Embedding External Binaries

You may need to embed depending binaries to make your application work or prevent users from installing additional dependencies (e.g., Node.js or Python). We call this binary a `sidecar`.

To bundle the binaries of your choice, you can add the `externalBin` property to the `tauri > bundle` object in your `tauri.conf.json`.

See more about tauri.conf.json configuration here.

`externalBin` expects a list of strings targeting binaries either with absolute or relative paths.

Here is a sample to illustrate the configuration. This is not a complete `tauri.conf.json` file:

```
{
  "tauri": {
    "bundle": {
      "externalBin": [
        "/absolute/path/to/sidecar",
        "relative/path/to/binary",
        "binaries/my-sidecar"
      ]
    },
    "allowlist": {
      "shell": {
        "sidecar": true,
        "scope": [
          { "name": "/absolute/path/to/sidecar", "sidecar": true },
          { "name": "relative/path/to/binary", "sidecar": true },
          { "name": "binaries/my-sidecar", "sidecar": true }
        ]
      }
    }
  }
}
```

A binary with the same name and a `-$TARGET_TRIPLE` suffix must exist on the specified path. For instance, `"externalBin": ["binaries/my-sidecar"]` requires a `src-tauri/binaries/my-sidecar-x86_64-unknown-linux-gnu` executable on Linux. You can find the **current** platform's target triple by looking at the `host:` property reported by the `rustc -Vv` command.

If the `grep` and `cut` commands are available, as they should on most Unix systems, you can extract the target triple directly with the following command:

```
$ rustc -Vv | grep host | cut -f2 -d' '
```

On Windows you can use PowerShell instead:

```
PS C:\> rustc -Vv | Select-String "host:" | ForEach-Object {$_.Line.split(" ")[1]}
```

Here's a Node.js script to append the target triple to a binary:

```
const execa = require('execa')
const fs = require('fs')

let extension = ''
if (process.platform === 'win32') {
  extension = '.exe'
}

async function main() {
  const rustInfo = (await execa('rustc', ['-vV'])).stdout
  const targetTriple = /host: (\S+)/g.exec(rustInfo)[1]
  if (!targetTriple) {
    console.error('Failed to determine platform target triple')
  }
  fs.renameSync(
    `src-tauri/binaries/sidecar${extension}`,
    `src-tauri/binaries/sidecar-${targetTriple}${extension}`
  )
}

main().catch((e) => {
  throw e
})
```

Running it from JavaScript

In the JavaScript code, import the `Command` class on the `shell` module and use the `sidecar` static method.

Note that you must configure the allowlist to enable `shell > sidecar` and configure all binaries in `shell > scope`.

```
import { Command } from '@tauri-apps/api/shell'
// alternatively, use `window.__TAURI__.shell.Command`
// `binaries/my-sidecar` is the EXACT value specified on `tauri.conf.json > tauri > bundle > externalBin`
const command = Command.sidecar('binaries/my-sidecar')
const output = await command.execute()
```

Running it from Rust

On the Rust side, import the `Command` struct from the `tauri::api::process` module:

```
// `new_sidecar()` expects just the filename, NOT the whole path like in JavaScript
let (mut rx, mut child) = Command::new_sidecar("my-sidecar")
  .expect("failed to create `my-sidecar` binary command")
  .spawn()
  .expect("Failed to spawn sidecar");

tauri::async_runtime::spawn(async move {
  // read events such as stdout
  while let Some(event) = rx.recv().await {
    if let CommandEvent::Stdout(line) = event {
      window
        .emit("message", Some(format!("'{}'", line)))
        .expect("failed to emit event");
      // write to stdin
      child.write("message from Rust\n".as_bytes()).unwrap();
    }
  }
});
```

Note that you must enable the **process-command-api** Cargo feature (Tauri's CLI will do this for you once you changed the config):

```
# Cargo.toml
[dependencies]
tauri = { version = "1", features = ["process-command-api", ...] }
```

Passing arguments

You can pass arguments to Sidecar commands just like you would for running normal `Command`s (see Restricting access to the Command APIs).

First, define the arguments that need to be passed to the Sidecar command in `tauri.conf.json`:

```
{
  "tauri": {
    "bundle": {
      "externalBin": [
        "/absolute/path/to/sidecar",
        "relative/path/to/binary",
        "binaries/my-sidecar"
      ]
    },
    "allowlist": {
      "shell": {
        "sidecar": true,
        "scope": [
          {
            "name": "binaries/my-sidecar",
            "sidecar": true,
            "args": [
              "arg1",
              "-a",
              "--arg2",
              {
                "validator": "\\S+"
              }
            ]
          }
        ]
      }
    }
  }
}
```

Then, to call the sidecar command, simply pass in **all** the arguments as an array:

```
import { Command } from '@tauri-apps/api/shell'
// alternatively, use `window.__TAURI__.shell.Command`
// `binaries/my-sidecar` is the EXACT value specified on `tauri.conf.json > tauri > bundle >
externalBin`
// notice that the args array matches EXACTLY what is specified on `tauri.conf.json`.
const command = Command.sidecar('binaries/my-sidecar', [
  'arg1',
  '-a',
  '--arg2',
  'any-string-that-matches-the-validator',
])
const output = await command.execute()
```

Using Node.js on a Sidecar

The Tauri sidecar example demonstrates how to use the sidecar API to run a Node.js application on Tauri. It compiles the Node.js code using pkg and uses the scripts above to run it.

Edit this page

*Last updated on **Jun 30, 2023***