Capturing Errors and Events

Learn how to use the SDK to manually capture errors and other events.

Sentry's SDK hooks into your runtime environment and automatically reports errors, uncaught exceptions, and unhandled rejections as well as other types of errors depending on the platform.

> Key terms:
>
> - An *event* is one instance of sending data to Sentry. Generally, this data is an error or exception.
> - An *issue* is a grouping of similar events.
> - The reporting of an event is called *capturing*. When an event is captured, it's sent to Sentry.

The most common form of capturing is to capture errors. What can be captured as an error varies by platform. In general, if you have something that looks like an exception, it can be captured. For some SDKs, you can also omit the argument to `captureException` and Sentry will attempt to capture the current exception. It is also useful for manual reporting of errors or messages to Sentry.

While capturing an event, you can also record the breadcrumbs that lead up to that event. Breadcrumbs are different from events: they will not create an event in Sentry, but will be buffered until the next event is sent. Learn more about breadcrumbs in our Breadcrumbs documentation.

Docs

to *capture* uncaught exceptions and unhandled promise rejections, as described in the official ECMAScript 6 standard. You can disable this default behavior by changing the `onunhandledrejection` option to `false` in your GlobalHandlers integration and manually hook into each event handler, then call `Sentry.captureException` or `Sentry.captureMessage` directly.

You can pass an `Error` object to `captureException()` to get it captured as an event. It's also possible to pass non-`Error` objects and strings, but be aware that the resulting events in Sentry may be missing a stack trace.

JavaScript

```
import * as Sentry from "@sentry/react";

try {
  aFunctionThatMightFail();
} catch (err) {
  Sentry.captureException(err);
}
```

Sentry calls like `captureException` or `captureMessage` are side effects, so they should be wrapped in a `useEffect` hook to avoid triggering them on every render.

JavaScript

```
import * as Sentry from "@sentry/react";
import { useEffect } from "react";

function App() {
  const [info, error] = useQuery("/api/info");
  useEffect(() => {
    if (error) {
      Sentry.captureException(error);
    }
  }, [error]);

  // ...
}
```

Docs

information that should be sent to Sentry. Typically, our SDKs don't automatically capture messages, but you can capture them manually.

Messages show up as issues on your issue stream, with the message as the issue name.

JavaScript

```
Sentry.captureMessage("Something went wrong");
```

Previous
<< Source Maps

Next
React Features >>

Help improve this content

Our documentation is open source and available on GitHub. Your contributions are welcome, whether fixing a typo (drat!) or suggesting an update ("yeah, this would be better").

How to contribute | Edit this page | Create a docs issue | Get support