

# VS Code Extension Project Structure


TypeScript-based VS Code extension with commands, configuration, and activation events.

Updated 2025-12-22

#vscode #extension #typescript #editor #ide

 PNG

 PDF

 Copy

 Prompt

## Project Directory

### my-extension/

- package.json Extension manifi...
- tsconfig.json
- .vscodeignore Exclude from pa...
- src/ Source code
  - extension.ts Entry point, ac...
  - commands/ Command handlers
    - index.ts Register all co...
    - helloWorld.ts Example command
  - providers/ VS Code provide...
    - completionProvider.ts IntelliSense
    - hoverProvider.ts
    - treeDataProvider.ts Sidebar tree vi...
  - webview/ Webview panels
    - webviewPanel.ts Panel manager
    - media/
      - main.js
      - style.css
    - utils/
      - config.ts Read extension ...
      - logger.ts Output channel ...
      - types.ts Shared type def...
  - test/ Extension tests
    - runTest.ts Test runner ent...
    - suite/
      - index.ts Test suite setup
      - extension.test.ts
  - resources/ Static assets
    - icon.png Extension icon
    - dark/ Dark theme icons
      - icon.svg
    - light/ Light theme ico...
      - icon.svg
  - .vscode/ Development con...
    - launch.json Debug configura...
    - tasks.json Build tasks
    - settings.json
  - CHANGELOG.md
  - .gitignore

## Why This Structure?

This structure separates concerns into clear folders: commands, providers, and webviews. The `extension.ts` entry point handles activation while delegating to specialized modules. TypeScript provides type safety and better VS Code API intellisense.

## Key Directories

**src/commands/** - Command handlers registered via `contributes.commands`

**src/providers/** - IntelliSense, hovers, tree views, and other providers

**src/webview/** - Custom UI panels with HTML/CSS/JS

**resources/** - Icons with dark/light theme variants

## Package.json Contributes

```
// package.json contributes section
"contributes": {
  "commands": [{
    "command": "myExtension.helloWorld",
    "title": "Hello World"
  }],
  "configuration": {
    "title": "My Extension",
    "properties": {
      "myExtension.enable": { "type": "boolean", "default"
    }
  }
}
```

## Getting Started

- `npm install -g yo generator-code`
- `yo code` to scaffold a new extension
- `npm install` to install dependencies
- Press `F5` to launch Extension Development Host
- `Ctrl+Shift+P` → Run your command

## Activation Events

**onCommand** - Activate when command is invoked

**onLanguage** - Activate for specific file types

**workspaceContains** - Activate if workspace has specific files

**\*** - Activate on startup (avoid if possible)

## Best Practices

- Use lazy activation—avoid `*` activation event
- Dispose resources in `deactivate()` function
- Store state in `ExtensionContext.globalState`
- Use `OutputChannel` for logging, not `console.log`
- Bundle with esbuild or webpack for faster load times

## Core VS Code APIs

**vscode.commands** - Register and execute commands

**vscode.workspace** - Access files, config, workspace folders

**vscode.window** - UI: editors, terminals, notifications

**vscode.languages** - Language features, diagnostics

## When To Use This

- Adding new commands or keybindings
- Custom IntelliSense for a language
- Sidebar panels and tree views
- Integrating external tools into VS Code
- Custom editors or webview-based UIs

## Trade-offs

**TypeScript required** - Best DX but adds build step

**API surface** - Large API to learn, check extension samples

**Testing complexity** - Requires Extension Development Host for integration tests