommerce data collection are clear, and RudderStack's standard ecommerce spec aligns with those conventions. This is especially true for platforms like Shopify: our Shopify SDK collects certain ecommerce funnel events directly from the server, where data can be routed directly to any "cloud mode" destination in our catalog.

Focus on tracking state changes, funnel steps, and other meaningful "progress" through the user experience, not just button clicks

It's tempting to just trigger events on each button called "<Button name> Clicked". This seems like an obvious catchall, but it adds little value beyond the autotrack approach. There are often multiple ways of accomplishing the same action on a site, and it's better to capture that the action itself was completed, regardless of which button was clicked. (If you really care about comparing the performance of specific buttons, do that with an A/B testing or heatmapping tool, not clickstream analysis.) Ideally you want events to describe the actions users are taking independent of specific buttons. The “Product Added” event, for instance, should be the same whether the user updates their cart from the product detail page, a “quick add” modal on the product list page, or a recommended products component. For information about the button itself, just pass the location of the button as a property in the event! Over time, buttons and UX details may change, but if the underlying user journey is intact, your implementation can provide consistency and continuity across those changes. For some great advice on how to name your events with a clean “Object-Action'' naming taxonomy, take a look at this post from our friends at Segment.

Aggregate requirements across the experience and across platforms

Similar functionality should be rolled up to create a single scalable framework that is forward-looking and can adapt to new features or content. For example: you don’t need a separate event for each video. You should create a single event called “Video Played” that applies to all videos and player types across websites and mobile apps, and capture the platform as a property of the event, alongside video name, video player, and any other relevant details. This makes analysis far easier because it doesn’t require knowledge of how the experience looked at a specific point in time to answer simple questions about content performance. It also abstracts the user journey out of the technical nuances of a particular platform. Analysis and activation use cases should ultimately be platform-agnostic, as the user journey can and often will traverse across mobile, web, server-side, and even offline environments.

Recognize the difference between successful completion of an action vs. intent to complete

There’s a theme emerging here: don’t just tag button clicks. There may be a lot of processing that happens between intent and completion, and therefore plenty of occasion for errors. Form validation is an obvious example. If I click to proceed to the next step and the application throws an error, I didn’t progress down the funnel! An event that just triggers on the button click will not accurately reflect the user’s experience. That’s why you should capture any errors (and their descriptions) when they are thrown and wait to trigger events until the application returns a successful response. Better yet, you can fire those key conversion events directly from the server. RudderStack offers a suite of server-side SDK’s and that allow for much more reliable event triggers, which will vastly reduce discrepancies between back-end systems of record and downstream destinations.

Bundle various interactive elements into discrete funnel steps

There’s no need to track every possible interaction as a discrete custom event. You can often bundle these actions into a single event that signifies progress down the funnel. Consider a flight search widget for an airline:

The user can select dates from a calendar, input the source and destination airport, and perhaps click a few other checkboxes to express preferences (one way vs roundtrip, “book with miles,” etc). Each of these options is a distinct action, but it’s not until the user submits their input by clicking “Find Flights” that the information becomes relevant. You likely don’t need to capture every toggle, every dropdown interaction, or god forbid every keystroke (unless there is a valid product use case in capturing that level of granularity). Rather, you can wait until they have completed that step of the experience and fire the event when they successfully submit their inputs. In this example, you have a single event – “Flight Search Submitted” – passed alongside a slew of properties that capture the various preferences they selected. It’s much easier to build audiences, segments, and dashboards with cleanly designed event data like this.

rudderstack

RudderStack is the warehouse-first, customer data platform built for developers.

We take a new approach to building and operating your customer data infrastructure, making it easy to collect, unify, transform, and store customer data as well as securely route it to a wide range of marketing, analytics, sales, and product tools.

Over 18,000 sites and apps run RudderStack including Crate & Barrel, Acorns, Hinge, Stripe, Allbirds, and more.

rudderstack.com

@rudderstack