# **Events**

The Tauri event system is a multi-producer multi-consumer communication primitive that allows message passing between the frontend and the backend. It is analogous to the command system, but a payload type check must be written on the event handler and it simplifies communication from the backend to the frontend, working like a channel.

A Tauri application can listen and emit global and window-specific events. Usage from the frontend and the backend is described below.

## **Frontend**

The event system is accessible on the frontend on the event and window modules of the @tauri-apps/api package.

### **Global events**

To use the global event channel, import the event module and use the emit and listen functions:

```
import { emit, listen } from '@tauri-apps/api/event'

// listen to the `click` event and get a function to remove the event listener

// there's also a `once` function that subscribes to an event and automatically unsubscribes
the listener on the first event

const unlisten = await listen('click', (event) => {
    // event.event is the event name (useful if you want to use a single callback fn for multiple
event types)
    // event.payload is the payload object
})

// emits the `click` event with the object payload
emit('click', {
    theMessage: 'Tauri is awesome!',
})
```

## Window-specific events

Window-specific events are exposed on the window module.

```
import { appWindow, WebviewWindow } from '@tauri-apps/api/window'

// emit an event that is only visible to the current window
appWindow.emit('event', { message: 'Tauri is awesome!' })

// create a new webview window and emit an event only to that window
const webview = new WebviewWindow('window')
webview.emit('event')
```

# **Backend**

On the backend, the global event channel is exposed on the App struct, and window-specific events can be emitted using the Window trait.

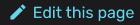
#### **Global events**

```
use tauri::Manager;
// the payload type must implement `Serialize` and `Clone`.
#[derive(Clone, serde::Serialize)]
struct Payload {
  message: String,
fn main() {
  tauri::Builder::default()
    .setup(|app| {
      // listen to the `event-name` (emitted on any window)
      let id = app.listen global("event-name", |event| {
        println!("got event-name with payload {:?}", event.payload());
      });
      // unlisten to the event using the `id` returned on the `listen_global` function
      // a `once_global` API is also exposed on the `App` struct
      app.unlisten(id);
      // emit the `event-name` event to all webview windows on the frontend
      app.emit all("event-name", Payload { message: "Tauri is awesome!".into() }).unwrap();
      Ok(())
    })
    .run(tauri::generate context!())
    .expect("failed to run app");
```

# Window-specific events

To use the window-specific event channel, a Window object can be obtained on a command handler or with the get window function:

```
use tauri::{Manager, Window};
// the payload type must implement `Serialize` and `Clone`.
#[derive(Clone, serde::Serialize)]
struct Payload {
  message: String,
// init a background process on the command, and emit periodic events only to the window that
used the command
#[tauri::command]
fn init_process(window: Window) {
  std::thread::spawn(move | | {
    loop {
      window.emit("event-name", Payload { message: "Tauri is awesome!".into() }).unwrap();
  });
fn main() {
  tauri::Builder::default()
    .setup(|app| {
      // `main` here is the window label; it is defined on the window creation or under
`tauri.conf.json`
      let main_window = app.get_window("main").unwrap();
      // listen to the `event-name` (emitted on the `main` window)
      let id = main_window.listen("event-name", |event| {
        println!("got window event-name with payload {:?}", event.payload());
      });
      // unlisten to the event using the `id` returned on the `listen` function
      // an `once` API is also exposed on the `Window` struct
      main window.unlisten(id);
      // emit the `event-name` event to the `main` window
      main window.emit("event-name", Payload { message: "Tauri is awesome!".into() }).unwrap();
      Ok(())
    })
    .invoke handler(tauri::generate handler![init process])
    .run(tauri::generate_context!())
    .expect("failed to run app");
```



Last updated on **Mar 25, 2023**