Default Integrations

Learn about default integrations, what they do, and how they hook into the standard library or the interpreter itself.

System integrations are integrations enabled by default that integrate into the standard library or the interpreter itself. Sentry documents them so you can see what they do and that they can be disabled if they cause issues. To disable all system integrations, set `default_integrations=False` when calling `init()`.

Atexit

Import name: `sentry_sdk.integrations.atexit.AtexitIntegration`

This integrates with the interpreter's `atexit` system to automatically flush events from the background queue on interpreter shutdown. Typically, one does not need to disable this. Even if the functionality is not wanted, it's easier to disable it by setting the `shutdown_timeout` to `0` in the options passed to `init()`.

Excepthook

Import name: `sentry_sdk.integrations.excepthook.ExcepthookIntegration`

This integration registers with the interpreter's except hook system. Through this, any exception that is unhandled will be reported to Sentry automatically. Note that exceptions raised in interactive interpreter sessions will not be reported.

Skip to content

You can pass the following keyword arguments to `ExcepthookIntegration()`:

```
$ python
>>> import sentry_sdk
>>> from sentry_sdk.integrations.excepthook import ExcepthookIntegration
>>> sentry_sdk.init(..., integrations=[ExcepthookIntegration(always_run=
>>> raise Exception("I will become an error")
```

By default, the SDK does not capture errors occurring in the REPL (`always_run=False`).

Deduplication

Import name: `sentry_sdk.integrations.dedupe.DedupeIntegration`

This integration deduplicates certain events. The Sentry Python SDK enables it by default, and it should not be disabled except in rare circumstances. Disabling this integration, for instance, will cause duplicate error logging in the Flask framework.

Stdlib

Import name: `sentry_sdk.integrations.stdlib.StdlibIntegration`

The stdlib integration instruments certain modules in the standard library to emit breadcrumbs. The Sentry Python SDK enables this by default, and it rarely makes sense to disable.

- Any outgoing HTTP request done with `httplib` will result in a breadcrumb being logged. `urllib3` and `requests` use `httplib` under the hood, so HTTP requests from those packages should be covered as well.
- Subprocesses spawned with the `subprocess` module will result in a breadcrumb being logged.

Modules

Skip to content

Import name: `sentry_sdk.integrations.modules.ModulesIntegration`

Argv

Import name: `sentry_sdk.integrations.argv.ArgvIntegration`

Adds `sys.argv` as an `extra` attribute to each event.

Logging

Import name: `sentry_sdk.integrations.logging.LoggingIntegration`

See *Logging*

Threading

Import name: `sentry_sdk.integrations.threading.ThreadingIntegration`

Reports crashing threads.

This integration also accepts an option `propagate_scope`, which influences the way data is transferred between threads. If set to `True` (default), the current scope will be shared between threads and scope data (such as tags) will be transferred from the parent thread to the child thread.

Context data is automatically propagated in `concurrent.futures` thread pools beginning with version 2.41.0, when `propagate_scope` is `True`.

Next are two code samples that demonstrate what boilerplate you would have to write without `propagate_scope`. This boilerplate is necessary if you want to propagate context data into a thread pool before version 2.41.0, for example.

Manual propagation

Python

```
import threading

import sentry_sdk
```

Docs

```
sentry_sdk.set_tag("mydata", 42)

def run(thread_scope):
    thread_scope.capture_message("hi")  # event will have `mydata` tag atta

# We take all context data (the tags map and even the entire client
# configuration), and pass it as explicit variable
# into the thread.
with isolation_scope() as thread_scope:
    tr = threading.Thread(target=run, args=[thread_scope])
    tr.start()
    tr.join()
```

Automatic propagation

Python

Docs

```
import sentry_sdk
from sentry_sdk.integrations.threading import ThreadingIntegration

sentry_sdk.init(
    ...,
    integrations=[
        ThreadingIntegration(propagate_scope=True),
    ],
)

sentry_sdk.set_tag("mydata", 42)

def run():
    sentry_sdk.capture_message("hi")  # event will have `mydata` tag attach

# The threading integration hooks into the stdlib to automatically pass
# existing context data when a `Thread` is instantiated.
tr = threading.Thread(target=run)
tr.start()
tr.join()
```

Previous
<< WSGI

Next
Data Management >>

Was this helpful? Yes 👍 No 👎

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