Mermaid Tooling Monorepo

Table of Contents

Complete Guide to Building Chrome Extension and Desktop App with Latest Mermaid

Overview

This monorepo provides a complete solution for rendering **Mermaid diagrams** using the latest version built from source. It includes:

1. **Chrome MV3 Extension** - Renders Mermaid blocks on any webpage with overlay buttons
2. **Tauri Desktop App** - Full-featured editor with SVG/PNG export capabilities
3. **Shared Mermaid Build** - Single ESM bundle consumed by both applications

Table of Contents

Quick Start

Prerequisites

- **Node.js** ≥20.0.0
- **pnpm** ≥8.0.0
- **Git**
- **Rust** (for Tauri desktop app)

Installation

```
# Clone the repository<a></a>
git clone &lt;your-repo-url&gt;
cd mermaid-tooling-monorepo

# Install dependencies<a></a>
pnpm install

# Build Mermaid from source (latest release)<a></a>
pnpm run build:mermaid

# Build both applications<a></a>
pnpm run build:all
```

Building Mermaid from Source

The `build-mermaid.js` script clones the official Mermaid repository, builds it from source, and copies the ESM bundle to the shared vendor directory.

Usage

```
# Build latest stable release<a></a>
node scripts/build-mermaid.js

# Build specific version<a></a>
node scripts/build-mermaid.js v11.4.0

# Build from main branch (bleeding edge)<a></a>
node scripts/build-mermaid.js main
```

What It Does

1. **Clones** mermaid-js/mermaid repository
2. **Checks out** specified version/tag
3. **Installs** dependencies with pnpm
4. **Builds** using `pnpm run build`
5. **Copies** `mermaid.esm.min.mjs` to `packages/shared/vendor/`
6. **Creates** version metadata in `build-info.json`
7. **Cleans up** temporary files

Output Files

- `packages/shared/vendor/mermaid.esm.min.mjs` - Main ESM bundle
- `packages/shared/vendor/mermaid.d.ts` - TypeScript definitions
- `packages/shared/vendor/build-info.json` - Build metadata

Version Metadata Example

```
{
  "version": "v11.4.0",
  "commitHash": "a1b2c3d4",
  "buildDate": "2024-10-23T00:00:00.000Z",
  "file": "mermaid.esm.min.mjs",
  "source": "https://github.com/mermaid-js/mermaid.git"
}
```

Chrome Extension

Architecture

The Chrome MV3 extension injects content scripts that:

1. **Scan** pages for Mermaid code blocks
2. **Add overlay buttons** to each block
3. **Render on click** using the vendored Mermaid ESM
4. **Replace blocks** with rendered SVG

Supported Block Formats

- `<pre>...</pre>`
- Fenced code blocks: `<pre><code>...</code></pre>`

Key Files

manifest.json

```
{
  "manifest_version": 3,
  "name": "Mermaid Diagram Renderer",
  "permissions": ["scripting", "activeTab"],
  "content_scripts": [{
    "matches": ["&lt;all_urls&gt;"],
    "js": ["content.js"]
  }],
  "web_accessible_resources": [{
    "resources": ["vendor/mermaid.esm.min.mjs"],
    "matches": ["&lt;all_urls&gt;"]
  }]
}
```

content.ts (pseudo-code)

```
// Detect blocks
const blocks = document.querySelectorAll(
  'pre.mermaid, pre code.language-mermaid'
);

// Add overlay buttons
blocks.forEach(block =&gt; {
  const button = createButton();
  button.onclick = async () =&gt; {
    // Dynamic import from extension bundle
    const mermaid = await import(
      chrome.runtime.getURL('vendor/mermaid.esm.min.mjs')
    );

    // Render
    mermaid.initialize({ startOnLoad: false });
    const { svg } = await mermaid.render(
      generateId(),
      block.textContent
    );

    // Replace
    block.replaceWith(createSvgContainer(svg));
  };

  block.appendChild(button);
});

// Watch for dynamic content
new MutationObserver(mutations =&gt; {
  // Re-scan on DOM changes
}).observe(document.body, { childList: true, subtree: true });
```

Building the Extension

```
# Development mode<a></a>
pnpm --filter @mermaid-tools/extension dev

# Production build<a></a>
pnpm --filter @mermaid-tools/extension build
```

Loading in Chrome

1. Open `chrome://extensions`
2. Enable "Developer mode"
3. Click "Load unpacked"
4. Select `apps/extension/dist/`

Usage

1. **Navigate** to any webpage with Mermaid code blocks
2. **See overlay buttons** appear on detected blocks
3. **Click button** to render diagram inline
4. **Use keyboard shortcut** `Ctrl+Alt+M` (or `Cmd+Alt+M` on Mac) to re-scan page

Per-Site Configuration

The extension includes an options page (`options.html`) where you can:

- Enable/disable on specific domains
- Configure overlay button position
- Set default theme (default, dark, forest, neutral)

CSP Compatibility

The extension loads Mermaid from its own bundle (not CDN) to avoid Content Security Policy conflicts. This ensures it works on sites with strict CSP headers.

Desktop Application

Architecture

The Tauri app provides a split-pane interface:

- **Left pane**: Code editor (textarea)
- **Right pane**: SVG preview
- **Toolbar**: Render, export, and utility buttons

Features

- **Live rendering** with Mermaid v11+ features
- **SVG export** - Save diagrams as vector graphics
- **PNG export** - Rasterize to high-quality PNG (HiDPI support)
- **Example templates** - Load common diagram types
- **Syntax validation** - Clear error messages
- **Keyboard shortcuts** - Fast workflow

Key Files

tauri.conf.json

```
{
  "tauri": {
    "allowlist": {
      "fs": { "writeFile": true },
      "dialog": { "save": true }
    },
```

```
    "windows": [{
      "title": "Mermaid Desktop Editor",
      "width": 1200,
      "height": 800
    }]
  }
}
```

main.ts (pseudo-code)

```
import mermaid from './vendor/mermaid.esm.min.mjs';

mermaid.initialize({ startOnLoad: false });

async function renderDiagram() {
  const code = editor.value;

  try {
    const { svg } = await mermaid.render(
      generateId(),
      code
    );
    preview.innerHTML = svg;
    enableExportButtons();
  } catch (error) {
    showError(error.message);
  }
}

async function exportSVG() {
  const svg = preview.querySelector('svg');
  const blob = new Blob(
    [svg.outerHTML],
    { type: 'image/svg+xml' }
  );

  const path = await dialog.save({
    defaultPath: `mermaid-${Date.now()}.svg`
  });

  await fs.writeBinaryFile(path, await blob.arrayBuffer());
}

async function exportPNG() {
  const svg = preview.querySelector('svg');
  const canvas = document.createElement('canvas');
  const ctx = canvas.getContext('2d');

  // Scale for HiDPI
  const scale = 2;
  canvas.width = svg.width.baseVal.value * scale;
  canvas.height = svg.height.baseVal.value * scale;
  ctx.scale(scale, scale);

  // Serialize SVG to image
  const img = new Image();
  const svgBlob = new Blob([svg.outerHTML], {
    type: 'image/svg+xml'
  });
  const url = URL.createObjectURL(svgBlob);

  img.onload = async () =&gt; {
    ctx.drawImage(img, 0, 0);
    const dataUrl = canvas.toDataURL('image/png');

    const path = await dialog.save({
      defaultPath: `mermaid-${Date.now()}.png`
    });

    // Convert data URL to bytes and save
    const bytes = Uint8Array.from(
```

```
      atob(dataUrl.split(',')[1]),
      c =&gt; c.charCodeAt(0)
    );
    await fs.writeBinaryFile(path, bytes);

    URL.revokeObjectURL(url);
  };

  img.src = url;
}
```

Building the Desktop App

```
# Development mode (hot reload)<a></a>
pnpm --filter @mermaid-tools/desktop dev

# Production build (Linux AppImage)<a></a>
pnpm --filter @mermaid-tools/desktop build

# Cross-platform builds<a></a>
pnpm --filter @mermaid-tools/desktop tauri build --target x86_64-pc-windows-msvc
pnpm --filter @mermaid-tools/desktop tauri build --target aarch64-apple-darwin
```

Output Artifacts

- **Linux**: `apps/desktop/src-tauri/target/release/bundle/appimage/mermaid-desktop-editor_1.0.0_amd64.AppImage`
- **Windows**: `.msi` and `.exe` installers
- **macOS**: `.dmg` and `.app` bundle

Keyboard Shortcuts

- `Ctrl+R` / `Cmd+R` - Render diagram
- `Ctrl+E` / `Cmd+E` - Export SVG
- `Ctrl+Shift+E` / `Cmd+Shift+E` - Export PNG
- `Ctrl+K` / `Cmd+K` - Clear editor
- `Ctrl+L` / `Cmd+L` - Load example

Architecture

Shared Mermaid Bundle

Both applications consume the same ESM bundle from `packages/shared/vendor/`. This ensures:

- **Consistent behavior** across extension and desktop app
- **Single build process** for Mermaid
- **Easy version upgrades** (rebuild once, deploy to both)

Monorepo Structure

```
packages/shared/vendor/     # Shared Mermaid build
apps/extension/vendor/      # Symlink → packages/shared/vendor/
apps/desktop/src/vendor/    # Symlink → packages/shared/vendor/
```

Loading Strategy

Extension (Content Script)

```
// Load from extension bundle (avoids CSP issues)
const mermaid = await import(
  chrome.runtime.getURL('vendor/mermaid.esm.min.mjs')
);
```

Desktop (Frontend)

```
// Direct ES module import
import mermaid from './vendor/mermaid.esm.min.mjs';
```

API Usage Pattern

Both apps follow the same pattern:

1. **Initialize once**: `mermaid.initialize({ startOnLoad: false })`
2. **Render on demand**: `await mermaid.render(id, code)`
3. **Insert SVG**: Replace target element with returned SVG string

This keeps behavior identical between the two environments.

Development Workflow

Initial Setup

```
# Install pnpm if needed<a></a>
npm install -g pnpm

# Clone and setup<a></a>
git clone &lt;repo&gt;
cd mermaid-tooling-monorepo
pnpm install

# Build Mermaid<a></a>
pnpm run build:mermaid
```

Daily Development

```
# Extension dev server (hot reload)<a></a>
pnpm run dev:extension

# Desktop dev server<a></a>
pnpm run dev:desktop

# Both simultaneously<a></a>
pnpm run dev:extension &amp;amp; pnpm run dev:desktop
```

Updating Mermaid Version

```
# Switch to specific version<a></a>
node scripts/build-mermaid.js v11.5.0

# Or use latest<a></a>
node scripts/build-mermaid.js

# Rebuild apps to use new bundle<a></a>
pnpm run build:all
```

Testing

```
# Run all tests<a></a>
pnpm test

# Test specific package<a></a>
pnpm --filter @mermaid-tools/extension test
pnpm --filter @mermaid-tools/desktop test
```

CI/CD Integration

Example GitHub Actions workflow:

```
name: Build and Release

on:
  push:
    tags:
      - 'v*'

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '20'

      - name: Install pnpm
        run: npm install -g pnpm

      - name: Install dependencies
        run: pnpm install

      - name: Build Mermaid
        run: pnpm run build:mermaid

      - name: Build all apps
        run: pnpm run build:all

      - name: Package extension
        run: cd apps/extension/dist &amp;amp;&amp;amp; zip -r ../extension.zip .

      - name: Create release
        uses: actions/create-release@v1
        with:
          files: |
            apps/extension/extension.zip
            apps/desktop/src-tauri/target/release/bundle/**/*
```

Troubleshooting

Common Issues

Mermaid Build Fails

Problem: `pnpm run build` fails in mermaid repo

Solutions:

- Ensure Node.js ≥20.0.0: `node --version`
- Clear pnpm cache: `pnpm store prune`
- Try different version: `node scripts/build-mermaid.js v11.3.0`

Extension Not Loading

Problem: Chrome won't load unpacked extension

Solutions:

- Check manifest.json syntax
- Ensure dist/ folder exists
- Look for errors in chrome://extensions page
- Rebuild: `pnpm --filter @mermaid-tools/extension build`

Diagrams Not Rendering

Problem: Clicking button does nothing

Solutions:

- Open DevTools Console for error messages
- Check if Mermaid bundle loaded: Network tab should show `mermaid.esm.min.mjs`
- Verify syntax: Test diagram at mermaid.live
- Check CSP headers: May block inline scripts

Desktop App Won't Start

Problem: Tauri app crashes on launch

Solutions:

- Check Rust toolchain: `rustc --version`
- Rebuild Tauri: `pnpm --filter @mermaid-tools/desktop tauri build`
- Check console logs: Look in terminal output
- Verify tauri.conf.json syntax

Export Fails

Problem: SVG/PNG export produces empty file

Solutions:

- Ensure diagram is rendered first
- Check file permissions
- Try different export path
- Check browser console for Canvas errors

Debug Mode

Enable verbose logging:

Extension:

```
// In content.ts
const DEBUG = true;
if (DEBUG) console.log('Mermaid block detected:', block);
```

Desktop:

```
// In main.ts
mermaid.initialize({
  startOnLoad: false,
  logLevel: 'debug'
});
```

Performance Issues

Large Diagrams:

- Use `mermaid.mermaidAPI.initialize({ maxTextSize: 50000 })`
- Consider pagination for complex flowcharts
- Enable Hardware Acceleration in Chrome

Memory Leaks:

- Call `mermaid.reset()` between renders
- Dispose of old SVG elements properly
- Avoid re-rendering unnecessarily

Advanced Topics

Custom Themes

Create custom Mermaid themes:

```
mermaid.initialize({
  startOnLoad: false,
  theme: 'base',
  themeVariables: {
    primaryColor: '#ff6b6b',
    primaryTextColor: '#fff',
    primaryBorderColor: '#ff5252',
    lineColor: '#4ecdc4',
    secondaryColor: '#ffe66d',
    tertiaryColor: '#a8e6cf'
  }
});
```

Security Configuration

For extension security:

```
{
  "content_security_policy": {
    "extension_pages": "script-src 'self'; object-src 'self'"
  }
}
```

For desktop CSP:

```
{
  "security": {
    "csp": "default-src 'self'; script-src 'self' 'unsafe-eval'"
  }
}
```

Plugin Architecture

Extend functionality with plugins:

```
// Custom plugin for extension
class MermaidPlugin {
  async beforeRender(code: string): Promise&lt;string&gt; {
    // Preprocess Mermaid code
    return code.replace(/TODO:/g, 'Action:');
  }

  async afterRender(svg: string): Promise&lt;string&gt; {
    // Post-process SVG
    return svg.replace(/fill="#000"/g, 'fill="#333"');
  }
}
```

Testing Strategy

Unit Tests (Vitest):

```
import { describe, it, expect } from 'vitest';
import { detectMermaidBlocks } from './content';

describe('detectMermaidBlocks', () =&gt; {
  it('finds pre.mermaid blocks', () =&gt; {
    document.body.innerHTML = '<pre>graph TD</pre>';
    const blocks = detectMermaidBlocks();
    expect(blocks).toHaveLength(1);
  });
});
```

Integration Tests:

```
import { test, expect } from '@playwright/test';

test('extension renders diagram', async ({ page }) =&gt; {
  await page.goto('https://example.com/with-mermaid');
  await page.click('.mermaid-render-button');
  await expect(page.locator('svg')).toBeVisible();
});
```

Deployment

Extension:

1. Build: `pnpm run build:extension`
2. Create zip: `cd apps/extension/dist &amp;&amp; zip -r extension.zip .`
3. Upload to Chrome Web Store
4. Submit for review

Desktop:

1. Build: `pnpm run build:desktop`

2. Sign binaries (platform-specific)
3. Create installer packages
4. Distribute via GitHub Releases or app stores

License

MIT License - See LICENSE file for details

Contributing

1. Fork the repository
2. Create feature branch: `git checkout -b feature/my-feature`
3. Commit changes: `git commit -am 'Add feature'`
4. Push to branch: `git push origin feature/my-feature`
5. Submit pull request

Support

- **Issues**: GitHub Issues
- **Discussions**: GitHub Discussions
- **Email**: support@example.com

Built with ❤ using Mermaid, Tauri, and modern web technologies.