Bringing a new API to KiCad

Jon Evans

First, some background

The era of SWIG

The era of SWIG 😀

The era of SWIG 😅

The era of SWIG 😬

SWIG (in KiCad) Challenges

1. The interface is fragile
2. Modern C++ doesn’t always play nice
3. Slows down the build
4. Python plugins can completely break KiCad

Packaging Challenges

1. wxPython and wxWidgets
2. Bundled Python interpreter
3. Dependency management

Developer Experience Challenges

1. Environment your code runs in is hard to re-create in a development context
2. KiCad Python Console is… okay

How We Didn’t Fix It

Make a real API layer in C++; have SWIG wrap that.

- Minimal paradigm shift! Not that risky.
- Doesn’t solve most of the problems ☹️

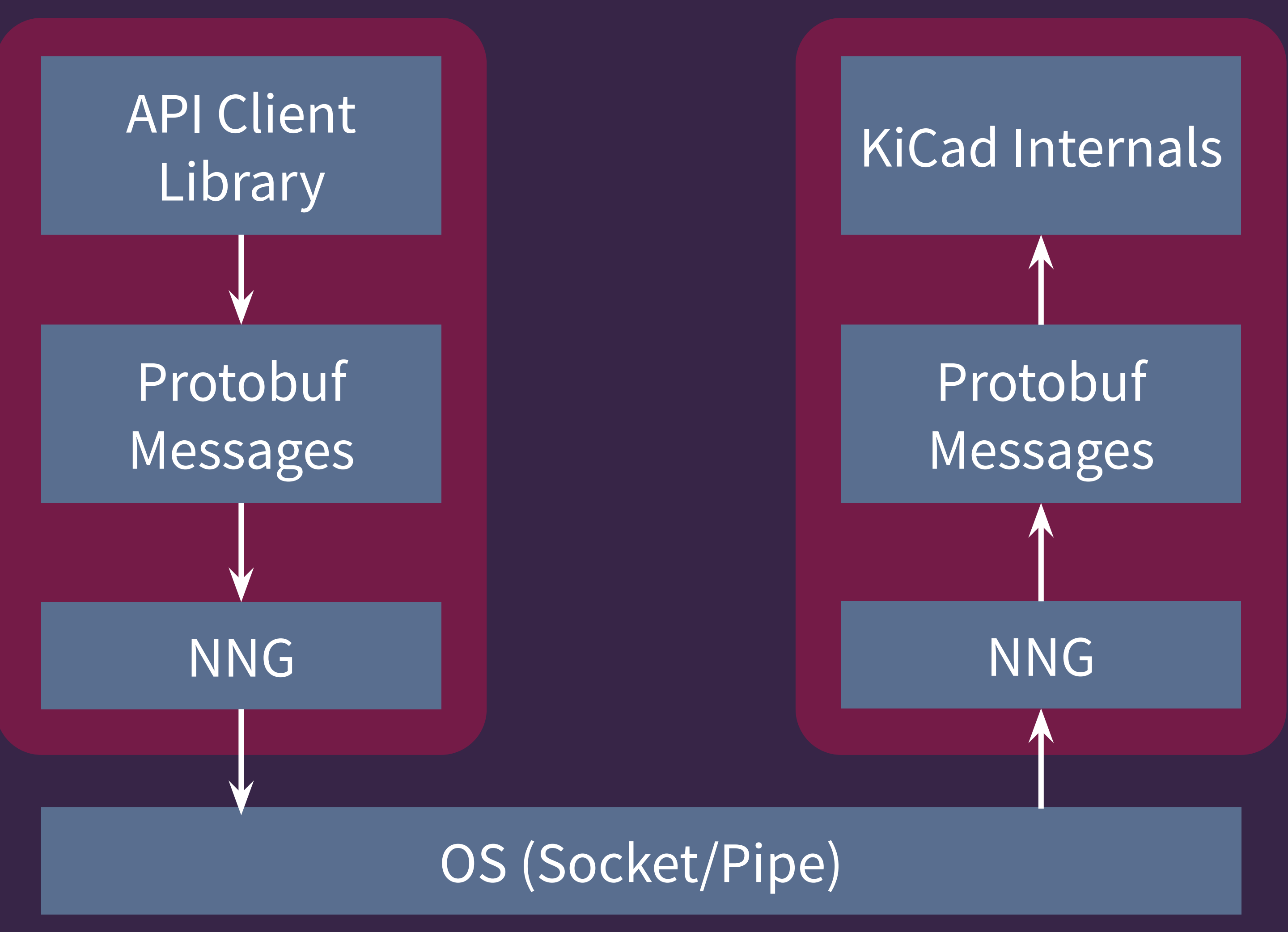
Make a real API layer in C++; have Pybind11 wrap that.

- (Maybe) solves a few more problems!
- Still doesn’t solve the rest of them ☹️

Introducing the IPC API

Key Principles

1. Be robust against code evolution in KiCad itself
2. API users should not have direct access to KiCad internal state (run in external process)
3. Developer experience is important, both for the KiCad team and API users


API Client Library
Protobuf Messages
NNG
KiCad Internals
Protobuf Messages
NNG
OS (Socket/Pipe)

Transport: Why Nanomsg-NG (NNG)?

- Simple to implement
- Target native IPC mechanisms (UNIX sockets, named pipes on Windows) with the same code
- Good availability across platforms and languages
- *Also considered: ZeroMQ, gRPC, D-Bus*

Protocol: Why Protocol Buffers (protobuf)?

- The API is the message definitions
- *When used well*, allows for API evolution over time and cross-version compatibility
- Alternatives are either not as widely used and supported, or don’t solve the same problems

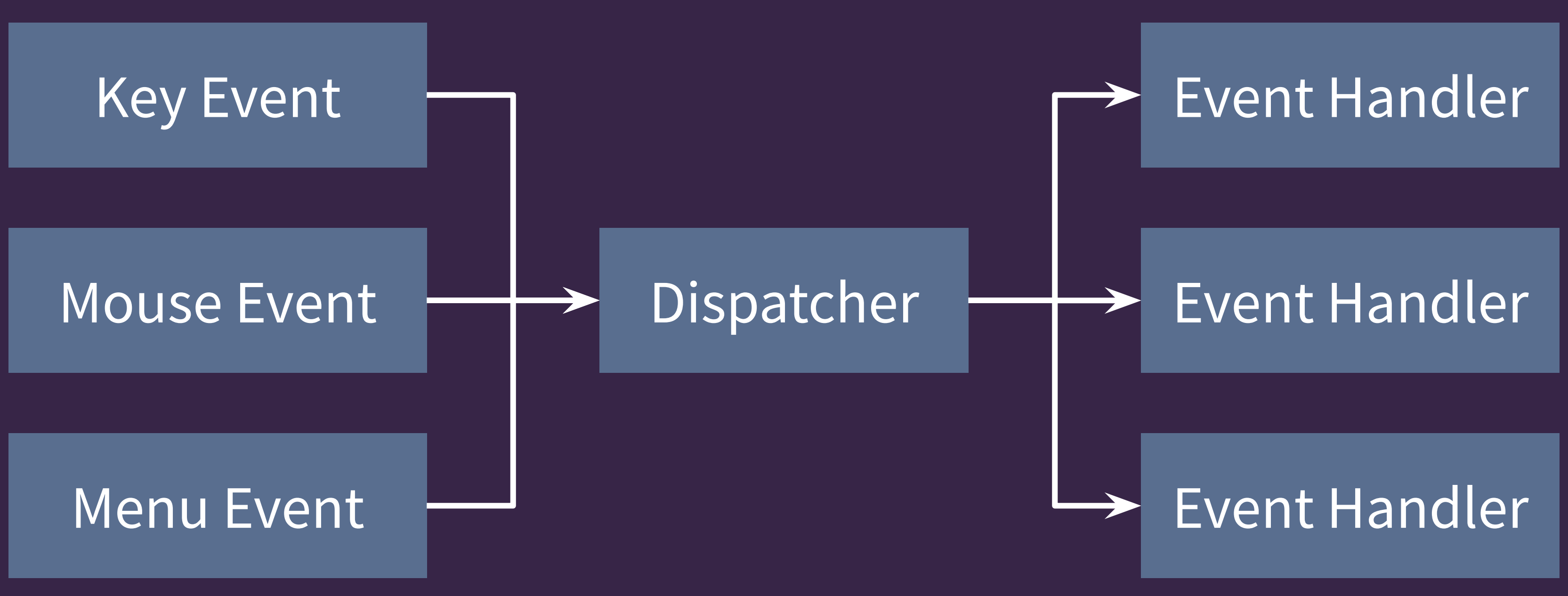
```
message Vector2
{
  int64 x_nm = 1;
  int64 y_nm = 2;
}

message Box2
{
  kiapi.common.types.Vector2 position = 1;
  kiapi.common.types.Vector2 size = 2;
}
```

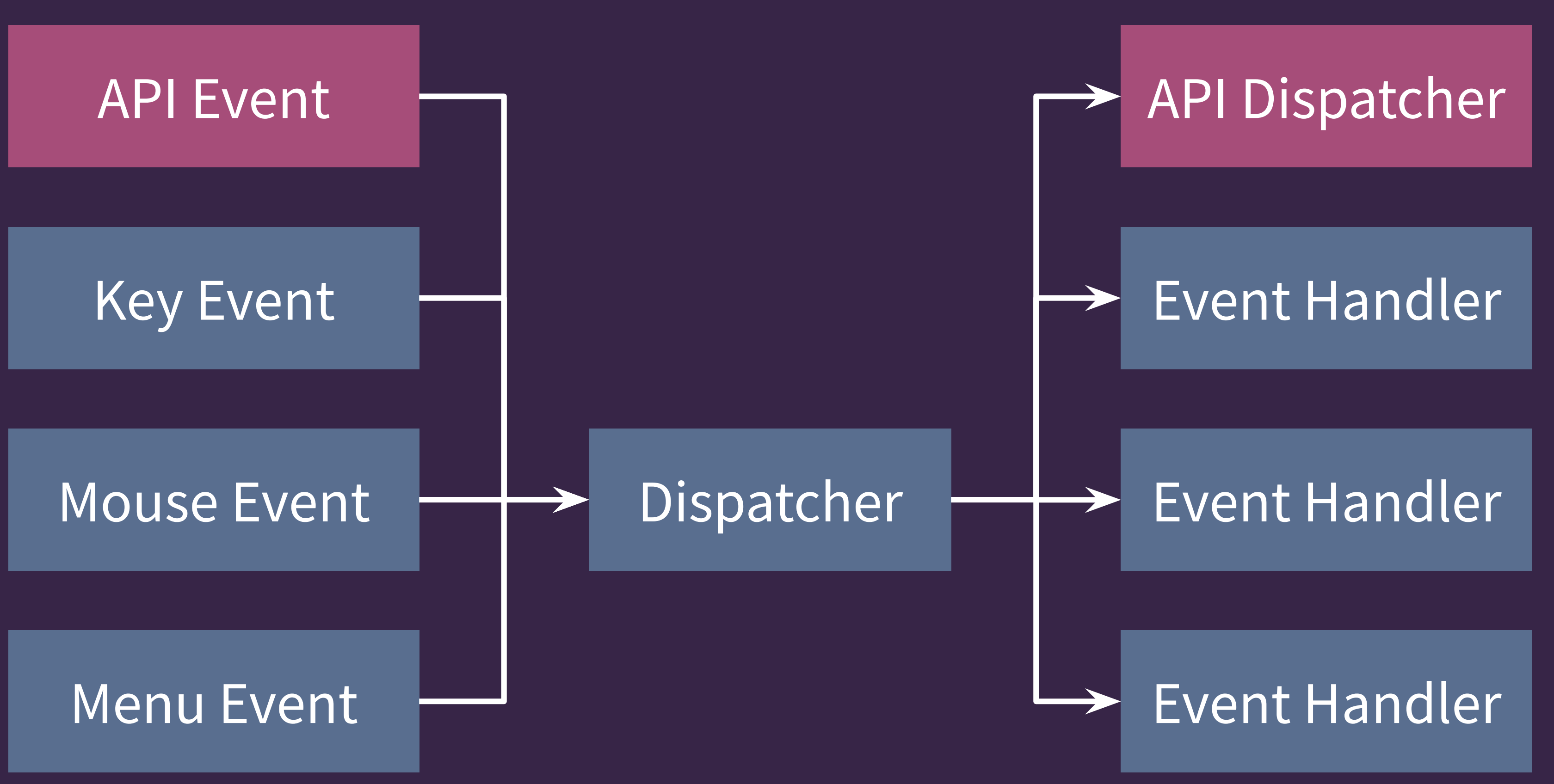
```
// returns kiapi.common.types.Box2
message GetTextExtents
{
  kiapi.common.types.Text text = 1;
}
```

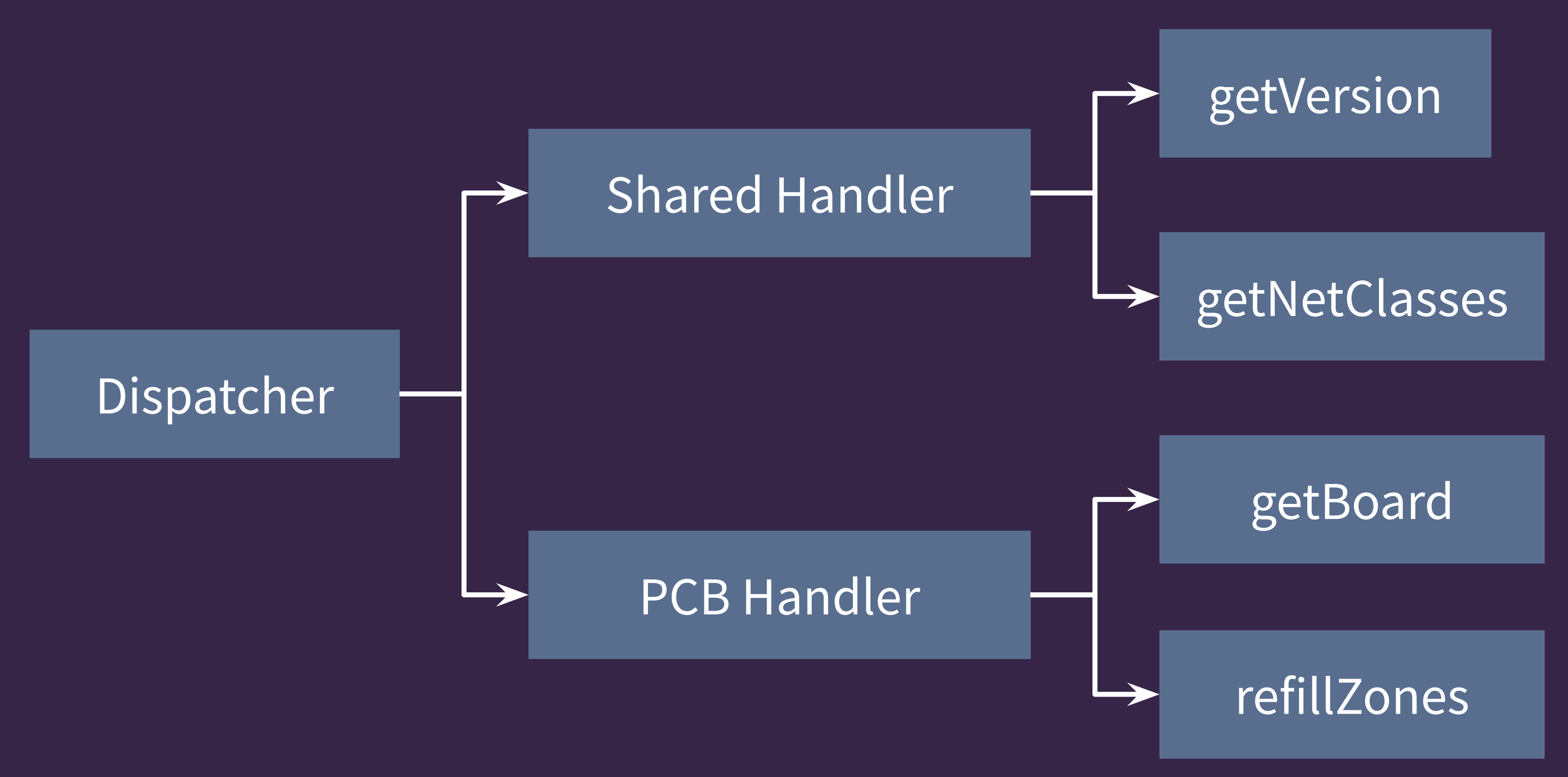
```
message ApiRequest
{
  ApiRequestHeader header = 1;
  google.protobuf.Any msg = 2;
}

message ApiResponse
{
  ApiResponseHeader header = 1;
  ApiResponseStatus status = 2;
  google.protobuf.Any msg = 3;
}
```

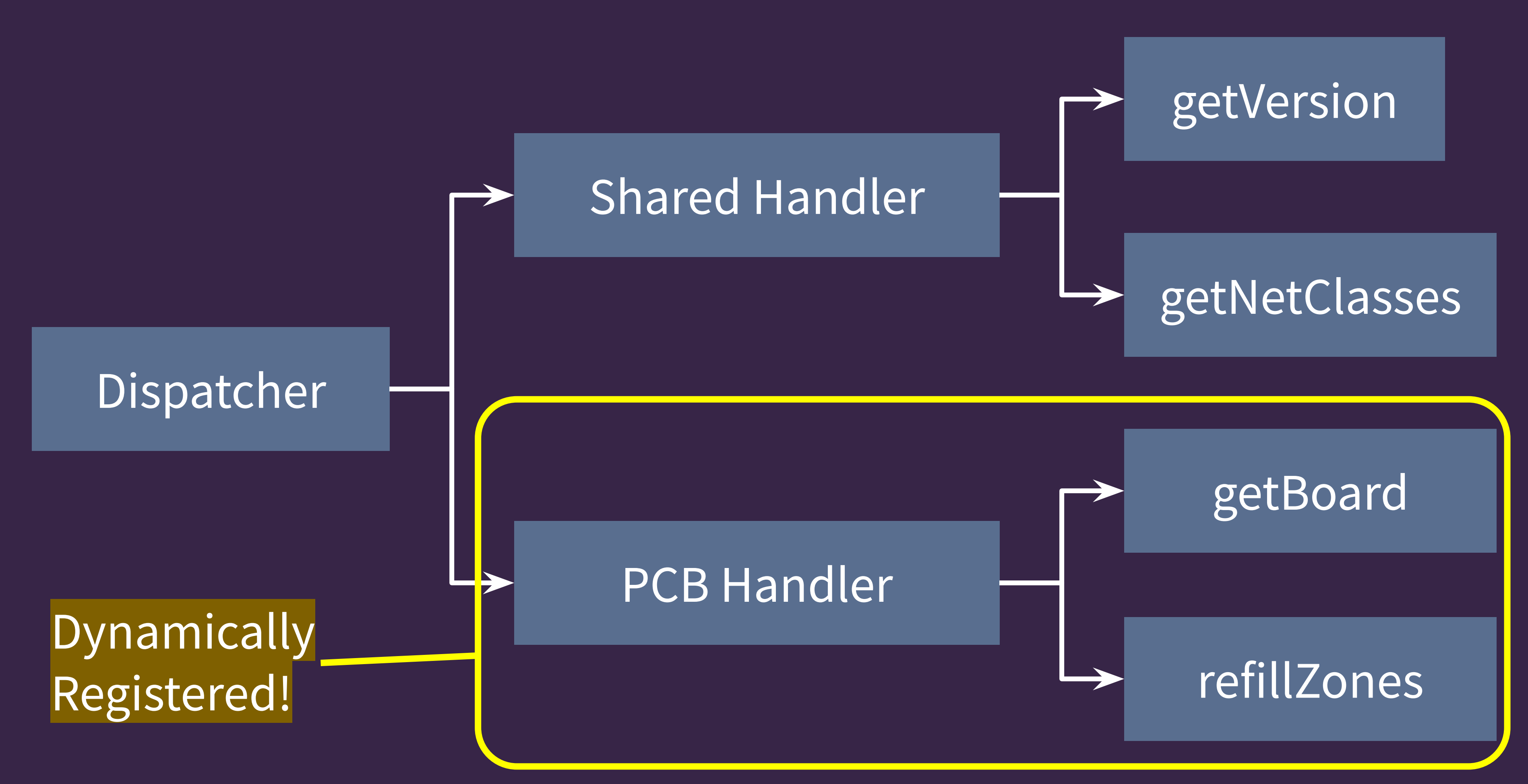
Plumbing it in to KiCad

KiCad’s Event Handling, basically

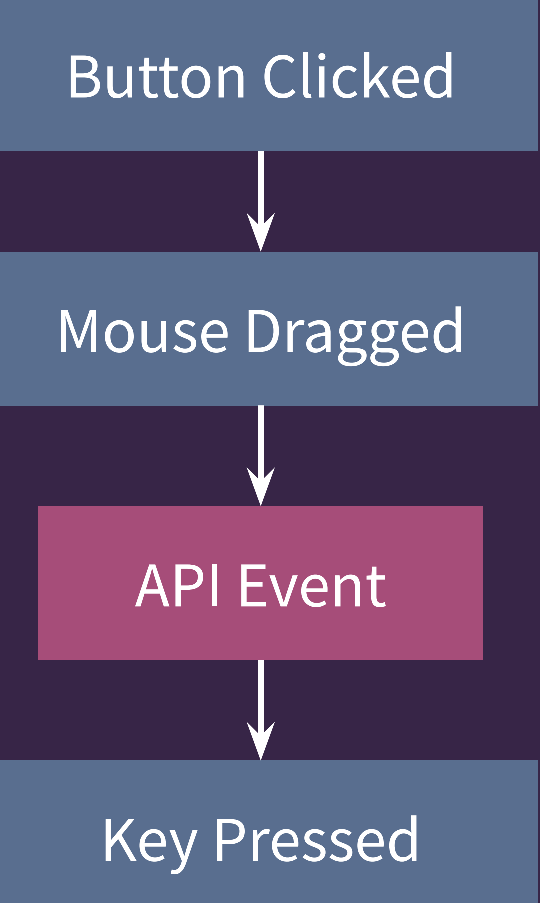



API Event
Key Event
Mouse Event
Menu Event
Dispatcher
API Dispatcher
Event Handler
Event Handler
Event Handler


Dispatcher
Shared Handler
PCB Handler
getVersion
getNetClasses
getBoard
refillZones


Dispatcher
Shared Handler
getVersion
getNetClasses
PCB Handler
getBoard
refillZones
Dynamically Registered!

API Events Are Synchronous!

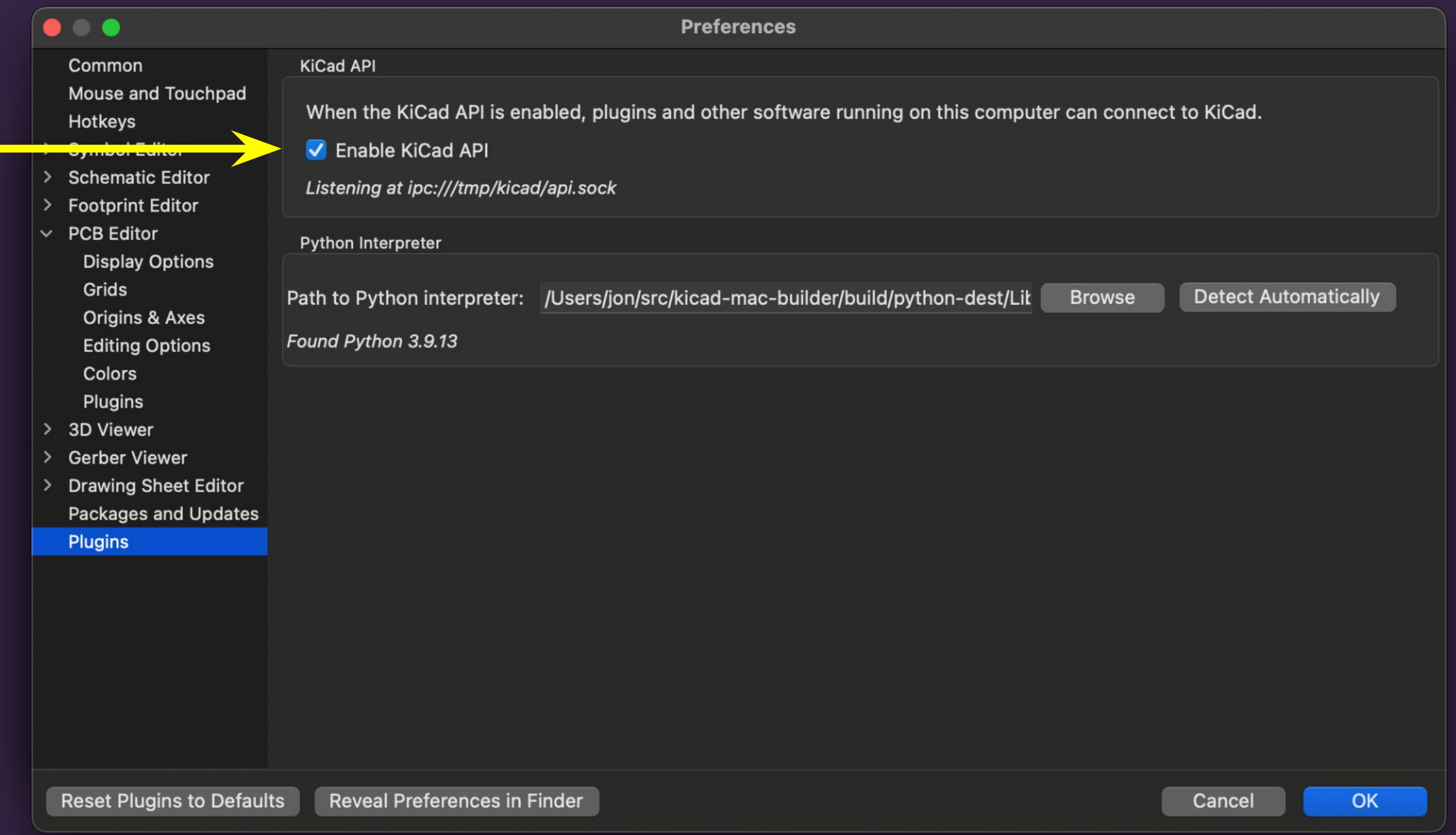


- Plugins can control their own undo/redo transactions
- The API will sometimes reject a command and say “I’m busy”

Using the API from Python

```
$ pip install kicad-python
```

https://gitlab.com/kicad/code/kicad-python


Preferences
Common
Mouse and Touchpad
Hotkeys
Schematic Editor
Footprint Editor
PCB Editor
Display Options
Grids
Origins & Axes
Editing Options
Colors
Plugins
3D Viewer
Gerber Viewer
Drawing Sheet Editor
Packages and Updates
Plugins
KiCad API
When the KiCad API is enabled, plugins and other software running on this computer can connect to KiCad.
Enable KiCad API
Listening at ipc:///tmp/kicad/api.sock
Python Interpreter
Path to Python interpreter:
/Users/jon/src/kicad-mac-builder/build/python-dest/Lit
Browse
Detect Automatically
Found Python 3.9.13
Reset Plugins to Defaults
Reveal Preferences in Finder
Cancel
OK

```python
from kipy import KiCad
from kipy.board_types import (
    BoardLayer,
    Zone
)
from kipy.common_types import PolygonWithHoles
from kipy.geometry import PolyLine, PolyLineNode
from kipy.util import from_mm

if __name__=='__main__':
    kicad = KiCad()
    board = kicad.get_board()

    outline = PolyLine()
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(100)))
    outline.append(PolyLineNode.from_xy(from_mm(110), from_mm(100)))
    outline.append(PolyLineNode.from_xy(from_mm(110), from_mm(110)))
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(110)))
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(100)))
    polygon = PolygonWithHoles()
    polygon.outline = outline
    zone = Zone()
    zone.layers = [BoardLayer.BL_F_Cu, BoardLayer.BL_B_Cu]
    zone.outline = polygon
    board.create_items(zone)
```

```python
from kipy import KiCad
from kipy.board_types import (
    BoardLayer,
    Zone
)
from kipy.common_types import PolygonWithHoles
from kipy.geometry import PolyLine, PolyLineNode
from kipy.util import from_mm

if __name__=='__main__':
    kicad = KiCad()
    board = kicad.get_board()

    outline = PolyLine()
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(100)))
    outline.append(PolyLineNode.from_xy(from_mm(110), from_mm(100)))
    outline.append(PolyLineNode.from_xy(from_mm(110), from_mm(110)))
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(110)))
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(100)))
    polygon = PolygonWithHoles()
    polygon.outline = outline
    zone = Zone()
    zone.layers = [BoardLayer.BL_F_Cu, BoardLayer.BL_B_Cu]
    zone.outline = polygon
    board.create_items(zone)
```

```python
from kipy import KiCad
from kipy.board_types import (
    BoardLayer,
    Zone
)
from kipy.common_types import PolygonWithHoles
from kipy.geometry import PolyLine, PolyLineNode
from kipy.util import from_mm

if __name__=='__main__':
    kicad = KiCad()
    board = kicad.get_board()

    outline = PolyLine()
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(100)))
    outline.append(PolyLineNode.from_xy(from_mm(110), from_mm(100)))
    outline.append(PolyLineNode.from_xy(from_mm(110), from_mm(110)))
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(110)))
    outline.append(PolyLineNode.from_xy(from_mm(100), from_mm(100)))
    polygon = PolygonWithHoles()
    polygon.outline = outline
    zone = Zone()
    zone.layers = [BoardLayer.BL_F_Cu, BoardLayer.BL_B_Cu]
    zone.outline = polygon
    board.create_items(zone)
```

🤔

New Python plugin launching system

- Virtual environments per-plugin
- Automatic* dependency installation
- KiCad tells the plugin how to connect to the API
- *Bonus*: also supports non-Python plugins!

*as long as binary wheels for your platform are available 😅

New Python plugin launching system

- Virtual environments per-plugin
- Automatic* dependency installation
- KiCad tells the plugin how to connect to the API
- *Bonus*: also supports non-Python plugins!

Roadmap

- Footprint editor integration + wizards
- Plotting/exporting (use `kicad-cli` today)
- Headless operations
- Schematic and symbols
- Your idea here?

https://dev-docs.kicad.org

Help Sponsors Log in Register

Search projects

kicad-python 0.1.2

```
pip install kicad-python
```

Latest version

Released: Jan 17, 2025

KiCad API Python Bindings

Navigation

Project description

Release history

Download files

Verified details

These details have been verified by PyPI

Maintainers

craftyjon

Project description

KiCad API Python Bindings

`kicad-python` is the official Python bindings for the KiCad IPC API. This library makes it possible to develop scripts and tools that interact with a running KiCad session.

The KiCad IPC API can be considered in "public beta" state with the release of KiCad 9 (currently planned for on or around February 1, 2025). The existing SWIG-based Python bindings for KiCad's PCB editor still exist in KiCad 9, but are in maintenance mode and will not be expanded.

For more information about the IPC API, please see the KiCad developer documentation. Specific documentation for developing add-ons is also available.

> *Note: Version 0.0.2 and prior of this package are an obsolete earlier effort and are unrelated to this codebase.*

Questions?

jon@craftyjon.com