Making Your Own CLI

Tauri enables your app to have a CLI through clap, a robust command line argument parser. With a simple CLI definition in your `tauri.conf.json` file, you can define your interface and read its argument matches map on JavaScript and/or Rust.

Base Configuration

Under `tauri.conf.json`, you have the following structure to configure the interface:

src-tauri/tauri.conf.json

```
{
  "tauri": {
    "cli": {
      "description": "", // command description that's shown on help
      "longDescription": "", // command long description that's shown on help
      "beforeHelp": "", // content to show before the help text
      "afterHelp": "", // content to show after the help text
      "args": [], // list of arguments of the command, we'll explain it later
      "subcommands": {
        "subcommand-name": {
          // configures a subcommand that is accessible
          // with `./app subcommand-name --arg1 --arg2 --etc`
          // configuration as above, with "description", "args", etc.
        }
      }
    }
  }
}
```

> **NOTE**
>
> All JSON configurations here are just samples, many other fields have been omitted for the sake of clarity.

Adding Arguments

The `args` array represents the list of arguments accepted by its command or subcommand. You can find more details about the way to configure them here.

Positional Arguments

A positional argument is identified by its position in the list of arguments. With the following configuration:

```
{
  "args": [
    {
      "name": "source",
      "index": 1,
      "takesValue": true
    },
    {
      "name": "destination",
      "index": 2,
      "takesValue": true
    }
  ]
}
```

Users can run your app as `./app tauri.txt dest.txt` and the arg matches map will define `source` as `"tauri.txt"` and `destination` as `"dest.txt"`.

Named Arguments

A named argument is a (key, value) pair where the key identifies the value. With the following configuration:

```
{
  "args": [
    {
      "name": "type",
      "short": "t",
      "takesValue": true,
      "multiple": true,
      "possibleValues": ["foo", "bar"]
    }
  ]
}
```

Users can run your app as `./app --type foo bar`, `./app -t foo -t bar` or `./app --type=foo,bar` and the arg matches map will define `type` as `["foo", "bar"]`.

Flag Arguments

A flag argument is a standalone key whose presence or absence provides information to your application. With the following configuration:

```
{
  "args": [
    {
      "name": "verbose",
      "short": "v",
      "multipleOccurrences": true
    }
  ]
}
```

Users can run your app as `./app -v -v -v`, `./app --verbose --verbose --verbose` or `./app -vvv` and the arg matches map will define `verbose` as `true`, with `occurrences = 3`.

Subcommands

Some CLI applications have additional interfaces as subcommands. For instance, the `git` CLI has `git branch`, `git commit` and `git push`. You can define additional nested interfaces with the `subcommands` array:

```
{
  "cli": {
    ...
    "subcommands": {
      "branch": {
        "args": []
      },
      "push": {
        "args": []
      }
    }
  }
}
```

Its configuration is the same as the root application configuration, with the `description`, `longDescription`, `args`, etc.

Reading the matches

Rust

```rust
fn main() {
  tauri::Builder::default()
    .setup(|app| {
      match app.get_cli_matches() {
        // `matches` here is a Struct with { args, subcommand }.
        // `args` is `HashMap<String, ArgData>` where `ArgData` is a struct with { value, occurrences }.
        // `subcommand` is `Option<Box<SubcommandMatches>>` where `SubcommandMatches` is a struct with { name, matches }.
        Ok(matches) => {
          println!("{:?}", matches)
        }
        Err(_) => {}
      }
      Ok(())
    })
    .run(tauri::generate_context!())
    .expect("error while running tauri application");
}
```

JavaScript

```javascript
import { getMatches } from '@tauri-apps/api/cli'

getMatches().then((matches) => {
  // do something with the { args, subcommand } matches
})
```

Complete documentation

You can find more about the CLI configuration here.

Edit this page

*Last updated on **May 20, 2023***