

Obsidian Plugin Project Structure

TypeScript-based Obsidian plugin with commands, settings tab, and modular architecture.

Updated 2025-12-22

#obsidian #plugin #typescript #notes #markdown

PNG

PDF

Copy

Prompt

Project Directory

my-plugin/

- manifest.json Plugin metadata
- main.ts Plugin entry po...
- src/ Source code
 - settings/
 - settings.ts Settings interf...
 - settingsTab.ts Settings UI
 - commands/ Command handlers
 - index.ts Register comman...
 - sampleCommand.ts
 - views/ Custom views
 - sampleView.ts ItemView subcla...
 - modals/
 - sampleModal.ts Modal dialog
 - utils/
 - helpers.ts
- styles.css Plugin styles
- package.json
- tsconfig.json
- esbuild.config.mjs Build configura...
- versions.json Version compati...
- .gitignore
- README.md

Why This Structure?

This structure separates concerns into folders: commands, views, modals, and settings. The `main.ts` entry point extends `Plugin` and orchestrates everything. TypeScript provides type safety with Obsidian's extensive API.

Key Directories

main.ts - Extends `Plugin`, handles `onload()` and `onunload()`

src/settings/ - Settings interface and `PluginSettingTab`

src/commands/ - Commands registered via `addCommand()`

src/views/ - Custom views extending `ItemView`

Plugin Entry Point

```
// main.ts
import { Plugin } from 'obsidian';
import { MySettings, DEFAULT_SETTINGS } from './src/setting...
import { MySettingTab } from './src/settings/settingsTab';

export default class MyPlugin extends Plugin {
  settings: MySettings;

  async onload() {
    await this.loadSettings();
    this.addSettingTab(new MySettingTab(this.app, this));
    this.addCommand({ id: 'sample', name: 'Sample', callba
  }
}
```

Getting Started

- Clone the sample plugin from `obsidianmd/obsidian-sample-plugin`
- `npm install` to install dependencies
- Update `manifest.json` with your plugin info
- `npm run dev` to start watching for changes
- Copy folder to `.obsidian/plugins/` in a test vault
- Enable plugin in Obsidian settings

Core Plugin APIs

this.app.vault - Read, create, modify notes and files

this.app.workspace - Manage leaves, views, and layout

this.addCommand() - Register commands with hotkeys

this.registerView() - Add custom sidebar views

Best Practices

- Always clean up in `onunload()` method
- Use `this.registerEvent()` for auto-cleanup of events
- Store settings with `this.saveData()` and `this.loadData()`
- Prefix CSS classes to avoid conflicts
- Test with both light and dark themes

Manifest Files

The `manifest.json` defines `id`, `name`, `version`, `minAppVersion`, and `author`. The `versions.json` maps your plugin versions to minimum Obsidian versions for compatibility checking.

When To Use This

- Automating note workflows
- Adding custom markdown processing
- Integrating external services
- Creating custom views and panels
- Extending editor functionality

Trade-offs

Private API - Obsidian API is not fully documented, check source

Build step - TypeScript requires esbuild compilation

Testing - No official testing framework, manual testing in vault