Future Plans: Understanding Pages & MetaTree Editor

Overview

Two major enhancements planned after Code Explorer work is complete:

1. **Integrate Code Explorer** as the unifying example throughout Understanding pages
2. **Build MetaTree Editor** - a WYSIWYG visualization coding platform

---

Plan 1: Code Explorer Integration into Understanding Pages

Goal

Use Code Explorer as the running example across all Understanding pages, making abstract concepts concrete and observable.

Per-Page Integration

Grammar Page (`/understanding/grammar`)

Concepts to illustrate:

- `select` - Entry point: `select "#code-explorer-viz"`
- `appendChild` - Structure building: SVG → zoomGroup → nodesGroup/linksGroup
- `joinData` - Node/link creation from module dependency data
- `setAttrs` - Positioning, coloring, sizing

Code snippets to extract from:

- `src/website/Viz/Spago/Draw.purs` - Main drawing logic
- `src/website/Viz/Spago/Render.purs` - Rendering callbacks

Interactive element:

- "Open Code Explorer and inspect the SVG structure in DevTools to see this hierarchy"

Attributes Page (`/understanding/attributes`)

Concepts to illustrate:

- Static values: `radius 5.0`, `strokeWidth 1.0`
- Datum functions: `cx (\n -> n.x)`, `cy (\n -> n.y)`
- Datum+index: Position calculations for initial layout
- Contravariant: `cmap` for extracting fields

Code snippets to extract:

- Node circle attributes (position from simulation)

- Link line attributes (source/target positions)
- Color mapping from module category
- Text label positioning

Interactive element:

- "Drag a node in Code Explorer - watch how `cx` and `cy` update from the datum's simulation coordinates"

Selections Page (`/understanding/selections`)

Concepts to illustrate:

- State machine in action during expand/collapse
- Enter selection: New child modules appearing
- Update selection: Existing nodes repositioning
- Exit selection: Collapsed nodes animating out

Code snippets to extract:

- `src/website/Component/CodeExplorer/Scenes.purs` - Scene transitions
- Update function showing joinData and handling enter/update/exit

Interactive element:

- "Click a module node to expand it. Watch the console log showing enter/update/exit counts"
- Add debug logging option to visualize state transitions

TreeAPI Page (`/understanding/tree-api`)

Concepts to illustrate:

- Layer cake: Code Explorer uses both TreeAPI (structure) and Simulation API (physics)
- Static structure via TreeAPI
- Dynamic updates via Selection API
- Why this hybrid approach is necessary

Code snippets to extract:

- Initial SVG structure setup
- Where TreeAPI ends and Selection API begins
- Force simulation tick function

Interactive element:

- Show the initial tree structure vs. the dynamic updates
- "The outer shell is declarative, the inner updates are imperative"

Scenes Page (`/understanding/scenes`)

Concepts to illustrate (already partially done):

- Scene data structure for Code Explorer
- Transition matrix between states (collapsed → expanded → focused)
- How user actions trigger scene changes
- Declarative state management

Code snippets to extract:

- Scene type definition
- Transition configurations
- State update handlers

Interactive element:

- "The 'Focus' button demonstrates a scene transition with coordinated animations"

Implementation Tasks

1. **Extract code snippets** - Create a snippets file with annotated Code Explorer code
2. **Add "Try it" sections** - Links to live Code Explorer with specific instructions
3. **Add console logging** - Optional debug mode showing state transitions
4. **Create annotations** - Mermaid diagrams showing Code Explorer-specific flows
5. **Cross-link pages** - Each page links to relevant Code Explorer behavior

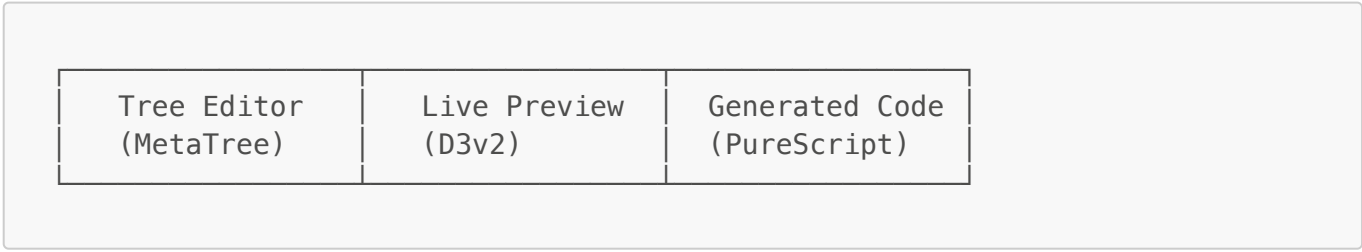
---

Plan 2: MetaTree Editor - WYSIWYG Visualization Platform

Vision

A self-hosted, dog-fooding application that uses an interactive metatree as a visual programming interface for PSD3 visualizations. Like CodePen but for data visualization, showing "how the sausage is made."

Core Layout



User manipulates tree → Preview updates instantly → Code is generated

Key Features

1. Interactive MetaTree Editor

- Visual tree structure (collapsible nodes)
- Drag-and-drop to reorder/reparent

- Click to select and edit properties
- Add/remove nodes via context menu
- Shows: element type, name, key attributes

2. Property Panel

- When a node is selected, show editable properties
- Element type dropdown (SVG, Group, Circle, Rect, Path, Text, etc.)
- Attribute editor (add/remove/edit attributes)
- Data binding configuration
- For Join nodes: data source, template, key function

3. Live Preview Pane

- Renders the visualization in real-time
- Updates on every tree change
- Shows actual D3v2 output
- Zoom/pan controls
- Element inspector (hover to highlight in tree)

4. Code Generation Pane

- Generates PureScript code from tree
- Syntax highlighted
- Copy to clipboard
- Shows both TreeAPI and Selection API versions
- Annotated with comments

5. Data Panel

- Edit sample data (JSON/records)
- Data flows into Join nodes
- See how data changes affect visualization

Data Type Matching

Key educational feature: Show how data structure must match visualization type.

Offer different dataset types:

- **Graph data** - `{ nodes: [...], links: [...] }` → Force layouts, chord diagrams
- **Tabular data** - `Array { x, y, ... }` → Scatter plots, bar charts, line charts
- **Hierarchical data** - `Tree { name, children, value }` → Tree layouts, treemaps, sunbursts
- **Flow data** - `{ source, target, value }` → Sankey diagrams

When user selects a visualization type, guide them to the right data structure. Show errors when data doesn't match (e.g., trying to use tabular data for a tree layout).

try.purescript.org Backend Integration

For more advanced usage, integrate with a PureScript compilation backend (like try.purescript.org uses):

- **Simple mode**: Visual attribute editing with dropdowns
- **Advanced mode**: Write actual PureScript lambdas for attributes

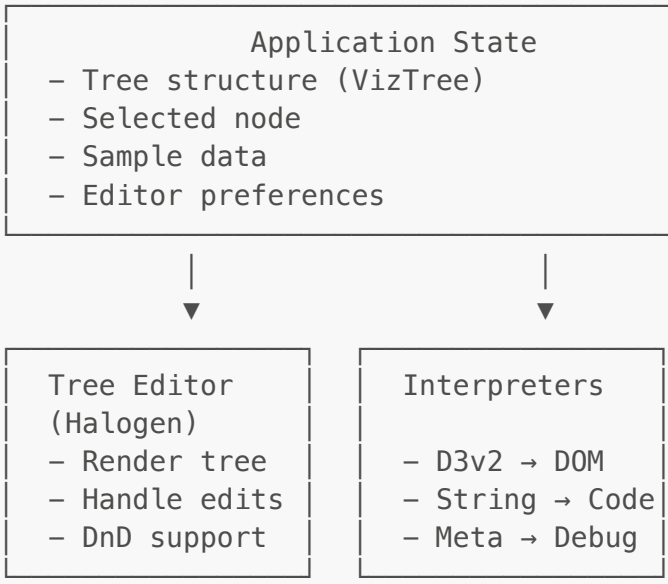
```
cx (\d -> d.x * scale + margin.left)
fill (\d -> if d.value > threshold then "red" else "blue")
```

- Backend compiles and validates the expressions
- Type errors shown inline
- Full power of PureScript available

This would allow:

- Complex conditional logic
- Mathematical transformations
- String formatting
- Pattern matching

Architecture



Implementation Phases

Phase 1: Basic Tree Editor

- Render VizTree as interactive tree view
- Select nodes to see properties
- Add/remove nodes
- Edit basic attributes (strings, numbers)

Phase 2: Live Preview

- Run D3v2 interpreter on tree
- Display in preview pane
- Auto-update on changes

Phase 3: Property Editing

- Full attribute editor
- Element type switching
- Simple datum function builder (dropdown patterns)

Phase 4: Code Generation

- Generate TreeAPI PureScript code
- Proper indentation and formatting
- Copy/export functionality

Phase 5: Data Binding & Type Matching

- Data editor panel
- Connect data to Join nodes
- Preview with different data sets
- Guide users to correct data structures for visualization types

Phase 6: Advanced Lambda Editor

- Text input for PureScript expressions
- Integration with try.purescript.org backend
- Type checking and error display
- Autocomplete for common patterns

Phase 7: Polish

- Drag-and-drop reordering
- Undo/redo
- Save/load projects
- Template gallery (start from working examples)

Example Workflow

1. User starts with empty SVG node
2. Adds a Group child for "chart area"
3. Adds a Join node, connects to sample data `[{x:10,y:20}, {x:30,y:40}]`
4. Sets Join to create Circles
5. Edits Circle template: `cx = _.x`, `cy = _.y`, `radius = 5`
6. Preview shows two circles at (10,20) and (30,40)
7. User switches to advanced mode, writes: `radius (\d -> sqrt d.value * 2)`

8. Backend compiles, preview updates with sized circles
9. User exports generated PureScript code
10. Code works directly in their PSD3 project

Dog-fooding Benefits

1. **Validates the API** - If it's hard to build in the editor, the API might need work
2. **Generates examples** - Users can export their creations as example code
3. **Teaching tool** - Shows the relationship between tree structure and output
4. **Debugging aid** - Visualize what's being built
5. **Data literacy** - Teaches which data structures suit which visualizations

Technical Considerations

Tree Representation

Use the existing `PSD3v2.VizTree.Tree` type but with additional metadata for the editor:

```
type EditorNode =
  { tree :: Tree EditorDatum
  , id :: NodeId
  , expanded :: Boolean
  , selected :: Boolean
  }
```

Synchronization

- Tree changes → immediate re-render
- Debounce rapid changes (typing in attribute fields)
- Optimistic updates with error recovery

Code Generation

Create a new interpreter that generates PureScript source:

```
-- Input tree
named SVG "svg" [width 800, height 600] `withChild`
  joinData "circles" "circle" [1,2,3] $ \d ->
    elem Circle [cx (d * 100), cy 100, radius 20]

-- Generated code
tree :: Tree Number
tree =
  T.named SVG "svg"
    [ width 800.0
    , height 600.0
    ] `T.withChild`
    T.joinData "circles" "circle" [1.0, 2.0, 3.0] $ \d ->
```

```
        T.elem Circle
          [ cx (d * 100.0)
          , cy 100.0
          , radius 20.0
          ]
```

Open Questions

- How to handle complex datum functions? (Full expression editor vs. templates)
- Should it support Selection API or just TreeAPI?
- How to handle behaviors (zoom, drag)?
- Local storage vs. backend for saving projects?
- Should it connect to actual data sources (CSV, JSON URLs)?
- How much of try.purescript.org's backend can we reuse?

Timeline

1. **Now**: Complete Understanding pages (done)
2. **Next few days**: Work on Code Explorer improvements
3. **After Code Explorer**: Integrate Code Explorer examples into Understanding pages
4. **Future**: Build MetaTree Editor platform

Related Files

- Understanding pages: `src/website/Component/Understanding/`
- Code Explorer: `src/website/Component/CodeExplorer/`
- VizTree types: `src/lib/PSD3v2/VizTree/Tree.purs`
- Interpreters: `src/lib/PSD3v2/Interpreter/`