typst-ribbons

Complete API Reference

A comprehensive library for creating ribbon diagrams in Typst

Version 0.1.0 (Development)

2025-11-07

Contents

1 Introduction 6
1.1 Installation 6
1.2 Quick Start 6
2 Main Diagram Functions 7
2.1 ribbon-diagram() 7
2.1.1 Signature 7
2.1.2 Parameters 7
2.1.3 Return Value 7
2.1.4 Example 7
2.2 sankey-diagram() 9
2.2.1 Signature 9
2.2.2 Parameters 9
2.2.3 Return Value 9
2.2.4 Example 1: Basic Sankey 9
2.2.5 Example 2: Vertical Layout 9
2.2.6 Example 3: Multiple Edges 9
2.2.7 Example 4: Custom Styled 9
2.3 chord-diagram() 11
2.3.1 Signature 11
2.3.2 Parameters 11
2.3.3 Return Value 11
2.3.4 Example 1: Symmetric Relationships 11
2.3.5 Example 2: Directed Flow 11
2.3.6 Example 3: Matrix Format 11
2.3.7 Example 4: Custom Colors 11
3 Data Formats 13
3.1 Format 1: Adjacency Dictionary (Recommended) 13
3.1.1 Simple Format 13
3.1.2 Example: Basic 13
3.1.3 Example: Multiple Edges 13
3.1.4 Detailed Format with Attributes 13
3.1.5 Example: Per-Edge Styling 13
3.2 Format 2: Adjacency List 14
3.2.1 Signature 14
3.2.2 Example: Basic 14
3.2.3 Example: With Attributes 14
3.3 Format 3: Adjacency Matrix 14
3.3.1 Signature 14
3.3.2 Example 14
4 Layout Functions 15
4.1 layout.auto-linear() 15
4.1.1 Signature 15
4.1.2 Parameters 15
4.1.3 Return Value 16
4.1.4 Example 1: Default 16
4.1.5 Example 2: Vertical with Wide Gaps 16

4.1.6 Example 3: Manual Layers 16
4.1.7 Example 4: Sharp and Straight 16
4.1.8 Example 5: Compact 17
4.2 layout.circular() 18
4.2.1 Signature 18
4.2.2 Parameters 18
4.2.3 Return Value 18
4.2.4 Example 1: Basic 18
4.2.5 Example 2: Larger 18
4.2.6 Example 3: Directed 19
4.2.7 Example 4: Rotated 19
5 Tinter Functions 20
5.1 tinter.default-tinter() 20
5.1.1 Signature 20
5.1.2 Parameters 20
5.1.3 Return Value 20
5.1.4 Example 20
5.2 tinter.layer-tinter() 20
5.2.1 Signature 20
5.2.2 Parameters 20
5.2.3 Return Value 20
5.2.4 Example 1: Default Palette 20
5.2.5 Example 2: Custom Palette 20
5.3 tinter.node-tinter() 21
5.3.1 Signature 21
5.3.2 Parameters 21
5.3.3 Return Value 21
5.3.4 Example 21
5.4 tinter.categorical-tinter() 21
5.4.1 Signature 21
5.4.2 Parameters 21
5.4.3 Return Value 21
5.4.4 Example 21
5.5 tinter.dict-tinter() 22
5.5.1 Signature 22
5.5.2 Parameters 22
5.5.3 Return Value 22
5.5.4 Example 1: Basic 22
5.5.5 Example 2: With Fallback 22
5.5.6 Example 3: RGB Colors 22
6 Color Palettes 24
6.1 palette.default-palette 24
6.2 palette.color-brewer-palette 24
6.2.1 Example 24
6.3 palette.tableau 24
6.3.1 Example 24
6.4 palette.catppuccin 24
6.4.1 Example 24

7 Ribbon Stylizer Functions 25
7.1 ribbon-stylizer.default() 25
7.1.1 Signature 25
7.1.2 Logic 25
7.1.3 Return Value 25
7.1.4 Example 25
7.2 ribbon-stylizer.match-from() 25
7.2.1 Signature 25
7.2.2 Parameters 25
7.2.3 Return Value 25
7.2.4 Example 1: Basic 25
7.2.5 Example 2: With Border 25
7.2.6 Example 3: Vivid 26
7.3 ribbon-stylizer.match-to() 26
7.3.1 Signature 26
7.3.2 Parameters 26
7.3.3 Return Value 26
7.3.4 Example 26
7.4 ribbon-stylizer.gradient-from-to() 26
7.4.1 Signature 26
7.4.2 Parameters 26
7.4.3 Return Value 26
7.4.4 Example 1: Basic 26
7.4.5 Example 2: With Border 26
7.4.6 Example 3: Subtle 27
7.5 ribbon-stylizer.solid-color() 27
7.5.1 Signature 27
7.5.2 Parameters 27
7.5.3 Return Value 27
7.5.4 Example 1: Gray 27
7.5.5 Example 2: Subtle Black 27
8 Label Drawer Functions 28
8.1 label.default-linear-label-drawer() 28
8.1.1 Signature 28
8.1.2 Parameters 28
8.1.3 Return Value 28
8.1.4 Example 1: Default 28
8.1.5 Example 2: Left-Aligned 28
8.1.6 Example 3: Name Only 29
8.1.7 Example 4: Custom Styling 29
8.1.8 Example 5: Center on Nodes 29
8.2 label.default-circular-label-drawer() 29
8.2.1 Signature 29
8.2.2 Parameters 29
8.2.3 Return Value 29
8.2.4 Example 1: Default 30
8.2.5 Example 2: Farther 30
8.2.6 Example 3: Name Only 30

8.2.7 Example 4: No Labels 30
9 Advanced Features 31
9.1 Per-Edge Custom Styling 31
9.1.1 Using Detailed Format 31
9.1.2 Using Dynamic Functions 31
9.2 Complex Example: Energy Flow 31
10 Complete Examples 33
10.1 Example 1: Company Revenue 33
10.2 Example 2: Migration Patterns 33
10.3 Example 3: Vertical Budget 33
11 Troubleshooting 34
11.1 Nodes Overlap 34
11.2 Labels Cut Off 34
11.3 Wrong Colors 34
11.4 Diagram Too Small 34
12 API Quick Reference 35
12.1 Function Signatures 35
12.2 Parameter Defaults 35
13 Credits 35

1 Introduction

typst-ribbons is a powerful library for creating various types of ribbon/flow diagrams in Typst, including:

- **Sankey diagrams** - for visualizing flow and distribution through a system
- **Chord diagrams** - for showing relationships between entities in a circular layout
- Custom ribbon diagrams with flexible layouts and styling

Built on top of cetz, typst-ribbons provides a high-level, declarative API for creating beautiful flow visualizations with minimal code.

1.1 Installation

For development, use local import:

```
#import "src/ribbons.typ": *
```

1.2 Quick Start

```
// Simple Sankey diagram
#sankey-diagram((
  "A": ("B": 10, "C": 5),
  "B": ("D": 8),
  "C": ("D": 7),
))
```

2 Main Diagram Functions

These are the primary functions you'll use to create diagrams.

2.1 ribbon-diagram()

The base function for creating any ribbon diagram. Most users will use `sankey-diagram()` or `chord-diagram()` instead.

2.1.1 Signature

```
ribbon-diagram(
  data,
  aliases: (:),
  categories: (:),
  layout: layout.auto-linear(),
  tinter: tinter.default-tinter(),
  ribbon-stylizer: ribbon-stylizer.default(),
  draw-label: none,
)
```

2.1.2 Parameters

data (various) Input data in one of the supported formats. See Data Formats section.

aliases (dictionary) Map of node IDs to display names. Default: `(:)`

- **Type:** `dictionary<string, string>`
- **Purpose:** Show user-friendly names instead of IDs
- **Example:** `("prod_a": "Product A")`

categories (dictionary) Map of node IDs to category names for categorical coloring. Default: `(:)`

- **Type:** `dictionary<string, string>`
- **Purpose:** Group nodes by category for coloring
- **Example:** `("coal": "fossil", "solar": "renewable")`

layout (function) Layout algorithm function. Default: `layout.auto-linear()`

- **Type:** `(function, function)` - tuple of (layouter, drawer)
- **Purpose:** Controls node positioning and rendering

tinter (function) Color assignment function. Default: `tinter.default-tinter()`

- **Type:** `function: nodes -> nodes`
- **Purpose:** Assigns colors to nodes

ribbon-stylizer (function) Styling function for ribbons. Default: `ribbon-stylizer.default()`

- **Type:** `function: (color, color, string, string, ...) -> dictionary`
- **Purpose:** Defines ribbon appearance (color, borders, etc.)

draw-label (function or none) Label drawing function. Default: `none`

- **Type:** `function | none`
- **Purpose:** If `none`, no labels are drawn

2.1.3 Return Value

Returns: **content** - The rendered diagram

2.1.4 Example

```
#ribbon-diagram(
  (
```

```
    "A": ("B": 10, "C": 5),
    "B": ("D": 8),
  ),
  aliases: (
    "A": "Input",
    "B": "Process",
    "D": "Output",
  ),
)
```

2.2 sankey-diagram()

Creates a Sankey diagram with linear (left-to-right or top-to-bottom) layout.

2.2.1 Signature

```
sankey-diagram(
  data,
  aliases: (:),
  categories: (:),
  layout: layout.auto-linear(),
  tinter: tinter.default-tinter(),
  ribbon-stylizer: ribbon-stylizer.default(),
  draw-label: label.default-linear-label-drawer(),
  ..args
)
```

2.2.2 Parameters

Same as `ribbon-diagram()`, with defaults optimized for Sankey diagrams.

2.2.3 Return Value

Returns: **content** - The rendered Sankey diagram

2.2.4 Example 1: Basic Sankey

```
#sankey-diagram((
  "A": ("B": 5, "C": 3),
  "B": ("D": 2, "E": 4),
  "C": ("D": 3, "E": 4),
))
```

2.2.5 Example 2: Vertical Layout

```
#sankey-diagram(
  (
    "Revenue": ("Gross": 1000, "COGS": 600),
    "Gross": ("Net": 300, "Expenses": 700),
  ),
  layout: layout.auto-linear(vertical: true)
)
```

2.2.6 Example 3: Multiple Edges

```
#sankey-diagram((
  ("A", "B", 2),
  ("A", "B", 3),  // Second edge A -> B
  ("A", "C", 3),
  ("B", "D", 5),
))
```

2.2.7 Example 4: Custom Styled

```
#sankey-diagram(
  (
    "Input": ("Process": 100),
    "Process": ("Output": 80, "Waste": 20),
  ),
  tinter: tinter.dict-tinter((
    "Input": blue,
    "Process": green,
    "Output": purple,
```

```
    "Waste": red,
  )),
  ribbon-stylizer: ribbon-stylizer.gradient-from-to(
    transparency: 50%,
    stroke-width: 0.5pt,
  )
)
```

2.3 chord-diagram()

Creates a circular chord diagram showing relationships between nodes.

2.3.1 Signature

```
chord-diagram(
  data,
  aliases: (:),
  categories: (:),
  layout: layout.circular(),
  tinter: tinter.default-tinter(),
  ribbon-stylizer: ribbon-stylizer.default(),
  draw-label: label.default-circular-label-drawer(),
  ..args
)
```

2.3.2 Parameters

Same as `ribbon-diagram()`, with defaults optimized for chord diagrams.

2.3.3 Return Value

Returns: **content** - The rendered chord diagram

2.3.4 Example 1: Symmetric Relationships

```
#chord-diagram((
  "A": ("A": 100, "B": 50, "C": 30),
  "B": ("A": 50, "B": 80, "C": 40),
  "C": ("A": 30, "B": 40, "C": 60),
))
```

2.3.5 Example 2: Directed Flow

```
#chord-diagram(
  (
    "Export": ("Import": 500),
    "Import": ("Export": 300),
  ),
  layout: layout.circular(directed: true)
)
```

2.3.6 Example 3: Matrix Format

```
#chord-diagram((
  matrix: (
    (100, 50, 30),
    (50, 80, 40),
    (30, 40, 60),
  ),
  ids: ("A", "B", "C")
))
```

2.3.7 Example 4: Custom Colors

```
#chord-diagram(
  (
    "black": ("black": 11975, "blond": 5871, "brown": 8916, "red": 2868),
    "blond": ("black": 1951, "blond": 10048, "brown": 2060, "red": 6171),
    "brown": ("black": 8010, "blond": 16145, "brown": 8090, "red": 8045),
    "red": ("black": 1013, "blond": 990, "brown": 940, "red": 6907)
  ),
```

```
  tinter: tinter.dict-tinter((
    "black": rgb("#000000"),
    "blond": rgb("#ffdd89"),
    "brown": rgb("#957244"),
    "red": rgb("#f26223"),
  ))
)
```

3 Data Formats

typst-ribbons supports three input data formats.

3.1 Format 1: Adjacency Dictionary (Recommended)

A dictionary where keys are node IDs and values define outgoing edges.

3.1.1 Simple Format

```
(
  "source": ("target": size, ...),
  ...
)
```

Type signature: `dictionary<string, dictionary<string, number | array<number>>>`

3.1.2 Example: Basic

```
#sankey-diagram((
  "A": ("B": 10, "C": 5),
  "B": ("D": 8),
  "C": ("D": 7),
))
```

3.1.3 Example: Multiple Edges

```
#sankey-diagram((
  "A": ("B": (3, 5, 2)),  // Three edges from A to B
  "B": ("C": 10),
))
```

3.1.4 Detailed Format with Attributes

```
(
  "source": (
    (to: "target", size: value, styles: ...),
    ...
  ),
)
```

Type signature: `dictionary<string, array<dictionary>>`

3.1.5 Example: Per-Edge Styling

```
#sankey-diagram((
  "A": (
    (to: "B", size: 10, styles: (fill: red.transparentize(80%))),
    (to: "C", size: 5, styles: (fill: blue.transparentize(80%))),
  ),
  "B": ((to: "D", size: 10),),
))
```

3.2 Format 2: Adjacency List

An array of edge tuples.

3.2.1 Signature

```
(
  (from, to, size),
  (from, to, size, attributes),
  ...
)
```

Type signature: `array<(string, string, number, ?dictionary)>`

3.2.2 Example: Basic

```
#sankey-diagram((
  ("A", "B", 10),
  ("A", "C", 5),
  ("B", "D", 8),
  ("C", "D", 7),
))
```

3.2.3 Example: With Attributes

```
#sankey-diagram((
  ("A", "B", 10, (styles: (fill: red.transparentize(80%)))),
  ("A", "C", 5),
  ("B", "D", 10),
))
```

3.3 Format 3: Adjacency Matrix

A dictionary with `matrix` and `ids` keys.

3.3.1 Signature

```
(
  matrix: ((values...), (values...), ...),
  ids: ("node1", "node2", ...)
)
```

Type signature: `(matrix: array<array<number>>, ids: array<string>)`

3.3.2 Example

```
#chord-diagram((
  matrix: (
    (0, 10, 5),
    (0, 0, 8),
    (0, 0, 7),
  ),
  ids: ("A", "B", "D")
))
```

Where `matrix[i][j]` = flow from `ids[i]` to `ids[j]`.

4 Layout Functions

4.1 layout.auto-linear()

Creates left-to-right or top-to-bottom Sankey diagram.

4.1.1 Signature

```
layout.auto-linear(
  layer-gap: 2,
  node-gap: 1.5,
  node-width: 0.25,
  base-node-height: 3,
  min-node-height: 0.1,
  centerize-layer: false,
  vertical: false,
  layers: (:),
  radius: 2pt,
  curve-factor: 0.3,
)
```

4.1.2 Parameters

layer-gap (number) Horizontal space between layers. Default: `2`

- **Type:** `number`
- **Purpose:** Controls spacing in flow direction
- **Typical range:** 1-4

node-gap (number) Minimum vertical space between nodes. Default: `1.5`

- **Type:** `number`
- **Purpose:** Prevents node overlap
- **Typical range:** 0.5-3

node-width (number) Width of node rectangles. Default: `0.25`

- **Type:** `number`
- **Purpose:** Thickness in flow direction
- **Typical range:** 0.1-0.5

base-node-height (number) Height for largest node. Default: `3`

- **Type:** `number`
- **Purpose:** Scales all node heights
- **Typical range:** 2-5

min-node-height (number) Minimum node height. Default: `0.1`

- **Type:** `number`
- **Purpose:** Ensures tiny nodes are visible

centerize-layer (boolean) Center nodes within each layer. Default: `false`

- **Type:** `boolean`
- **Effect:** `true` = symmetrical, `false` = force-directed

vertical (boolean) Use top-to-bottom layout. Default: `false`

- **Type:** `boolean`
- **Effect:** Rotates entire diagram 90°

layers (dictionary) Manual layer assignments. Default: `(:)`

- **Type:** `dictionary<string, integer>`

- **Purpose:** Override automatic layer calculation
- **Example:** `("A": 0, "B": 1, "C": 2)`

radius (length) Corner radius for nodes. Default: `2pt`

- **Type:** `length`
- **Common values:** `0pt` (sharp), `2pt` (subtle), `4pt` (rounded)

curve-factor (number) Ribbon curvature. Default: `0.3`

- **Type:** `number`
- **Range:** 0.0 (straight) to 1.0 (very curved)

4.1.3 Return Value

Returns: `(layouter: function, drawer: function)`

4.1.4 Example 1: Default

```
#sankey-diagram(
  data,
  layout: layout.auto-linear()
)
```

4.1.5 Example 2: Vertical with Wide Gaps

```
#sankey-diagram(
  data,
  layout: layout.auto-linear(
    vertical: true,
    layer-gap: 3,
    node-gap: 2,
  )
)
```

4.1.6 Example 3: Manual Layers

```
#sankey-diagram(
  (
    "A": ("B": 10),
    "B": ("C": 10),
    "X": ("C": 5),
  ),
  layout: layout.auto-linear(
    layers: (
      "A": 0,
      "X": 0,  // Force X to same layer as A
      "B": 1,
      "C": 2,
    )
  )
)
```

4.1.7 Example 4: Sharp and Straight

```
#sankey-diagram(
  data,
  layout: layout.auto-linear(
    radius: 0pt,
    curve-factor: 0,
  )
)
```

4.1.8 Example 5: Compact

```
#sankey-diagram(
  data,
  layout: layout.auto-linear(
    layer-gap: 1,
    node-gap: 0.5,
    node-width: 0.15,
    base-node-height: 2,
  )
)
```

4.2 layout.circular()

Creates circular chord diagram.

4.2.1 Signature

```
layout.circular(
  radius: 4,
  node-width: 0.5,
  node-gap: 1deg,
  angle-offset: 0deg,
  directed: false,
)
```

4.2.2 Parameters

radius (number) Circle radius. Default: `4`

- **Type:** `number`
- **Purpose:** Overall diagram size
- **Typical range:** 3-6

node-width (number) Radial width of node arcs. Default: `0.5`

- **Type:** `number`
- **Purpose:** Thickness of arc segments
- **Typical range:** 0.3-0.8

node-gap (angle) Angular gap between nodes. Default: `1deg`

- **Type:** `angle`
- **Purpose:** Spacing between nodes
- **Common values:** `0.5deg-3deg`

angle-offset (angle) Starting angle for first node. Default: `0deg`

- **Type:** `angle`
- **Purpose:** Rotates entire diagram
- **Common values:** `0deg` (top), `90deg` (right)

directed (boolean) Show directional flow. Default: `false`

- **Type:** `boolean`
- **Effect:** `false` = merge both directions, `true` = show asymmetry

4.2.3 Return Value

Returns: `(layouter: function, drawer: function)`

4.2.4 Example 1: Basic

```
#chord-diagram(
  data,
  layout: layout.circular()
)
```

4.2.5 Example 2: Larger

```
#chord-diagram(
  data,
  layout: layout.circular(
    radius: 6,
    node-width: 0.8,
  )
)
```

4.2.6 Example 3: Directed

```
#chord-diagram(
  (
    "A": ("B": 100, "C": 50),
    "B": ("C": 80, "A": 30),
  ),
  layout: layout.circular(directed: true)
)
```

4.2.7 Example 4: Rotated

```
#chord-diagram(
  data,
  layout: layout.circular(
    angle-offset: 90deg,
    node-gap: 2deg,
  )
)
```

5 Tinter Functions

Tinters assign colors to nodes.

5.1 tinter.default-tinter()

Automatically chooses `layer-tinter()` or `node-tinter()`.

5.1.1 Signature

```
tinter.default-tinter(
  palette: palette.default-palette
)
```

5.1.2 Parameters

palette (array) Array of colors. Default: `palette.default-palette`

- **Type:** `array<color>`

5.1.3 Return Value

Returns: `function: nodes -> nodes`

5.1.4 Example

```
#sankey-diagram(
  data,
  tinter: tinter.default-tinter()
)
```

5.2 tinter.layer-tinter()

Colors nodes by layer index.

5.2.1 Signature

```
tinter.layer-tinter(
  palette: palette.default-palette
)
```

5.2.2 Parameters

palette (array) Array of colors.

- **Type:** `array<color>`
- **Algorithm:** `color[layer_index mod palette.length]`

5.2.3 Return Value

Returns: `function: nodes -> nodes`

5.2.4 Example 1: Default Palette

```
#sankey-diagram(
  data,
  tinter: tinter.layer-tinter()
)
```

5.2.5 Example 2: Custom Palette

```
#sankey-diagram(
  data,
  tinter: tinter.layer-tinter(
    palette: (red, orange, yellow, green, blue)
  )
)
```

5.3 tinter.node-tinter()

Colors each node uniquely by index.

5.3.1 Signature

```
tinter.node-tinter(
  palette: palette.default-palette
)
```

5.3.2 Parameters

palette (array) Array of colors.

- **Type:** `array<color>`
- **Algorithm:** `color[node_index mod palette.length]`

5.3.3 Return Value

Returns: `function: nodes -> nodes`

5.3.4 Example

```
#chord-diagram(
  data,
  tinter: tinter.node-tinter(
    palette: palette.tableau
  )
)
```

5.4 tinter.categorical-tinter()

Colors nodes by category.

5.4.1 Signature

```
tinter.categorical-tinter(
  palette: palette.default-palette
)
```

5.4.2 Parameters

palette (array) Array of colors.

- **Type:** `array<color>`
- **Note:** Requires `categories` parameter in main diagram function

5.4.3 Return Value

Returns: `function: nodes -> nodes`

5.4.4 Example

```
#sankey-diagram(
  (
    "Coal": ("Power": 100),
    "Solar": ("Grid": 50),
    "Power": ("Grid": 100),
    "Grid": ("City": 150),
  ),
  categories: (
    "Coal": "fossil",
    "Solar": "renewable",
    "Power": "processing",
    "Grid": "distribution",
    "City": "consumption",
```

```
  ),
  tinter: tinter.categorical-tinter(
    palette: (gray, green, yellow, blue, purple)
  )
)
```

5.5 tinter.dict-tinter()

Manual color specification.

5.5.1 Signature

```
tinter.dict-tinter(
  color-map,
  override: none
)
```

5.5.2 Parameters

color-map (dictionary) Node ID to color mapping.

- **Type:** `dictionary<string, color>`
- **Purpose:** Explicit color control

override (function or none) Fallback tinter. Default: `none`

- **Type:** `function | none`
- **Purpose:** Colors unspecified nodes

5.5.3 Return Value

Returns: `function: nodes -> nodes`

5.5.4 Example 1: Basic

```
#sankey-diagram(
  data,
  tinter: tinter.dict-tinter((
    "A": red,
    "B": blue,
    "C": green,
  ))
)
```

5.5.5 Example 2: With Fallback

```
#sankey-diagram(
  data,
  tinter: tinter.dict-tinter(
    (
      "Critical": red,
      "Warning": orange,
    ),
    override: tinter.layer-tinter()
  )
)
```

5.5.6 Example 3: RGB Colors

```
#chord-diagram(
  data,
  tinter: tinter.dict-tinter((
    "A": rgb("#FF5733"),
    "B": rgb("#33FF57"),
```

```
    "C": rgb("#3357FF"),
  ))
)
```

6 Color Palettes

Pre-defined color palettes.

6.1 palette.default-palette

Alias for `palette.color-brewer-palette`.

6.2 palette.color-brewer-palette

ColorBrewer Set2 (8 colors).

Colors:

```
(
  rgb("#66C2A5"), rgb("#FC8D62"), rgb("#8DA0CB"), rgb("#E78AC3"),
  rgb("#A6D854"), rgb("#FFD92F"), rgb("#E5C494"), rgb("#B3B3B3")
)
```

6.2.1 Example

```
tinter: tinter.layer-tinter(
  palette: palette.color-brewer-palette
)
```

6.3 palette.tableau

Tableau 10 (10 colors).

Colors:

```
(
  rgb("#1F77B4"), rgb("#FF7F0E"), rgb("#2CA02C"), rgb("#D62728"),
  rgb("#9467BD"), rgb("#8C564B"), rgb("#E377C2"), rgb("#7F7F7F"),
  rgb("#BCBD22"), rgb("#17BECF")
)
```

6.3.1 Example

```
tinter: tinter.node-tinter(
  palette: palette.tableau
)
```

6.4 palette.catppuccin

Catppuccin Frappé (13 colors).

Colors:

```
(
  rgb("#e78284"), rgb("#a6d189"), rgb("#e5c890"), rgb("#8caaee"),
  rgb("#f4b8e4"), rgb("#81c8be"), rgb("#ca9ee6"), rgb("#ea999c"),
  rgb("#85c1dc"), rgb("#ef9f76"), rgb("#99d1db"), rgb("#eebebe"),
  rgb("#f2d5cf")
)
```

6.4.1 Example

```
tinter: tinter.layer-tinter(
  palette: palette.catppuccin
)
```

7 Ribbon Stylizer Functions

Control ribbon appearance.

7.1 ribbon-stylizer.default()

Auto-selects appropriate styling.

7.1.1 Signature

```
ribbon-stylizer.default()
```

7.1.2 Logic

- Chord diagrams: `gradient-from-to()` with white border
- Others: `match-from()` with no border

7.1.3 Return Value

Returns: `function: (...) -> dictionary`

7.1.4 Example

```
ribbon-stylizer: ribbon-stylizer.default()
```

7.2 ribbon-stylizer.match-from()

Ribbons match source node color.

7.2.1 Signature

```
ribbon-stylizer.match-from(
  transparency: 75%,
  stroke-width: 0pt,
  stroke-color: auto,
)
```

7.2.2 Parameters

transparency (percentage) Ribbon transparency. Default: `75%`

- **Type:** `percentage`
- **Range:** `0%` (opaque) to `100%` (invisible)

stroke-width (length) Border width. Default: `0pt`

- **Type:** `length`
- **Common values:** `0pt`, `0.5pt`, `1pt`

stroke-color (color or auto) Border color. Default: `auto`

- **Type:** `color | auto`
- **Auto:** Matches fill color

7.2.3 Return Value

Returns: `function: (from-color, to-color, from-node, to-node, ...) -> dictionary`

Returns dictionary with `fill` and `stroke` keys.

7.2.4 Example 1: Basic

```
ribbon-stylizer: ribbon-stylizer.match-from()
```

7.2.5 Example 2: With Border

```
ribbon-stylizer: ribbon-stylizer.match-from(
  transparency: 60%,
  stroke-width: 0.5pt,
```

```
  stroke-color: white,
)
```

7.2.6 Example 3: Vivid

```
ribbon-stylizer: ribbon-stylizer.match-from(
  transparency: 30%,
)
```

7.3 ribbon-stylizer.match-to()

Ribbons match target node color.

7.3.1 Signature

```
ribbon-stylizer.match-to(
  transparency: 75%,
  stroke-width: 0pt,
  stroke-color: auto,
)
```

7.3.2 Parameters

Same as `match-from()`.

7.3.3 Return Value

Returns: `function: (...) -> dictionary`

7.3.4 Example

```
ribbon-stylizer: ribbon-stylizer.match-to(
  transparency: 70%,
)
```

7.4 ribbon-stylizer.gradient-from-to()

Gradient from source to target color.

7.4.1 Signature

```
ribbon-stylizer.gradient-from-to(
  transparency: 75%,
  stroke-width: 0pt,
  stroke-color: auto,
)
```

7.4.2 Parameters

Same as `match-from()` and `match-to()`.

7.4.3 Return Value

Returns: `function: (...) -> dictionary`

7.4.4 Example 1: Basic

```
ribbon-stylizer: ribbon-stylizer.gradient-from-to()
```

7.4.5 Example 2: With Border

```
ribbon-stylizer: ribbon-stylizer.gradient-from-to(
  transparency: 50%,
  stroke-width: 0.5pt,
  stroke-color: white,
)
```

7.4.6 Example 3: Subtle

```
ribbon-stylizer: ribbon-stylizer.gradient-from-to(
  transparency: 85%,
  stroke-width: 0.2pt,
)
```

7.5 ribbon-stylizer.solid-color()

All ribbons use single color.

7.5.1 Signature

```
ribbon-stylizer.solid-color(
  color: black,
  transparency: 90%,
  stroke-width: 0pt,
  stroke-color: auto,
)
```

7.5.2 Parameters

color (color) Ribbon color. Default: `black`

- **Type:** `color`

transparency (percentage) Transparency. Default: `90%`

- **Type:** `percentage`

stroke-width (length) Border width. Default: `0pt`

- **Type:** `length`

stroke-color (color or auto) Border color. Default: `auto`

- **Type:** `color | auto`

7.5.3 Return Value

Returns: `function: (...) -> dictionary`

7.5.4 Example 1: Gray

```
ribbon-stylizer: ribbon-stylizer.solid-color(
  color: gray,
  transparency: 80%,
)
```

7.5.5 Example 2: Subtle Black

```
ribbon-stylizer: ribbon-stylizer.solid-color(
  color: black,
  transparency: 95%,
  stroke-width: 0.2pt,
)
```

8 Label Drawer Functions

8.1 label.default-linear-label-drawer()

Labels for Sankey diagrams.

8.1.1 Signature

```
label.default-linear-label-drawer(
  snap: auto,
  offset: auto,
  width-limit: auto,
  styles: (
    inset: 0.2em,
    fill: white.transparentize(50%),
    radius: 2pt
  ),
  draw-content: (properties) => { ... }
)
```

8.1.2 Parameters

snap (position or auto) Label position. Default: `auto`

- **Type:** `position | auto`
- **Options:** `left`, `right`, `top`, `bottom`, `center`, `auto`
- **Auto:** `right` for horizontal, `bottom` for vertical

offset (array or auto) Label offset. Default: `auto`

- **Type:** `(number, number) | auto`
- **Auto:** `(0.05, 0)` for right/bottom, `(-0.05, 0)` for left/top

width-limit (length, auto, or false) Max label width. Default: `auto`

- **Type:** `length | auto | false`
- **Auto:** 95% of `layer-gap`
- **False:** No limit

styles (dictionary) Box styling. Default shown above.

- **Type:** `dictionary`
- **Properties:** Any box properties (inset, fill, stroke, radius, etc.)

draw-content (function) Content renderer. Default shows name and size.

- **Type:** `function: properties -> content`
- **Parameter:** `properties` - node properties dictionary
- **Fields:** `name`, `size`, `id`, `color`, etc.

8.1.3 Return Value

Returns: `function: (node-name, properties, ...) -> content`

8.1.4 Example 1: Default

```
draw-label: label.default-linear-label-drawer()
```

8.1.5 Example 2: Left-Aligned

```
draw-label: label.default-linear-label-drawer(
  snap: left,
)
```

8.1.6 Example 3: Name Only

```
draw-label: label.default-linear-label-drawer(
  draw-content: (properties) => {
    text(properties.name, weight: "bold")
  }
)
```

8.1.7 Example 4: Custom Styling

```
draw-label: label.default-linear-label-drawer(
  styles: (
    inset: 0.3em,
    fill: blue.transparentize(80%),
    stroke: blue,
    radius: 4pt,
  ),
  draw-content: (properties) => [
    #set text(fill: blue)
    #text(properties.name, size: 0.9em) \
    #text(str(properties.size), weight: "bold")
  ]
)
```

8.1.8 Example 5: Center on Nodes

```
draw-label: label.default-linear-label-drawer(
  snap: center,
  styles: (fill: white, radius: 0pt),
)
```

8.2 label.default-circular-label-drawer()

Labels for chord diagrams.

8.2.1 Signature

```
label.default-circular-label-drawer(
  offset: 0.2,
  styles: (
    inset: 0.2em,
    fill: white.transparentize(50%),
    radius: 2pt
  ),
  draw-content: (properties) => { ... }
)
```

8.2.2 Parameters

offset (number) Distance from circle edge. Default: `0.2`

- **Type:** `number`
- **Units:** Same as radius

styles (dictionary) Box styling. Same as linear drawer.

- **Type:** `dictionary`

draw-content (function) Content renderer. Same as linear drawer.

- **Type:** `function: properties -> content`

8.2.3 Return Value

Returns: `function: (node-name, properties, ...) -> content`

8.2.4 Example 1: Default

```
draw-label: label.default-circular-label-drawer()
```

8.2.5 Example 2: Farther

```
draw-label: label.default-circular-label-drawer(
  offset: 0.5,
)
```

8.2.6 Example 3: Name Only

```
draw-label: label.default-circular-label-drawer(
  draw-content: (properties) => {
    text(properties.name, size: 0.8em)
  }
)
```

8.2.7 Example 4: No Labels

```
draw-label: none
```

9 Advanced Features

9.1 Per-Edge Custom Styling

9.1.1 Using Detailed Format

```
#sankey-diagram((
  "A": (
    (
      to: "B",
      size: 10,
      styles: (fill: red.transparentize(80%))
    ),
    (
      to: "C",
      size: 5,
      styles: (fill: blue.transparentize(80%))
    ),
  ),
  "B": ((to: "D", size: 10),),
))
```

9.1.2 Using Dynamic Functions

```
#sankey-diagram((
  "A": (
    (
      to: "B",
      size: 10,
      styles: (edge, from-props, to-id, to-props) => {
        if edge.size > 5 {
          (fill: red.transparentize(70%))
        } else {
          (fill: blue.transparentize(70%))
        }
      }
    ),
  ),
))
```

9.2 Complex Example: Energy Flow

```
#sankey-diagram(
  (
    "Solar": ("Battery": 100, "Grid": 50),
    "Wind": ("Battery": 80, "Grid": 60),
    "Battery": ("Home": 150, "Industry": 30),
    "Grid": ("Home": 80, "Industry": 30),
  ),
  aliases: (
    "Solar": "Solar Panels",
    "Wind": "Wind Turbines",
    "Battery": "Battery Storage",
    "Grid": "Power Grid",
    "Home": "Residential",
    "Industry": "Industrial",
  ),
  categories: (
    "Solar": "source",
```

```
    "Wind": "source",
    "Battery": "storage",
    "Grid": "distribution",
    "Home": "consumption",
    "Industry": "consumption",
  ),
  layout: layout.auto-linear(
    vertical: true,
    layer-gap: 3,
    curve-factor: 0.4,
  ),
  tinter: tinter.categorical-tinter(
    palette: (green, yellow, blue, purple)
  ),
  ribbon-stylizer: ribbon-stylizer.gradient-from-to(
    transparency: 60%,
    stroke-width: 0.5pt,
    stroke-color: white,
  ),
  draw-label: label.default-linear-label-drawer(
    snap: right,
    width-limit: 2.5cm,
    draw-content: (p) => [
      #text(p.name, weight: "bold", size: 0.85em) \
      #text(str(p.size) + " MW", size: 0.75em)
    ]
  )
)
```

10 Complete Examples

10.1 Example 1: Company Revenue

```
#sankey-diagram(
  (
    "Products": ("Revenue": 10000),
    "Services": ("Revenue": 5000),
    "Revenue": ("Operating": 8000, "Profit": 7000),
    "Operating": ("Salaries": 5000, "Marketing": 2000, "Other": 1000),
  ),
  tinter: tinter.layer-tinter(
    palette: (blue, green, yellow, orange)
  ),
)
```

10.2 Example 2: Migration Patterns

```
#chord-diagram(
  (
    "Asia": ("Asia": 50000, "Europe": 5000, "Americas": 8000),
    "Europe": ("Asia": 4000, "Europe": 30000, "Americas": 6000),
    "Americas": ("Asia": 3000, "Europe": 5000, "Americas": 40000),
  ),
  tinter: tinter.node-tinter(palette: palette.tableau),
  ribbon-stylizer: ribbon-stylizer.gradient-from-to(
    transparency: 80%,
    stroke-width: 0.5pt,
    stroke-color: white,
  ),
)
```

10.3 Example 3: Vertical Budget

```
#sankey-diagram(
  (
    "Budget": ("Dev": 500, "Marketing": 300, "Ops": 200),
    "Dev": ("Salaries": 400, "Tools": 100),
    "Marketing": ("Ads": 200, "Events": 100),
    "Ops": ("Rent": 100, "Other": 100),
  ),
  layout: layout.auto-linear(
    vertical: true,
    layer-gap: 2.5,
  ),
)
```

11 Troubleshooting

11.1 Nodes Overlap

Solution: Increase `node-gap`

```
layout: layout.auto-linear(node-gap: 2.5)
```

11.2 Labels Cut Off

Solutions:

```
// Option 1: Wider layers
layout: layout.auto-linear(layer-gap: 3)

// Option 2: Limit width
draw-label: label.default-linear-label-drawer(
  width-limit: 2cm
)

// Option 3: Shorter text
draw-label: label.default-linear-label-drawer(
  draw-content: (p) => text(p.name, size: 0.7em)
)
```

11.3 Wrong Colors

Solution: Use explicit colors

```
tinter: tinter.dict-tinter((
  "A": red,
  "B": blue,
))
```

11.4 Diagram Too Small

Solutions:

```
// Sankey
layout: layout.auto-linear(
  layer-gap: 3,
  base-node-height: 4,
)

// Chord
layout: layout.circular(radius: 6)
```

12 API Quick Reference

12.1 Function Signatures

```
// Main functions
sankey-diagram(data, aliases, categories, layout, tinter, ribbon-stylizer, draw-label) ->
content
chord-diagram(data, aliases, categories, layout, tinter, ribbon-stylizer, draw-label) ->
content

// Layouts
layout.auto-linear(layer-gap, node-gap, node-width, base-node-height, min-node-height,
                   centerize-layer, vertical, layers, radius, curve-factor) -> (function,
function)
layout.circular(radius, node-width, node-gap, angle-offset, directed) -> (function,
function)

// Tinters
tinter.default-tinter(palette) -> function
tinter.layer-tinter(palette) -> function
tinter.node-tinter(palette) -> function
tinter.categorical-tinter(palette) -> function
tinter.dict-tinter(color-map, override) -> function

// Stylizers
ribbon-stylizer.match-from(transparency, stroke-width, stroke-color) -> function
ribbon-stylizer.match-to(transparency, stroke-width, stroke-color) -> function
ribbon-stylizer.gradient-from-to(transparency, stroke-width, stroke-color) -> function
ribbon-stylizer.solid-color(color, transparency, stroke-width, stroke-color) -> function

// Labels
label.default-linear-label-drawer(snap, offset, width-limit, styles, draw-content) ->
function
label.default-circular-label-drawer(offset, styles, draw-content) -> function
```

12.2 Parameter Defaults

Function	Parameter	Default
auto-linear	layer-gap	2
	node-gap	1.5
	vertical	false
circular	radius	4
	directed	false
match-from	transparency	75%
	stroke-width	0pt
linear-label	snap	auto
	width-limit	auto

13 Credits

- Built with CeTZ
- Color palettes: ColorBrewer, Tableau, Catppuccin
- Demo data: SankeyMatic, D3.js examples

End of API Reference

typst-ribbons v0.1.0