Bundling Rust Application as macOS .app

This guide explains how to bundle your Rust egui application into a native macOS application (.app bundle).

Why Bundle as .app?

Without bundling:

- Running `cargo run` opens a terminal window
- Cannot double-click to launch from Finder
- No app icon in Dock
- Not installable like a normal Mac app

With .app bundle:

- ✅ Double-clickable from Finder
- ✅ Custom app icon
- ✅ No terminal window appears
- ✅ Can be installed to /Applications
- ✅ Looks and behaves like any other Mac app

Quick Start

Simply run the bundling script:

```
./bundle_macos.sh
```

This will create `Shell Script Manager.app` in your project directory.

Developer Testing

After bundling, macOS Gatekeeper may block the app with a "malware" warning. **For testing/development**, use this command to bypass the warning: Or manually run:

```
sudo xattr -rd com.apple.quarantine "Shell Script Manager.app"
```

Then double-click the app to launch it normally.

Alternative: Right-click the app → Select "Open" → Click "Open" in the dialog. macOS will remember your choice.

> **Note:** This is only needed for local development. For distribution to other users, you'll need proper code signing with an Apple Developer certificate.

How to Use the .app Bundle

Option 1: Run from Terminal

```
open "Shell Script Manager.app"
```

Option 2: Double-click in Finder

1. Navigate to your project folder in Finder
2. Double-click `Shell Script Manager.app`
3. The app launches without any terminal window!

Option 3: Install to Applications

1. Drag `Shell Script Manager.app` to `/Applications` folder
2. Find it in Launchpad or Applications folder
3. Launch like any other Mac app

What the Script Does

1. Build Release Binary

```
cargo build --release
```

- Compiles your Rust application in optimized release mode
- Creates binary at `target/release/shell_script_manager`

2. Create .app Bundle Structure

```
Shell Script Manager.app/
├── Contents/
    ├── MacOS/                 # Executable goes here
    │   └── shell_script_manager
    ├── Resources/             # Icons and resources
    │   ├── icon.icns
    │   └── icon.png
    └── Info.plist            # App metadata
```

3. Copy Executable

- Copies the compiled binary into `Contents/MacOS/`

4. Create App Icon (ICNS format)

Converts your PNG icons to macOS ICNS format with multiple sizes:

- 16×16, 32×32 (normal & retina)

- 128×128, 256×256 (normal & retina)
- 512×512, 1024×1024 (normal & retina)

This ensures the icon looks sharp at all sizes (Dock, Finder, etc.)

5. Create Info.plist

The metadata file that tells macOS about your app:

```
<key>CFBundleExecutable</key>
<string>shell_script_manager</string>      <!-- Binary to run -->

<key>CFBundleIconFile</key>
<string>icon.icns</string>                 <!-- App icon -->

<key>CFBundleIdentifier</key>
<string>com.shellscriptmanager.app</string> <!-- Unique ID -->

<key>CFBundleName</key>
<string>Shell Script Manager</string>      <!-- Display name -->

<key>NSHighResolutionCapable</key>
<true/>                                     <!-- Retina support -->
```

6. Set Executable Permissions

```
chmod +x Contents/MacOS/shell_script_manager
```

Makes the binary executable so macOS can run it.

7. Code Sign the App Bundle

```
codesign --force --deep --sign - "Shell Script Manager.app"
xattr -cr "Shell Script Manager.app"
```

The script automatically performs **ad-hoc code signing**:

- `--sign -` uses ad-hoc signing (no developer certificate required)
- Prevents macOS "damaged" or "can't be verified" warnings
- `xattr -cr` removes quarantine attributes that trigger Gatekeeper
- **For personal use only** - distribution requires a Developer ID certificate

Customization

Change App Name

Edit `bundle_macos.sh`:

```
APP_NAME="Your App Name"
BUNDLE_NAME="Your App Name.app"
```

Change Bundle Identifier

```
BUNDLE_ID="com.yourcompany.yourapp"
```

Change Version

```
VERSION="1.0.0"
```

Use Different Icon

Replace the icon files in the `assets/` folder:

- `icon-256.png` - 256×256 pixels
- `icon-1024.png` - 1024×1024 pixels

Understanding Info.plist Keys

Key	Description
`CFBundleExecutable`	Name of the executable file to run
`CFBundleIconFile`	Name of the icon file (without extension)
`CFBundleIdentifier`	Unique reverse-DNS identifier for your app
`CFBundleName`	Display name shown in Finder and Dock
`CFBundleVersion`	Build version number
`CFBundleShortVersionString`	User-visible version (e.g., "1.0.0")
`LSMinimumSystemVersion`	Minimum macOS version required
`NSHighResolutionCapable`	Support for Retina displays
`NSPrincipalClass`	Main application class (NSApplication for Cocoa apps)

Hiding the Terminal Window

The app is configured to hide the terminal in release mode via `main.rs`:

```
#![cfg_attr(not(debug_assertions), windows_subsystem = "windows")]
```

Debug mode (`cargo run`):

- Terminal visible for debugging output
- Console output (println, eprintln) works

Release mode (.app bundle):

- No terminal window appears
- Console output is conditional (wrapped in `#[cfg(debug_assertions)]`)

Distribution

For Personal Use

Just copy `Shell Script Manager.app` to your Applications folder.

For Other Users

You'll need to:

1. **Code sign** the app (requires Apple Developer account)
2. **Notarize** the app (for macOS Gatekeeper)
3. Create a DMG installer (optional, but recommended)

Basic Code Signing

```
codesign --force --deep --sign "Developer ID Application: Your Name"
"Shell Script Manager.app"
```

Troubleshooting

"Malware" or "can't be verified" Warning (Gatekeeper)

macOS Gatekeeper blocks unsigned apps from unidentified developers. There are several ways to open your app:

Method 1: Right-click to Open (Recommended)

1. Right-click (or Control+click) on `Shell Script Manager.app`
2. Select **"Open"** from the context menu
3. In the dialog that appears, click **"Open"** again
4. macOS will remember your choice and won't ask again

Method 2: Remove Quarantine Attribute

```
sudo xattr -rd com.apple.quarantine "Shell Script Manager.app"
```

This removes the quarantine flag that triggers Gatekeeper warnings.

Method 3: Disable Gatekeeper Temporarily (Not Recommended)

```
sudo spctl --master-disable
```

This disables Gatekeeper system-wide. Remember to re-enable it:

```
sudo spctl --master-enable
```

Why This Happens:

- Ad-hoc code signing (`--sign -`) doesn't use an Apple Developer certificate
- macOS treats it as an "unidentified developer" app
- This is normal for personal development apps
- For distribution, you need a proper Developer ID certificate and notarization

App won't open / "damaged" error

If the app still shows as "damaged":

```
# Remove all extended attributes
xattr -cr "Shell Script Manager.app"

# Re-sign the app
codesign --force --deep --sign - "Shell Script Manager.app"
```

Or disable Gatekeeper temporarily:

```
xattr -cr "Shell Script Manager.app"
```

Icon not showing

- Ensure `icon.icns` is in `Contents/Resources/`
- Restart Finder: `killall Finder`
- Check Info.plist has `<key>CFBundleIconFile</key>`

App crashes immediately

- Check the executable has correct permissions: `ls -la "Shell Script Manager.app/Contents/MacOS/"`
- Run from terminal to see error messages: `"Shell Script Manager.app/Contents/MacOS/shell_script_manager"`

Database not found

The app looks for the database in different locations:

- **Debug**: Current directory (`./database.db`)
- **Release**: `~/Library/Application Support/ShellScriptManager/database.db`

Advanced: Automated Bundling

Add to your `Cargo.toml`:

```
[package.metadata.bundle]
name = "Shell Script Manager"
identifier = "com.shellscriptmanager.app"
icon = ["assets/icon-256.png", "assets/icon-1024.png"]
version = "0.1.0"
```

Or use tools like:

- `cargo-bundle`
- `cargo-packager`

Summary

The `.app` bundle is just a specially structured folder that macOS recognizes as an application:

```
YourApp.app/              ← This is the "application"
└── Contents/             ← Required folder
    ├── Info.plist        ← App metadata (required)
    ├── MacOS/            ← Executable folder (required)
    │   └── yourapp       ← Your binary
    └── Resources/        ← Optional resources
        └── icon.icns     ← App icon
```

When you double-click the `.app`, macOS:

1. Reads `Info.plist` to get metadata
2. Runs the executable specified in `CFBundleExecutable`
3. Shows the icon from `CFBundleIconFile`
4. Manages it as a normal macOS application

That's it! Your Rust application is now a native macOS app. 🎉