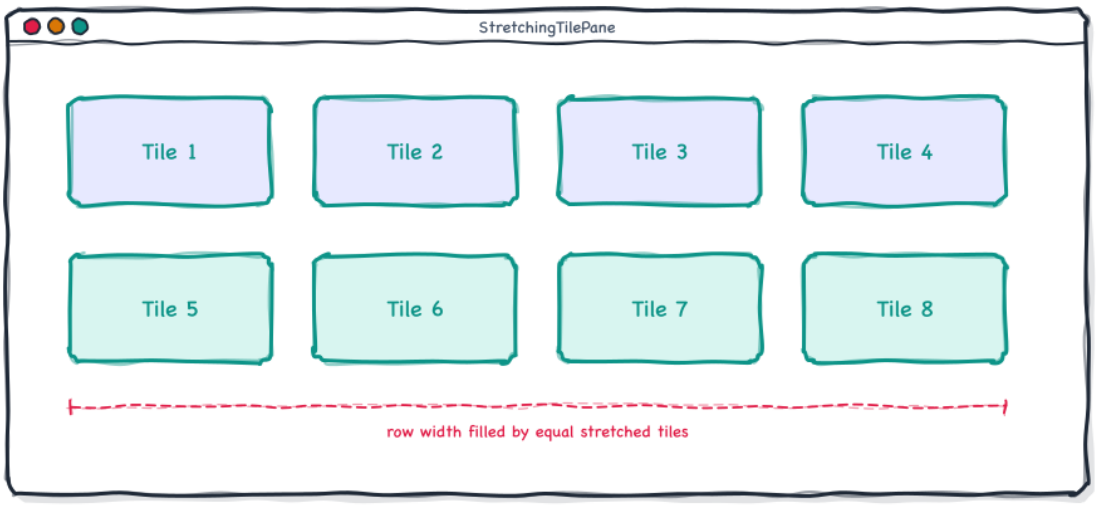
StretchingTilePane

Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls



A generated cartoon overview of StretchingTilePane.

StretchingTilePane lays out managed Region children in rows and stretches every tile in a row to the same width so the row fills the pane.

Table of Contents

1. Introduction

StretchingTilePane arranges managed Region children in rows. It computes how many preferred-width tiles fit, subtracts gaps and stretches each tile so the row fills all available width.

1.1 Key features

- Horizontal content bias: preferred height depends on width.
- Column count comes from maximum preferred tile width plus hgap.
- Tiles share one stretched width and the maximum preferred height.
- Managed children only participate.
- hgap and vgap are styleable CSS properties.

1.2 Maven dependency

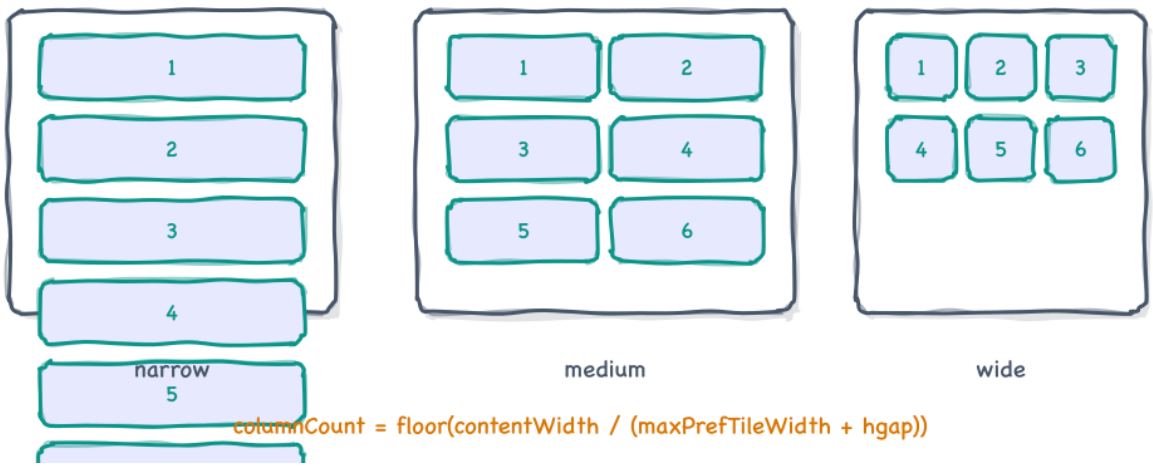
```
<dependency>
    <groupId>com.dlsc.gemsfx</groupId>
    <artifactId>gemsfx</artifactId>
    <version>4.4.1</version>
</dependency>
```

Maven coordinates for the GemsFX control library.

2. Getting started

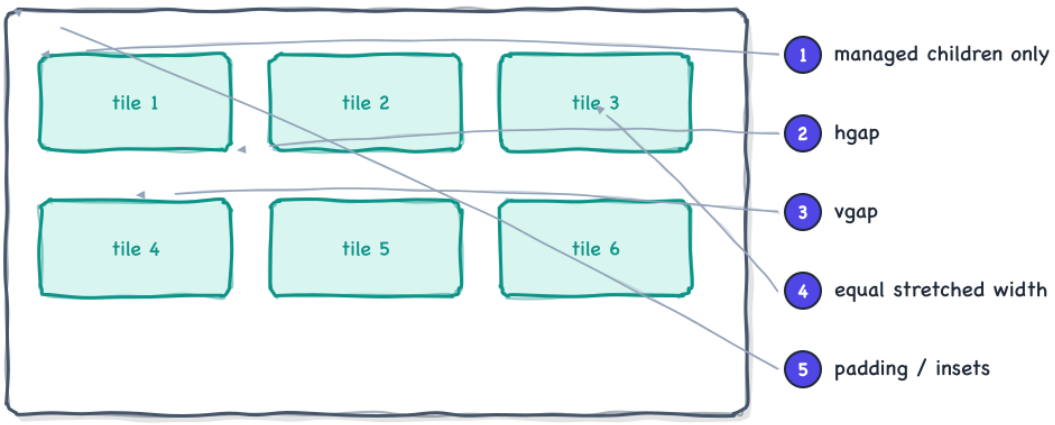
```
StretchingTilePane pane = new StretchingTilePane(10, 10);
pane.setPadding(new Insets(20));
for (int i = 0; i < 12; i++) {
    Label tile = new Label("Tile " + (i + 1));
    tile.setPrefSize(150, 100);
    pane.getChildren().add(tile);
}
```

A row-stretching tile grid.



Column count changes with available width.

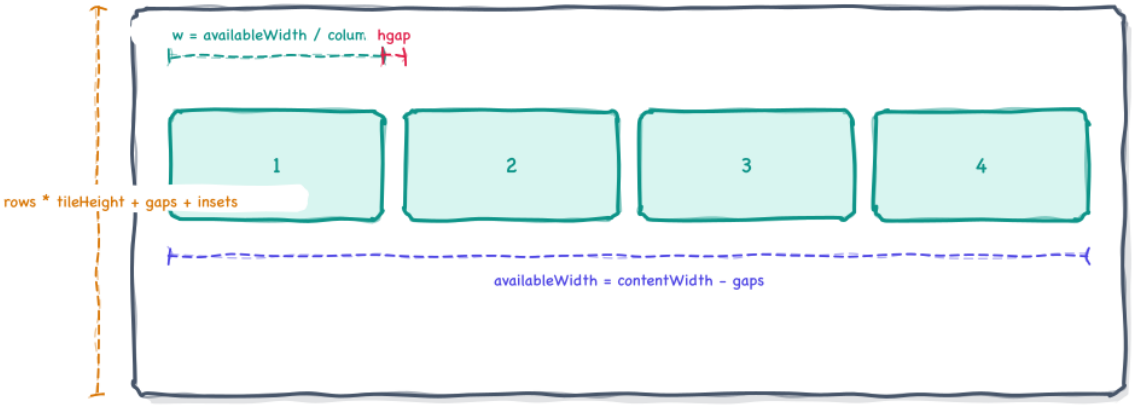
3. Anatomy



Insets, gaps and managed tile children.

Property	Type	Default	Description
hgap	DoubleProperty	0	Horizontal gap; styleable with -fx-hgap.
vgap	DoubleProperty	0	Vertical gap; styleable with -fx-vgap.
contentBias	Orientation	HORIZONTAL	Reports horizontal content bias.

4. Column geometry

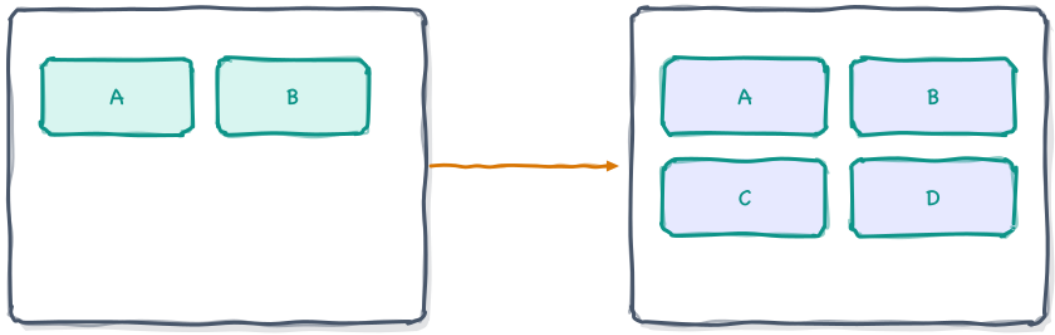


The row width is divided after gaps are subtracted.

The pane finds maximum preferred tile width, then computes `columnCount = (int) (contentWidth / (preferredTileWidth + hgap))`. If the count is zero, layout returns without placing children.

```
availableWidth = contentWidth - (columnCount - 1) * hgap
tileWidth = availableWidth / columnCount
tileHeight = max(child.prefHeight(contentWidth))
```

5. Preferred height



Layout is recomputed when width changes.

Preferred height is rows times maximum preferred tile height plus vertical gaps and insets. Minimum height delegates to preferred height.

Input	Effect
Wider width	May increase columns and reduce rows.
Larger hgap	Can reduce column count.
Larger vgap	Increases preferred height between rows.

6. Styling

No user agent stylesheet or root style class is added, but hgap and vgap are styleable.

CSS property	Type	Default
`-fx-hgap`	number	0
`-fx-vgap`	number	0

```
.stretching-tile-pane {
    -fx-hgap: 12;
    -fx-vgap: 12;
}
```

7. Managed children and Region assumptions

The layout loop uses `getManagedChildren()`. Unmanaged children remain in the scene graph but are ignored by column count, preferred height and placement.

The preferred tile height computation casts each managed child to `Region`. In normal use tiles are Labels, panes or controls, all of which are Region subclasses. Wrap non-Region nodes before adding them.

Child kind	Recommendation
Label / Button / Control	Safe: they are Region subclasses.
Pane / StackPane / VBox	Safe and useful for card tiles.
Canvas / Group / Shape	Wrap in StackPane before adding.
Temporarily hidden item	Set managed false as well as visible false.

> **Careful.** A managed non-Region child can fail during preferred-height computation because of the source cast.

8. Edge cases and zero columns

When available width is smaller than one preferred tile width plus hgap, `columnCount` becomes zero. Both preferred-height computation and layout return zero / do nothing for that pass.

Case	Source behaviour
No managed children	Preferred height is 0 and layout returns.
columnCount == 0	Preferred height is 0 and layout returns.
Negative hgap or vgap	Accepted by the property; the source does not clamp it.
Insets larger than width	contentWidth can become small or negative, leading to zero columns.

For predictable layout, keep gaps non-negative and choose tile preferred widths that allow at least one column in the pane sizes your application supports.

9. Comparing with TilePane

JavaFX TilePane keeps tiles at their tile width and may leave unused space at the end of a row. StretchingTilePane first decides how many columns fit, then divides the whole row width by that count. This makes dashboards and card grids align cleanly with the container edge.

Aspect	TilePane	StretchingTilePane
Row end	May leave unused remainder	Remainder is distributed into tile widths
Tile size	Configured tile width / height	Computed equal width and max preferred height
Preferred height	Depends on tile settings	Depends on current width due horizontal content bias

```
// TilePane-like fixed cards are not the goal; set pref size
// and let StretchingTilePane stretch the row.
tile.setPrefSize(150, 100);
```

10. Card grid design patterns

A row-stretching grid is ideal for dashboards, settings cards and launchers where columns should align to the pane edges. Give every card a consistent preferred size, and put flexible content inside each card rather than varying the tile nodes wildly.

Concern	Recommended practice
Preferred sizes	Set explicit preferred sizes for important children so the layout maths has stable inputs.
Insets and padding	Remember that pane insets are part of the available area calculation or child placement.
Managed state	Only managed children should be expected to participate in layout decisions.
Runtime changes	Property invalidation calls requestLayout, so batch related changes where possible.

The pane uses the maximum preferred tile width to choose columns. One unusually wide card therefore reduces the column count for the whole grid. Keep preferred widths consistent for predictable responsive behaviour.

```
for (Node card : pane.getChildren()) {
    ((Region) card).setPrefSize(180, 120);
}
pane.setHgap(12);
pane.setVgap(12);
```

11. Height-for-width integration

Because `getContentBias()` returns `Orientation.HORIZONTAL`, parent layouts should ask for preferred height using the current width. This is important in scroll panes and resizable dashboards.

Parent	Guidance
ScrollPane	Let the pane compute preferred height for the viewport width.
VBox	Give the tile pane a width through fillWidth or max width settings.
BorderPane center	The center width drives the number of rows.
Fixed-size parent	Choose preferred tile width so at least one column fits.

If the parent asks with an invalid or very small width, the source can report zero preferred height because no columns fit. This is expected from the implementation.

12. Responsive dashboard recipe

For dashboards, place the StretchingTilePane in a ScrollPane and let width changes drive the number of rows. The pane computes preferred height from width, so the scroll pane can scroll vertically without horizontal scrolling.

Step	Reason
Set tile preferred size	Defines the column-count baseline.
Set hgap and vgap	Creates stable gutters between cards.
Wrap in ScrollPane	Lets total height exceed viewport height.
Keep cards managed	Only managed cards participate in the grid.

```
ScrollPane scroll = new ScrollPane(pane);
scroll.setFitToWidth(true);
pane.setHgap(12);
pane.setVgap(12);
```

13. Common mistakes

Most layout surprises come from inconsistent preferred sizes. Since the maximum preferred tile width controls the column count, one oversized child can turn a four-column grid into a three-column grid for every row.

- Avoid mixing very wide and very narrow tile preferred widths.
- Avoid negative gaps even though the properties do not reject them.
- Do not add managed non-Region nodes directly.
- Do not expect individual rows to choose different column counts.

When a design needs masonry-style columns or variable-height cards, use a different layout. StretchingTilePane deliberately keeps rows regular and predictable.

```
StackPane wrapper = new StackPane(shapeNode);
wrapper.setPrefSize(160, 100);
pane.getChildren().add(wrapper);
```

14. Recipes

14.1 Ignore a child

```
node.setManaged(false);
node.setVisible(false);
```

14.2 Checklist

1. Use Region children for preferred-height calculation.
2. Set meaningful preferred tile sizes.
3. Use padding for outer margins.

4 Expect equal tile sizes per layout pass.

15. See also

- Demo application: `com.dlsc.gemsfx.demo.StretchingTilePaneApp` (run with `mvn javafx:run -f gemsfx-demo/pom.xml -Dmain.class=com.dlsc.gemsfx.demo.StretchingTilePaneApp`)
- JavaFX `TilePane` - standard non-stretching alternative.
- `ResponsivePane` - another width-sensitive layout.
- API documentation: https://dlsc-software-consulting-gmbh.github.io/GemsFX/api/