Tauri Architecture

Introduction

Tauri is a polyglot and generic toolkit that is very composable and allows engineers to make a wide variety of applications. It is used for building applications for desktop computers using a combination of Rust tools and HTML rendered in a Webview. Apps built with Tauri can ship with any number of pieces of an optional JS API and Rust API so that webviews can control the system via message passing. Developers can extend the default API with their own functionality and bridge the Webview and Rust-based backend easily.

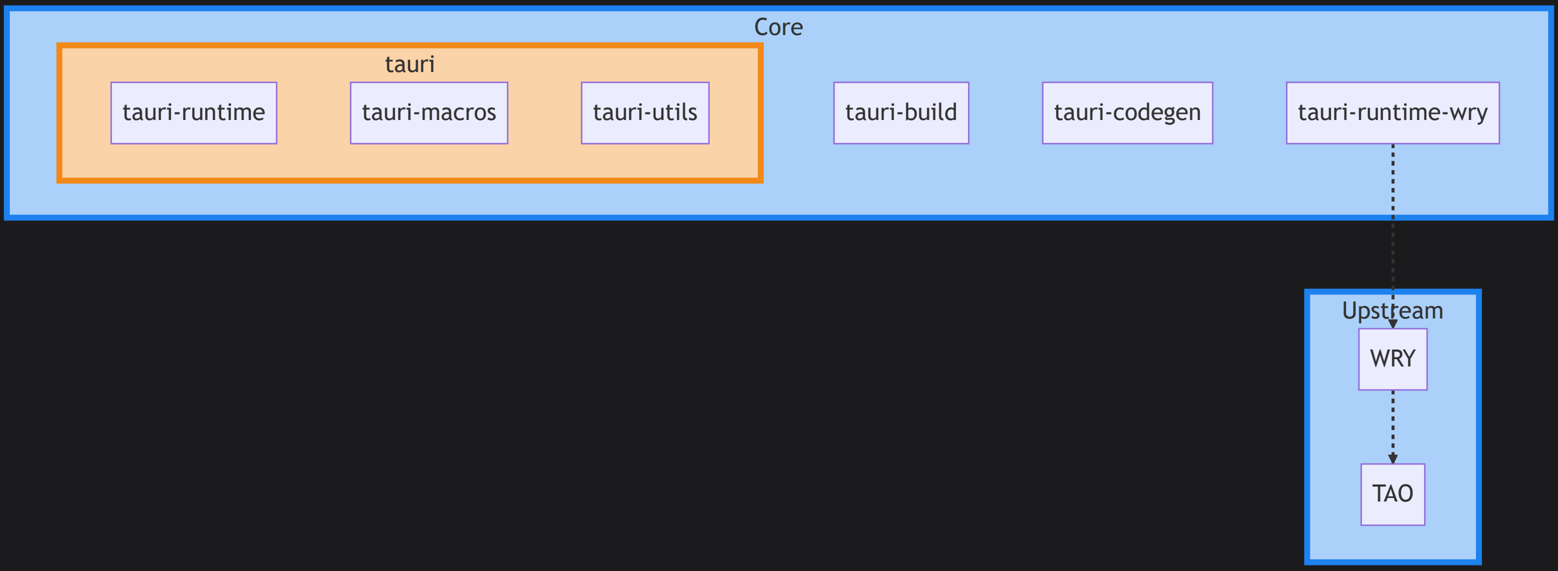
Tauri apps can have custom menus and tray-type interfaces. They can be updated and are managed by the user's operating system as expected. They are very small because they use the OS's webview. They do not ship a runtime since the final binary is compiled from Rust. This makes the reversing of Tauri apps not a trivial task.

What Tauri is Not

Tauri is not a lightweight kernel wrapper. Instead, it directly uses WRY and TAO to do the heavy lifting in making system calls to the OS.

Tauri is not a VM or virtualized environment. Instead, it is an application toolkit that allows making Webview OS applications.

Core Ecosystem



tauri

This is the major crate that holds everything together. It brings the runtimes, macros, utilities and API into one final product. It reads the `tauri.conf.json` file at compile time to bring in features and undertake the actual configuration of the app (and even the `Cargo.toml` file in the project's folder). It handles script injection (for polyfills / prototype revision) at runtime, hosts the API for systems interaction, and even manages the updating process.

tauri-runtime

The glue layer between Tauri itself and lower-level webview libraries.

tauri-macros

Creates macros for the context, handler, and commands by leveraging the `tauri-codegen` crate.

tauri-utils

Common code that is reused in many places and offers useful utilities like parsing configuration files, detecting platform triples, injecting the CSP, and managing assets.

tauri-build

Applies the macros at build-time to rig some special features needed by `cargo`.

tauri-codegen

Embeds, hashes, and compresses assets, including icons for the app as well as the system tray. Parses `tauri.conf.json` at compile time and generates the Config struct.

tauri-runtime-wry

This crate opens up direct systems-level interactions specifically for WRY, such as printing, monitor detection, and other windowing-related tasks.

Tauri Tooling

API (JavaScript / TypeScript)

A typescript library that creates `cjs` and `esm` JavaScript endpoints for you to import into your frontend framework so that the Webview can call and listen to backend activity. Also ships in pure typescript, because for some frameworks this is more optimal. It uses the message passing of webviews to their hosts.

Bundler (Rust / Shell)

A library that builds a Tauri app for the platform it detects or is told. Currently supports macOS, Windows and Linux - but in the near future will support mobile platforms as well. May be used outside of Tauri projects.

cli.rs (Rust)

This Rust executable provides the full interface to all of the required activities for which the CLI is required. It runs on macOS, Windows, and Linux.

cli.js (JavaScript)

Wrapper around `cli.rs` using `napi-rs` to produce npm packages for each platform.

create-tauri-app (JavaScript)

A toolkit that will enable engineering teams to rapidly scaffold out a new `tauri-apps` project using the frontend framework of their choice (as long as it has been configured).

Upstream Crates

The Tauri-Apps organisation maintains two "upstream" crates from Tauri, namely TAO for creating and managing application windows, and WRY for interfacing with the Webview that lives within the window.

TAO

Cross-platform application window creation library in Rust that supports all major platforms like Windows, macOS, Linux, iOS and Android. Written in Rust, it is a fork of winit that we have extended for our own needs - like menu bar and system tray.

WRY

WRY is a cross-platform WebView rendering library in Rust that supports all major desktop platforms like Windows, macOS, and Linux. Tauri uses WRY as the abstract layer responsible to determine which webview is used (and how interactions are made).

Additional Tooling

tauri-action

GitHub workflow that builds Tauri binaries for all platforms. Even allows creating a (very basic) Tauri app even if Tauri is not set up.

tauri-vscode

This project enhances the Visual Studio Code interface with several nice-to-have features.

vue-cli-plugin-tauri

Allows you to very quickly install Tauri in a vue-cli project.

Plugins

Tauri Plugin Guide

Generally speaking, plugins are authored by third parties (even though there may be official, supported plugins). A plugin generally does 3 things:

1. Enables Rust code to do "something".
2. Provides interface glue to make it easy to integrate into an app.
3. Provides a JavaScript API for interfacing with the Rust code.

Here are some examples of Tauri Plugins:

- tauri-plugin-sql
- tauri-plugin-stronghold
- tauri-plugin-authenticator

License

Tauri itself is licensed under MIT or Apache-2.0. If you repackage it and modify any source code, it is your responsibility to verify that you are complying with all upstream licenses. Tauri is provided AS-IS with no explicit claim for suitability for any purpose.

Here you may peruse our Software Bill of Materials.

Next Steps

Your First Tauri App

Development Cycle

Publishing

Updating

Edit this page

Last updated on ***Nov 25, 2022***