Window Menu

Native application menus can be attached to a window.

Creating a menu

To create a native window menu, import the `Menu`, `Submenu`, `MenuItem` and `CustomMenuItem` types. The `MenuItem` enum contains a collection of platform-specific items (currently not implemented on Windows). The `CustomMenuItem` allows you to create your own menu items and add special functionality to them.

```
use tauri::{CustomMenuItem, Menu, MenuItem, Submenu};
```

Create a `Menu` instance:

```
// here `"quit".to_string()` defines the menu item id, and the second parameter is the menu
item label.
let quit = CustomMenuItem::new("quit".to_string(), "Quit");
let close = CustomMenuItem::new("close".to_string(), "Close");
let submenu = Submenu::new("File", Menu::new().add_item(quit).add_item(close));
let menu = Menu::new()
  .add_native_item(MenuItem::Copy)
  .add_item(CustomMenuItem::new("hide", "Hide"))
  .add_submenu(submenu);
```

Adding the menu to all windows

The defined menu can be set to all windows using the `menu` API on the `tauri::Builder` struct:

```
use tauri::{CustomMenuItem, Menu, MenuItem, Submenu};

fn main() {
  let menu = Menu::new(); // configure the menu
  tauri::Builder::default()
    .menu(menu)
    .run(tauri::generate_context!())
    .expect("error while running tauri application");
}
```

Adding the menu to a specific window

You can create a window and set the menu to be used. This allows for defining a specific menu set for each application window.

```
use tauri::{CustomMenuItem, Menu, MenuItem, Submenu};
use tauri::WindowBuilder;

fn main() {
  let menu = Menu::new(); // configure the menu
  tauri::Builder::default()
    .setup(|app| {
      WindowBuilder::new(
        app,
        "main-window".to_string(),
        tauri::WindowUrl::App("index.html".into()),
      )
      .menu(menu)
      .build()?;
      Ok(())
    })
    .run(tauri::generate_context!())
    .expect("error while running tauri application");
}
```

Listening to events on custom menu items

Each `CustomMenuItem` triggers an event when clicked. Use the `on_menu_event` API to handle them, either on the global `tauri::Builder` or on a specific window.

Listening to events on global menus

```
use tauri::{CustomMenuItem, Menu, MenuItem};

fn main() {
  let menu = Menu::new(); // configure the menu
  tauri::Builder::default()
    .menu(menu)
    .on_menu_event(|event| {
      match event.menu_item_id() {
        "quit" => {
          std::process::exit(0);
        }
        "close" => {
          event.window().close().unwrap();
        }
```

```
        _ => {}
      }
    })
    .run(tauri::generate_context!())
    .expect("error while running tauri application");
}
```

Listening to events on window menus

```
use tauri::{CustomMenuItem, Menu, MenuItem};
use tauri::{Manager, WindowBuilder};

fn main() {
  let menu = Menu::new(); // configure the menu
  tauri::Builder::default()
    .setup(|app| {
      let window = WindowBuilder::new(
        app,
        "main-window".to_string(),
        tauri::WindowUrl::App("index.html".into()),
      )
      .menu(menu)
      .build()?;
      let window_ = window.clone();
      window.on_menu_event(move |event| {
        match event.menu_item_id() {
          "quit" => {
            std::process::exit(0);
          }
          "close" => {
            window_.close().unwrap();
          }
          _ => {}
        }
      });
      Ok(())
    })
    .run(tauri::generate_context!())
    .expect("error while running tauri application");
}
```

Updating menu items

The `Window` struct has a `menu_handle` method, which allows updating menu items:

```rust
fn main() {
  let menu = Menu::new(); // configure the menu
  tauri::Builder::default()
    .menu(menu)
    .setup(|app| {
      let main_window = app.get_window("main").unwrap();
      let menu_handle = main_window.menu_handle();
      std::thread::spawn(move || {
        // you can also `set_selected`, `set_enabled` and `set_native_image` (macOS only).
        menu_handle.get_item("item_id").set_title("New title");
      });
      Ok(())
    })
}
```

Edit this page

*Last updated on **Mar 25, 2023***