

[Guides](#)[Features](#)[Multiwindow](#)

Multiwindow

Manage multiple windows on a single application.

Creating a window

A window can be created statically from the Tauri configuration file or at runtime.

Static window

Multiple windows can be created with the [tauri.windows](#) configuration array. The following JSON snippet demonstrates how to statically create several windows through the config:

```
{
  "tauri": {
    "windows": [
      {
        "label": "external",
        "title": "Tauri Docs",
        "url": "https://tauri.app"
      },
      {
        "label": "local",
        "title": "Tauri",
        "url": "index.html"
      }
    ]
  }
}
```

Note that the window label must be unique and can be used at runtime to access the window instance. The complete list of configuration options available for static windows can be found in the [WindowConfig](#) documentation.

Runtime window

You can also create windows at runtime either via the Rust layer or through the Tauri API.

Create a window in Rust

A window can be created at runtime using the [WindowBuilder](#) struct.

To create a window, you must have an instance of the running [App](#) or an [AppHandle](#).

Create a window using the [App](#) instance

The [App](#) instance can be obtained in the setup hook or after a call to [Builder::build](#).

```
tauri::Builder::default()
    .setup(|app| {
        let docs_window = tauri::WindowBuilder::new(
            app,
            "external", /* the unique window label */
            tauri::WindowUrl::External("https://tauri.app/".parse().unwrap())
        ).build()?;
        let local_window = tauri::WindowBuilder::new(
            app,
            "local",
            tauri::WindowUrl::App("index.html".into())
        ).build()?;
        Ok(())
    })
    .run(tauri::generate_context!())
    .expect("error while running app");
```

Using the setup hook ensures static windows and Tauri plugins are initialized. Alternatively, you can create a window after building the [App](#):

```
let app = tauri::Builder::default()
    .build(tauri::generate_context!())
    .expect("error while building tauri application");

let docs_window = tauri::WindowBuilder::new(
    &app,
    "external", /* the unique window label */
    tauri::WindowUrl::External("https://tauri.app/".parse().unwrap())
).build().expect("failed to build window");

let local_window = tauri::WindowBuilder::new(
    &app,
    "local",
    tauri::WindowUrl::App("index.html".into())
).build()?;
```

```
// This will start the app and no other code below this will run.  
app.run(|_, _| {});
```

This method is useful when you cannot move ownership of values to the setup closure.

Create a window using an `AppHandle` instance

An `AppHandle` instance can be obtained using the `[App::handle]` function or directly injected in Tauri commands.

```
tauri::Builder::default()  
  .setup(|app| {  
    let handle = app.handle();  
    std::thread::spawn(move || {  
      let local_window = tauri::WindowBuilder::new(  
        &handle,  
        "local",  
        tauri::WindowUrl::App("index.html".into())  
      ).build()?;  
    });  
    Ok(())  
  })
```

```
#[tauri::command]  
async fn open_docs(handle: tauri::AppHandle) {  
  let docs_window = tauri::WindowBuilder::new(  
    &handle,  
    "external", /* the unique window label */  
    tauri::WindowUrl::External("https://tauri.app/".parse().unwrap())  
  ).build().unwrap();  
}
```

! INFO

When creating windows in a Tauri command, ensure the command function is `async` to avoid a deadlock on Windows due to the [wry#583](#) issue.

Create a window in JavaScript

Using the Tauri API you can easily create a window at runtime by importing the `WebviewWindow` class.

```
import { WebviewWindow } from '@tauri-apps/api/window'  
const webview = new WebviewWindow('theUniqueLabel', {
```

```
url: 'path/to/page.html',
}))
// since the webview window is created asynchronously,
// Tauri emits the `tauri://created` and `tauri://error` to notify you of the creation response
webview.once('tauri://created', function () {
  // webview window successfully created
}))
webview.once('tauri://error', function (e) {
  // an error occurred during webview window creation
}))
```

Creating additional HTML pages

If you want to create additional pages beyond `index.html`, you will need to make sure they are present in the `dist` directory at build time. How you do this depends on your frontend setup. If you use Vite, create an additional `input` for the second HTML page in `vite.config.js`.

Accessing a window at runtime

The window instance can be queried using its label and the `get_window` method on Rust or `WebviewWindow.getByLabel` on JavaScript.

```
use tauri::Manager;
tauri::Builder::default()
  .setup(|app| {
    let main_window = app.get_window("main").unwrap();
    Ok(())
  })
```

Note that you must import `tauri::Manager` to use the `get_window` method on `App` or `AppHandle` instances.

```
import { WebviewWindow } from '@tauri-apps/api/window'
const mainWindow = WebviewWindow.getByLabel('main')
```

Communicating with other windows

Window communication can be done using the event system. See the [Event Guide](#) for more information.

