Configuration Files

Since Tauri is a toolkit for building applications there can be many files to configure project settings. Some common files that you may run across are tauri.conf.json, package.json and Cargo.toml. We briefly explain each on this page to help point you in the right direction for which file to modify.

Tauri Config #

The file can be either tauri.conf.json, tauri.conf.json5, or Tauri.tom1. The default is tauri.conf.json. See the note below for more information.

This is the file used by the Tauri process. You can define build settings (such as the command run before tauri build or tauri dev), set the name and version of your app, control the Tauri process, and configure any plugin settings. You can find all of the options in the tauri.conf.json API reference.

(i) NOTE

The default Tauri config format is .json. The .json5 or .tom1 format can be enabled by adding the config-json5 or config-tom1 feature flag (respectively) to the tauri and tauri-build dependencies in Cargo.tom1. Note that the .tom1 format is only available from Tauri 1.1 and above.

```
Cargo.toml
```

```
[build-dependencies]
tauri-build = { version = "1.0.0", features = [ "config-json5" ] }

[dependencies]
serde_json = "1.0"
serde = { version = "1.0", features = ["derive"] }
tauri = { version = "1.0.0", features = [ "api-all", "config-json5" ] }
```

The structure and values are the same across all formats, however, the formatting should be consistent with the respective file's format.

Cargo's manifest file is used to declare Rust crates your app depends on, metadata about your app, and other Rust-related features. If you do not intend to do backend development using Rust for your app then you may not be modifying it much, but it's important to know that it exists and what it does.

Below is an example of a barebones Cargo.toml file for a Tauri project:

```
Cargo.toml
[package]
name = "app"
version = "0.1.0"
description = "A Tauri App"
authors = ["you"]
license = ""
repository = ""
default-run = "app"
edition = "2021"
rust-version = "1.57"
[build-dependencies]
tauri-build = { version = "1.0.0" }
[dependencies]
serde_json = "1.0"
serde = { version = "1.0", features = ["derive"] }
tauri = { version = "1.0.0", features = [ "api-all" ] }
[features]
# by default Tauri runs in production mode
# when `tauri dev` runs it is executed with `cargo run --no-default-features` if `devPath` is
an URL
default = [ "custom-protocol" ]
# this feature is used for production builds where `devPath` points to the filesystem
# DO NOT remove this
custom-protocol = [ "tauri/custom-protocol" ]
```

The most important parts to take note of are the tauri-build and tauri dependencies. Generally, they must both be on the latest minor versions as the Tauri CLI, but this is not strictly required. If you encounter issues while trying to run your app you should check that any Tauri versions (tauri and tauri-cli) are on the latest versions for their respective minor releases.

Cargo version numbers use Semantic Versioning. Running cargo update will pull the latest available Semver-compatible versions of all dependencies. For example, if you specify 1.0.0 as the version for tauri-build, Cargo will detect and download version 1.0.4 because it is the latest Semver-compatible version available. Tauri will update the major version number whenever a breaking change is introduced,

meaning you should always be capable of safely upgrading to the latest minor and patch versions without fear of your code breaking.

If you want to use a specific crate version you can use exact versions instead by prepending = to the version number of the dependency:

```
tauri-build = { version = "=1.0.0" }
```

An additional thing to take note of is the features=[] portion of the tauri dependency. Running tauri dev and tauri build will automatically manage which features need to be enabled in your project based on the "allowlist" properties you set in tauri.conf.json.

When you build your application a Cargo.lock file is produced. This file is used primarily for ensuring that the same dependencies are used across machines during development (similar to yarn.lock or packagelock.json in Node.js). Since you are developing a Tauri app, this file should be committed to your source repository (only Rust libraries should omit committing this file).

To learn more about Cargo.toml you can read more in the official documentation.

package.json

This is the package file used by Node.js. If the frontend of a Tauri app is developed using Node.js-based technologies (such as npm, yarn, or pnpm) this file is used to configure the frontend dependencies and scripts.

An example of a barebones package.json file for a Tauri project might look a little something like this:

```
package.json

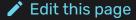
{
    "scripts": {
        "dev": "command-for-your-framework",
        "tauri": "tauri"
    },
    "dependencies": {
        "@tauri-apps/api": "^1.0",
        "@tauri-apps/cli": "^1.0"
    }
}
```

It's common to use the "scripts" section to store the command used to launch the frontend used by your Tauri application. The above file specifies the dev command that you can run using yarn dev or npm run dev to start the frontend framework.

The dependencies object specifies which dependencies Node.js should download when you run either yarn or npm install (in this case the Tauri CLI and API).

In addition to the package.json file you may see either a yarn.lock file or a package-lock.json file. These files assist in ensuring that when you download the dependencies later you'll get the exact same versions that you have used during development (similar to Cargo.lock in Rust).

To learn more about package.json you can read more in the official documentation.



Last updated on Nov 11, 2022