Code Organization and Rules

This document is currently an unorganized scratch pad of thoughts on what the future pipeline should look like. Discussions are very welcome.

The intention of the core pipeline design is to lower developer friction when developing, easy to deploy, and support arbitrary front ends (Blender, Godot, Krita, small CLIs, etc).

Rules

The rules should be followed unless it makes more sense to not follow them. They are there to make it easier to reason about the code and where code should live.

- API first: All functionality should be independent of UI, and the UI code should only handle displaying and manipulating the data that gets passed to/from the APIs. This is to allow for greater reuse of functionality across the pipeline. Blender's operators are considered UIs in this context. Functionality that have to be in the lower API includes business logic such as communicating with a database, creating meshes, modifying a scene, etc. Functionality that can live in the UI include transforming the data into a human readable format (unless it is a very generic task that will likely be shared such as converting a number into a distance, size, temperature, etc with units), extracting information from a Blender context to pass into the API, etc.
- Invalid states should be impossible: If a tool expects a certain state, and that state is invalid, then that tool must not allow the user to progress until the state is valid. For example, if a publish requires a valid object or collection to be selected, and for comments, then the publish button must be disabled until the conditions are valid. The API should also not assume the data is valid and validate/throw an error. We could use the new type pattern (See https://doc.rust-lang.org/rust-by-example/generics/new_types.html for an example) to do the validations up front and encode them in a class that must remain valid at all times.
- Code should live in their appropriate place: Generic code that does not interact with applications should live in the core, where application specific code should live in the application areas. For example, the generic publish code should live in the core pipeline, where application specific publishing such as rigs, animation, geometry, etc should live in the application area. Also, the libraries are split between the absolute core such as path resolution, and more specific such as turntable generation and management.
- Static where possible, dynamic when necessary: Whenever possible, the code should be static. This could mean functionality, environment resolution, static types, etc. Static code is easier to manage. Code should only be dynamic when static could not solve the issue better than dynamic.

- Simple: We should strive for simplicity both for lower maintenance work and easier reuse.

Project Structure

```
/pipeline/
  blender/
    src/
      operators/
      ui/
      pipeblend/
        core/
        {tool_lib}/
      __init__.py
  core/
    src/
      pipecore/
        core/
        {tool_lib}/
      __init__.py
```

Revision #2
Created 2026-03-14 12:46:28 PDT by propersquid
Updated 2026-03-14 22:55:07 PDT by propersquid