JetBrains Plugin Project Structure

IntelliJ Platform plugin with Kotlin, Gradle build system, and support for all JetBrains IDEs.

#jetbrains #intellij #plugin #kotlin #ide #pycharm #webstorm

PNG PDF Copy Prompt

Project Directory

```
my-plugin/
    build.gradle.kts  Gradle build co…
    settings.gradle.kts
    gradle.properties  Plugin and IDE …
    gradlew
    gradlew.bat
  gradle/
    wrapper/
      gradle-wrapper.jar
      gradle-wrapper.properties
  src/
    main/
      kotlin/  Plugin source c…
        com/example/myplugin/
          MyPlugin.kt  Plugin entry po…
          actions/  Menu/toolbar ac…
            MyAction.kt
            ActionGroup.kt
          services/  Application/pro…
            MyProjectService.kt  Per-project ser…
            MyApplicationService.kt  Global service
          toolwindow/
            MyToolWindowFactory.kt
            MyToolWindow.kt
          settings/
            MySettingsConfigurable.kt  Settings UI
            MySettingsState.kt  Persistent state
          listeners/
            MyProjectListener.kt
      resources/
        META-INF/
          plugin.xml  Plugin descript…
          pluginIcon.svg
        messages/  i18n bundles
          MyBundle.properties
    test/
      kotlin/
        com/example/myplugin/
          MyPluginTest.kt
  README.md
  .gitignore
  CHANGELOG.md
```

Why This Structure?

This structure follows JetBrains' official plugin template. Actions handle menu/toolbar clicks, services provide project or application-scoped logic, and `plugin.xml` wires everything together. Kotlin is preferred over Java for new plugins.

Key Directories

META-INF/plugin.xml - Plugin descriptor - actions, services, extensions

actions/ - Menu items and toolbar buttons

services/ - Singleton services (project or application scope)

toolwindow/ - Side panel tool windows

settings/ - Settings UI and persistent state

Plugin Descriptor

```
com.example.myplugin
My Plugin
Your Name

com.intellij.modules.platform
```

Getting Started

1. Clone JetBrains plugin template from GitHub
2. Update `gradle.properties` with your plugin details
3. Run `./gradlew runIde` to test in sandbox IDE
4. Implement actions and services in Kotlin
5. Build with `./gradlew buildPlugin`

When To Use This

- Adding features to JetBrains IDEs
- Custom language support or syntax highlighting
- Integration with external tools or services
- Code generation or refactoring tools
- Custom inspections and quick fixes

IDE Compatibility

Platform plugins - Works in all IDEs: IntelliJ, PyCharm, WebStorm, etc.

Product-specific - Add depends for Java, Python, JavaScript support

Version range - Specify min/max IDE versions in gradle.properties

Best Practices

- Use `@Service` annotation for services (no XML needed)
- Prefer Kotlin coroutines over background threads
- Use `MessageBundle` for user-facing strings
- Test with multiple IDE versions before release
- Follow IntelliJ Platform UI guidelines

Trade-offs

Learning curve - IntelliJ Platform API is large and complex

IDE downloads - Sandbox IDE downloaded for each version tested

Breaking changes - Platform APIs can change between major versions