Chapter 29. Configuration directories

The Today program needs a place to put all the data files used by various event providers. Typically this place is a directory somwehere in the home directory of the currently logged-in user, so that the data and configuration can be customized by each user.

The path to the configuration directory is going to be different from one operating system to another. For those operating systems that we support directly, the configuration directories for the Today program would be as follows:

- Linux: either the traditional `$HOME/.today` directory, or the `.config/today` directory, as per the XDG Base Directory Specification.
- macOS: `$HOME/Application Support/Today`, a Mac-specific folder
- Microsoft Windows: `%USERPROFILE%\AppData\Roaming\Today`, where `USERPROFILE` is an environment variable that points to the user's home directory

For Linux and macOS, the expression `$HOME` in a shell gets the value of the `HOME` environment variable that points to the user's home directory.

Correspondingly, for Windows, the expression `%USERPROFILE%` in Command Prompt gets the value of the `USERPROFILE` environment variable that points to the user's home directory.

Rust has a configuration conditional attribute called `cfg` that can be used to compile different code for different operating systems, so we could use that to write different code for those operating systems. However, we don't need to manually handle all these cases, since there is a crate for that, called `dirs` .

Getting the config directory name

Let's set up a quick test to see the values of the various directories. In a suitable directory, make a new package with `cargo new configdirs`. Add the `dirs` crate to the package (either manually or with `cargo add dirs`), so that your `Cargo.toml` will mention the `dirs` crate:

```
[dependencies]
```

```
dirs = "6.0.0"
```

Then edit the `src/main.rs` to contain the following:

Example 29.1. Helper program to print out the configuration directory

```
use dirs;

fn main() {
    match dirs::config_dir() {
        Some(dir) => println!("Config directory: '{}'", dir.display()),
        None => println!("No config directory found!")
    }
}
```

If you now run the program with `cargo run`, it should show your configuration directory. In macOS Tahoe (26.3) it prints out

```
Config directory: '/Users/me/Library/Application Support'
```

In Linux (Ubuntu 22.04) the value prints out as

```
Config directory: '/home/me/.config'
```

In Microsoft Windows 11, the result is

```
Config directory: 'C:\Users\me\AppData\Roaming'
```

These results match the ones reported in the documentation of the `dirs` crate.

It is interesting to compare the messages that appear when compiling the program for different operating systems. In Windows, compiling and running the program with Cargo shows this:

```
cargo run
   Compiling windows-link v0.2.1
   Compiling option-ext v0.2.0
   Compiling windows-sys v0.61.2
   Compiling dirs-sys v0.5.0
   Compiling dirs v6.0.0
   Compiling configdirs v0.1.0 (C:\Users\me\Projects\Rust\configdirs)
```

```
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 3.72s
     Running `target\debug\configdirs.exe`
Config directory: 'C:\Users\me\AppData\Roaming'
```

However, in Ubuntu 22.04 the list of crates is slightly different:

```
   Compiling libc v0.2.182
   Compiling option-ext v0.2.0
   Compiling dirs-sys v0.5.0
   Compiling dirs v6.0.0
   Compiling configdirs v0.1.0 (/home/me/Projects/Rust/configdirs)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.62s
     Running `target/debug/configdirs`
Config directory: '/home/me/.config'
```

As you can see, the Windows version has some crates specific to that operating system. In macOS the list of crates is the same as in the Linux version, because both macOS and Linux are Unix-based operating systems, and the underlying mechanism works the same. However, the configuration directory paths are still different, and inside the `dirs` crate the `cfg` attribute is being used to differentiate.

Building the configuration path

The actual path of the configuration file should contain a directory appended to the configuration directory, for example `/home/me/.config/today` in Linux. Let's make a helper function that uses the `dirs` crate and the Rust standard library type `std::path::PathBuf` to build and return the path, or `None` on failure. The function takes a string parameter so that it can be used in other programs too. We also need to bring `std::path::PathBuf` and `std::fs` into scope:

Example 29.2. Helper function to get the configuration directory

```
use std::path::PathBuf;
use std::fs;
use dirs;

// Gets the configuration directory path for `app_name`.
// If the directory does not exist, tries to create it.
```

```
// Returns an optional `PathBuf` containing the directory path,
// or `None` if the directory can't be created.
fn get_config_path(app_name: &str) -> Option<PathBuf> {
    if let Some(config_dir) = dirs::config_dir() {
        let config_path = config_dir.join(app_name);
        if !config_path.exists() {
            if let Err(_) = fs::create_dir(&config_path) {
                eprintln!("Unable to create config directory for {}", app_name);
                return None;
            }
        }
        return Some(config_path);
    }
    None
}
```

Finally we'll edit the `main` function to use this new helper:

Example 29.3. Constructing the configuration directory

```
fn main() {
    const APP_NAME: &str = "today";
    let config_path = get_config_path(APP_NAME);
    match config_path {
        Some(path) => println!("Config path: '{}'", path.display()),
        None => println!("No config directory found!")
    }
}
```

On macOS, the program prints

```
Config path: '/Users/me/Library/Application Support/today'
```