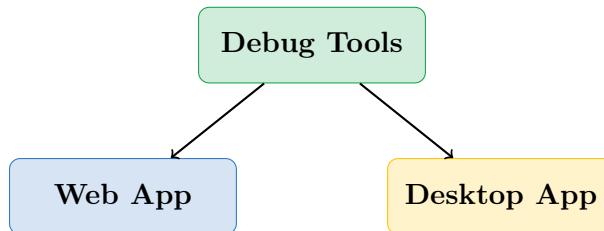


# DesignLibre

## Debugging Guide

TypeScript Web & Tauri Desktop



**Document Version:** 1.0  
**Application Version:** 0.1.0  
**Date:** January 2026

For use with Claude Code debugging sessions

## Contents

## 1 Introduction

This guide covers debugging practices for DesignLibre across both deployment targets:

- **TypeScript/Web** — Browser-based development build
- **Tauri/Desktop** — Native application (Linux, macOS, Windows)

The same TypeScript codebase runs in both environments, but the runtime context differs. This can cause behavioral differences in:

- Color rendering (WebView engine differences)
- File system access patterns
- Native dialog behavior
- Keyboard shortcuts
- Window management

## 2 Development Environment Setup

### 2.1 Starting Debug Builds

#### TypeScript/Web:

Browser Development

```
# Start Vite dev server with hot reload
npm run dev

# Open in browser
# http://localhost:5173

# Browser DevTools: F12 or Cmd+Option+I
```

#### Tauri/Desktop:

Desktop Development

```
# Start Tauri dev mode (includes Vite + native shell)
npm run tauri:dev

# DevTools in Tauri window: F12 or right-click -> Inspect

# With Rust debug logging:
RUST_LOG=debug npm run tauri:dev
```

## 2.2 Debug vs Release Builds

Aspect	Debug Build	Release Build
Command	<code>npm run tauri:dev</code>	<code>npm run tauri:build</code>
DevTools	Available (F12)	Hidden by default
Source Maps	Yes	No
Rust Logging	Verbose	Minimal
Performance	Slower	Optimized
Hot Reload	Yes	No

Table 1: Debug vs Release Build Comparison

## 3 Debugging Tools

### 3.1 Browser DevTools (Both Platforms)

DevTools work identically in browser and Tauri (press F12):

1. **Console Tab** — JavaScript logs, errors, warnings
2. **Elements Tab** — Inspect DOM, CSS styles
3. **Network Tab** — API calls, asset loading
4. **Sources Tab** — Set breakpoints, step through code
5. **Performance Tab** — Profile rendering, CPU usage
6. **Application Tab** — localStorage, IndexedDB

#### 3.1.1 Useful Console Commands

Listing 1: Debug Commands in Console

```

1 // Inspect the scene graph
2 window.__DESIGNLIBRE_DEBUG__ = true;
3 console.log(runtime.getSceneGraph());
4
5 // Check selected nodes
6 console.log(runtime.getSelection());
7
8 // Measure render performance
9 console.time('render');
10 renderer.render();
11 console.timeEnd('render');
12
13 // Export current state as JSON
14 JSON.stringify(runtime.exportState(), null, 2);

```

## 3.2 TypeScript-Specific Debugging

### TypeScript/Web:

VS Code Launch Configuration Create .vscode/launch.json:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "chrome",
      "request": "launch",
      "name": "Debug in Chrome",
      "url": "http://localhost:5173",
      "webRoot": "${workspaceFolder}/src",
      "sourceMaps": true
    }
  ]
}
```

## 3.3 Tauri-Specific Debugging

### Tauri/Desktop:

Rust Backend Debugging

```
# Enable verbose Rust logging
RUST_LOG=debug npm run tauri:dev

# Log levels: error, warn, info, debug, trace
RUST_LOG=tauri=debug,designlibre=trace npm run tauri:dev

# Check Rust panics with backtraces
RUST_BACKTRACE=1 npm run tauri:dev
```

### Tauri/Desktop:

Inspecting Tauri Commands Add logging to Rust commands in src-tauri/src/commands.rs:

```
#[command]
pub fn read_design_file(path: String) -> Result<DesignFile, String> {
    println!("[DEBUG] Reading file: {}", path); // Shows in terminal
    // ... rest of function
}
```

## 4 Common Issues: Web vs Tauri

## 4.1 Color Differences

### Known Issue: Color Rendering

Colors may appear different between browser and Tauri due to:

- Different WebView color profiles (sRGB handling)
- CSS color space interpretation
- GPU rendering differences

### Debugging Steps:

1. Compare the exact CSS/canvas color values in both environments
2. Check if colors are specified in hex, RGB, or HSL
3. Test with known reference colors (pure red: #FF0000)
4. Inspect canvas pixel values: `ctx.getImageData(x, y, 1, 1).data`

Listing 2: Color Comparison Debug

```

1 // In DevTools console - compare color rendering
2 const canvas = document.querySelector('canvas');
3 const ctx = canvas.getContext('2d');
4
5 // Draw test colors
6 ctx.fillStyle = '#FF0000'; // Pure red
7 ctx.fillRect(0, 0, 10, 10);
8
9 // Read back pixel
10 const pixel = ctx.getImageData(5, 5, 1, 1).data;
11 console.log('Red channel:', pixel[0]); // Should be 255
12 console.log('Green channel:', pixel[1]); // Should be 0
13 console.log('Blue channel:', pixel[2]); // Should be 0

```

## 4.2 Missing Functionality

When features work in web but not in Tauri (or vice versa):

1. **Check for browser-only APIs** — Some Web APIs aren't available in WebView
2. **Check Tauri permissions** — Missing capabilities in `capabilities/default.json`
3. **Check for native overrides** — Tauri may intercept certain behaviors

Listing 3: Feature Detection

```

1 // Check if running in Tauri
2 const isTauri = '__TAURI__' in window;
3 console.log('Running in Tauri:', isTauri);
4
5 // Check available Tauri APIs
6 if (isTauri) {
7   console.log('Tauri APIs:', Object.keys(window.__TAURI__));
8 }

```

## 5 Bug Documentation Best Practices

### 5.1 Bug Report Template

When documenting bugs, include the following information:

#### Bug Report Template

```
## Bug Title
[Clear, descriptive title]

## Environment
- Platform: [Linux/macOS/Windows/Web Browser]
- App Version: [e.g., 0.1.0]
- Build Type: [Dev/Release/AppImage/DMG]
- Browser (if web): [Chrome 120/Firefox 121/Safari 17]

## Steps to Reproduce
1. [First step]
2. [Second step]
3. [Third step]

## Expected Behavior
[What should happen]

## Actual Behavior
[What actually happens]

## Screenshots/Screen Recording
[Attach if visual bug]

## Console Output
```
[Paste relevant console logs/errors]
```

## Additional Context
[Any other relevant information]
```

### 5.2 Severity Levels

Level	Label	Description
P0	Critical	App crashes, data loss, security issue
P1	High	Feature completely broken, no workaround
P2	Medium	Feature impaired, workaround exists
P3	Low	Minor issue, cosmetic, edge case
P4	Enhancement	Not a bug, but could be improved

Table 2: Bug Severity Levels

### 5.3 Capturing Debug Information

### 5.3.1 Console Logs

Listing 4: Saving Console Output

```

1 # In DevTools Console, right-click -> Save as...
2 # Or copy/paste relevant sections
3
4 # For Tauri terminal output:
5 npm run tauri:dev 2>&1 | tee debug-log.txt

```

### 5.3.2 Screenshots and Screen Recording

- **Linux:** gnome-screenshot, flameshot, or PrtSc key
- **macOS:** Cmd+Shift+4 (area), Cmd+Shift+5 (recording)
- **Windows:** Win+Shift+S (Snip & Sketch)

### 5.3.3 State Export

Listing 5: Export Application State for Bug Report

```

1 // Run in DevTools console to capture full state
2 const debugState = {
3   timestamp: new Date().toISOString(),
4   platform: navigator.platform,
5   userAgent: navigator.userAgent,
6   isTauri: '__TAURI__' in window,
7   windowSize: { width: window.innerWidth, height: window.innerHeight },
8   // Add app-specific state
9   // sceneGraph: runtime.exportState(),
10  // selection: runtime.getSelection(),
11 };
12
13 // Copy to clipboard
14 copy(JSON.stringify(debugState, null, 2));
15 console.log('Debug state copied to clipboard');

```

## 6 Submitting Bugs to Claude

When reporting bugs to Claude Code for fixing, follow this format for best results:

### 6.1 Effective Bug Report Format

#### Optimal Bug Submission

```

## Bug: [Title]

**Environment:** Linux, Tauri AppImage, v0.1.0

**Reproduction:**
```

```
1. Open the app
2. Click on [specific element]
3. [Action that triggers bug]

**Expected:** [What should happen]
**Actual:** [What happens instead]

**Console errors:**
```
[Paste error messages here]
```

**Relevant files:**
- src/ui/components/toolbar.ts (line ~150)
- src/renderer/core/renderer.ts

**Screenshot:** [Describe or paste image path]
```

## 6.2 Information Priority

When submitting to Claude, prioritize:

1. **Exact error messages** — Copy full stack traces
2. **Reproduction steps** — Numbered, specific steps
3. **File locations** — If you know where the bug likely is
4. **Environment details** — Platform, build type
5. **What you've tried** — Any debugging you've done

## 6.3 Examples

### 6.3.1 Good Bug Report

**Bug:** Rectangle fill color appears darker in Tauri than in browser

**Environment:** Linux Ubuntu 22.04, Tauri AppImage 0.1.0

**Steps:**

1. Create a rectangle
2. Set fill color to #3B82F6 (blue)
3. Compare with same file in browser at localhost:5173

**Expected:** Same blue color in both

**Actual:** Tauri version is noticeably darker

**Console:** No errors

**Relevant:** Likely in `src/renderer/core/renderer.ts` or CSS color handling

### 6.3.2 Poor Bug Report

“Colors are wrong in the app”

*(Missing: which colors, where, what environment, steps to reproduce, expected vs actual)*

## 7 Debugging Checklist

Before submitting a bug, verify:

Reproduced the issue at least twice

Tested in both web and Tauri (if applicable)

Checked browser console for errors

Checked terminal output for Rust errors (Tauri)

Tried clearing cache / hard refresh (Ctrl+Shift+R)

Noted exact steps to reproduce

Captured relevant screenshots

Identified affected files (if possible)

## 8 Quick Reference

Action	Command/Key
Start web dev server	<code>npm run dev</code>
Start Tauri dev mode	<code>npm run tauri:dev</code>
Open DevTools	F12 or Cmd/Ctrl+Shift+I
Hard refresh	Ctrl+Shift+R
Clear console	Ctrl+L (in DevTools)
Pause on exceptions	DevTools → Sources → Pause on exceptions
Rust verbose logging	<code>RUST_LOG=debug npm run tauri:dev</code>
Rust backtrace	<code>RUST_BACKTRACE=1 npm run tauri:dev</code>
Build debug AppImage	<code>npm run tauri:build:debug --bundles appimage</code>

Table 3: Debug Quick Reference

---

*Document bugs thoroughly. Fix them systematically.*

---