This document is a work in progress throughout the refactoring process, which is happening on this branch. The final documentation will be modified to remove the additional styling.

I just wanted to have this in dark mode because it makes my eyes hurty 😓

— TimeTravelPenguin 🐧

DSL Module

- mk-node()
- is-node()
- is-nodetype()
- assert-nodetype()
- declare-schema-set()
- mk-schema-set()
- ctor()
- render()
- Line()

Variables

- default-node-fields
- node-schema

mk-node

Construct a node of a given type with data and optional fields.

Parameters

```
mk-node(
  name: str,
  data: any,
  ..fields: dictionary
) -> dictionary
```

name `str`

The type of node to create.

data `any`

The data to store in the node.

..fields `dictionary`

Additional fields to store in the node.

is-node

Check if a value is a node.

Parameters

```
is-node(node: any) -> boolean
```

node `any`

The value to check.

is-nodetype

Check if a node is of a given type.

Parameters

```
is-nodetype(
  node: any,
  nodetype: str
) -> boolean
```

node `any`

The node to check.

nodetype `str`

The expected node type.

assert-nodetype

Assert that a value is a node of a given type.

Parameters

```
assert-nodetype(
  node: any,
  nodetype: str
)
```

node `any`

The value to check.

nodetype `str`

The expected node type.

declare-schema-set

Define a schema set for a given node type.

A schema serves as a blueprint for creating and rendering nodes of that type.

A schema set includes:

- **typename**: The name of the node type.
- **constructor**: A function to create nodes of this type.
- **renderer**: A function to render nodes of this type.

Parameters

```
declare-schema-set(name: str) -> dictionary
```

name `str`

The name of the node type.

mk-schema-set

Create a schema set for a given node type.

Parameters

```
mk-schema-set(
  name: str,
  ctor: function,
  render: function
) -> dictionary
```

name `str`

The name of the node type.

ctor `function`

The constructor function for the node type.

render `function`

The renderer function for the node type.

ctor

Retrieve the constructor function for a given node type.

Parameters

```
ctor(name: str) -> function
```

name `str`

The name of the node type.

render

Render a node based on its type using the appropriate renderer.

Parameters

```
render(item: dictionary) -> content
```

item `dictionary`

The node to render.

Line

Construct a Line node.

Parameters

```
Line(
  val: any,
  ..args: arguments
) -> dictionary
```

val `any`

The content of the line.

..args `arguments`

Additional fields for the node. Refer to `default-node-fields`.

default-node-fields `dictionary`

Default schema fields for all nodes.

It has the following definition:

```
import "@preview/valkyrie:0.2.2" as z

let default-node-fields = (
   inline: z.boolean(default: false, optional: true),
)
```

node-schema `dictionary`

Schema for nodes. Nodes encapsulate all data for a given AST item.

It has the following definition:

```
import "@preview/valkyrie:0.2.2" as z

let node-schema = z.dictionary(
  (
    nodetype: z.string(optional: false),
```

```
      data: z.any(optional: false),
    )
      + default-node-fields,
  )
```

See `default-node-fields` for additional fields.