Alfred Workflow Project Structure

Alfred workflow with Script Filters, hotkeys, and proper workflow distribution setup for macOS.

#alfred #workflow #macos #automation #productivity

PNG PDF Copy Prompt

Project Directory

```
my-workflow/
    info.plist  Workflow defini…
    icon.png  Workflow icon (…
  src/  Source scripts
    main.py  Primary script …
    action.py  Action handler
    config.py  Configuration h…
    utils.py  Helper functions
  lib/  Bundled depende…
    workflow/  Alfred-Workflow…
      __init__.py
      workflow.py
      web.py
  icons/  Result icons
    default.png
    error.png
    success.png
  scripts/
    build.sh  Package .alfred…
    install-deps.sh
  README.md
  LICENSE
  .gitignore
  requirements.txt
```

Why This Structure?

Alfred workflows are macOS-specific automation tools. This structure separates source code from bundled dependencies. The `lib/` folder contains vendored packages since workflows run in isolation. Script Filters output JSON to Alfred for displaying results.

Key Directories

`info.plist` - Workflow definition, objects, and connections

`src/` - Your Python scripts for Script Filters and actions

`lib/` - Vendored dependencies (bundled with workflow)

`icons/` - Icons displayed in Alfred results

Script Filter

```python
# src/main.py - Script Filter
import sys
sys.path.insert(0, 'lib')
from workflow import Workflow3

def main(wf):
    query = wf.args[0] if wf.args else ''

    items = [
        {'title': 'Result 1', 'arg': 'action1'},
        {'title': 'Result 2', 'arg': 'action2'},
    ]

    for item in items:
        if query.lower() in item['title'].lower():
            wf.add_item(title=item['title'], arg=item['arg

    wf.send_feedback()

if __name__ == '__main__':
    wf = Workflow3()
    sys.exit(wf.run(main))
```

Getting Started

1. Create workflow in Alfred Preferences
2. Add Script Filter object with `/usr/bin/python3 src/main.py "{query}"`
3. Install alfred-workflow: `pip install --target=lib alfred-workflow`
4. Connect Script Filter to actions
5. Export as `.alfredworkflow` for distribution

When To Use This

- Building macOS productivity launchers
- Quick access to frequently used actions
- Integration with web APIs and services
- File and folder manipulation tools
- System automation and shortcuts

Workflow Objects

`Script Filter` - Shows results as you type, outputs JSON

`Run Script` - Executes action with selected item's arg

`Hotkey` - Global keyboard shortcut trigger

`Keyword` - Trigger by typing a keyword

Best Practices

- Bundle all dependencies in `lib/` for portability
- Use Workflow3 class for modern Alfred features
- Cache API responses with `wf.cached_data()`
- Store settings with `wf.settings`
- Add `wf.logger` debug output for troubleshooting

Trade-offs

`macOS only` - Alfred is exclusive to macOS

`Python 3 bundling` - Must vendor dependencies, no pip at runtime

`Paid app` - Users need Alfred Powerpack for workflows