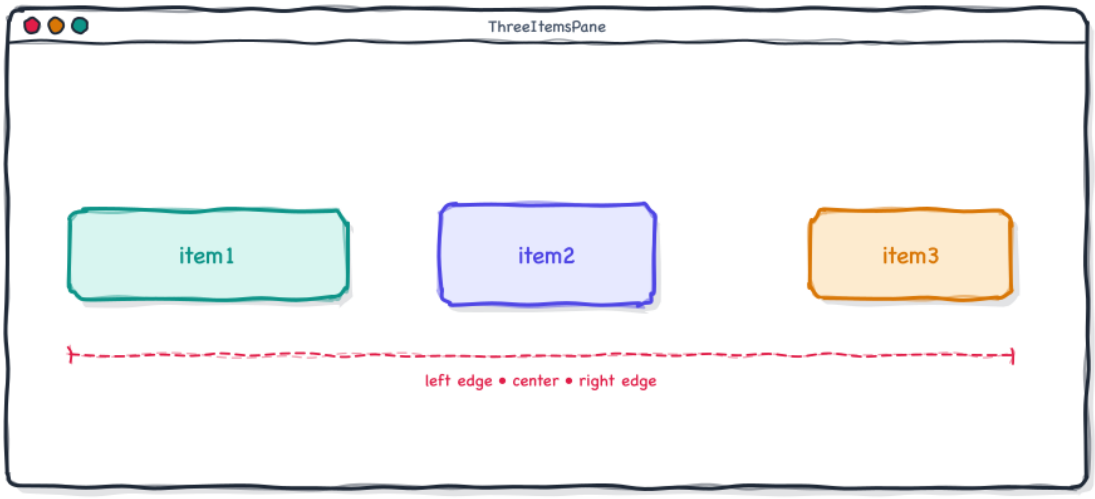
ThreeItemsPane

Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls



A generated cartoon overview of ThreeItemsPane.

ThreeItemsPane manages up to three nodes and positions them as edge, center and opposite-edge items in a horizontal or vertical orientation.

Table of Contents

1. Introduction 2
1.1 Key features 2
1.2 Maven dependency 2
2. Getting started 2
3. Anatomy 3
4. Layout algorithm 3
5. Sizing and children 4
6. Styling 5
7. Vertical layout details 5
8. Null items and managed children 5
9. Size computations by orientation 6
10. Toolbar design patterns 6
11. Choosing between alternatives 6
12. Practical sizing checklist 7
13. Common mistakes 7
14. Recipes 7
14.1 Centered title bar 7
14.2 Checklist 8
15. See also 8

1. Introduction

ThreeItemsPane manages up to three item nodes and positions them as edge, center and opposite-edge items in horizontal or vertical orientation. It is useful for headers, toolbars and status bars.

1.1 Key features

- item1, item2 and item3 are the public node slots.
- Horizontal: left, center, right.
- Vertical: top, center, bottom.
- Spacing prevents later items from overlapping earlier ones.
- Children list is rebuilt from non-null item properties.

1.2 Maven dependency

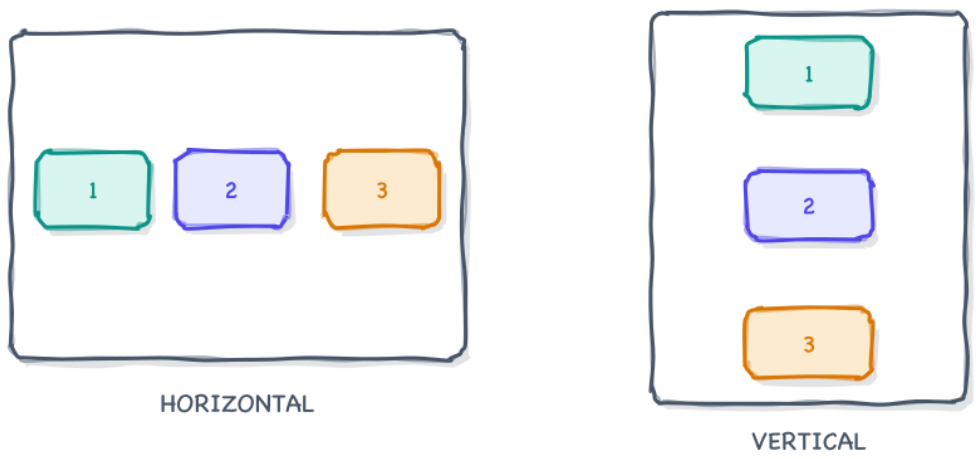
```
<dependency>
    <groupId>com.dlsc.gemsfx</groupId>
    <artifactId>gemsfx</artifactId>
    <version>4.4.1</version>
</dependency>
```

Maven coordinates for the GemsFX control library.

2. Getting started

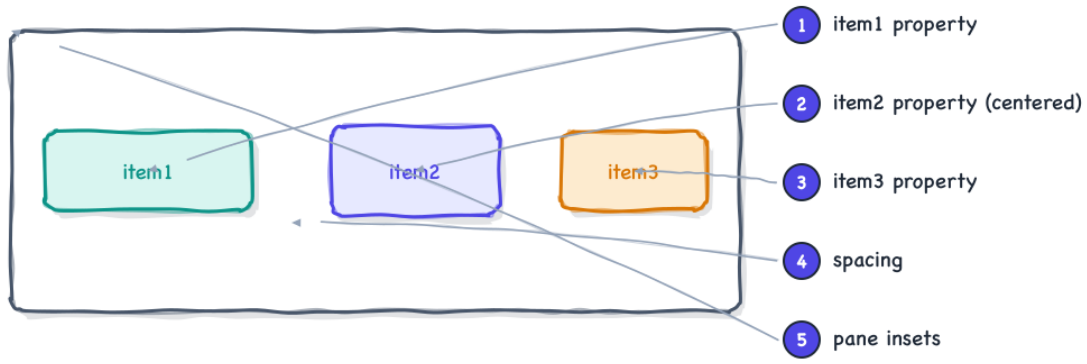
```
ThreeItemsPane pane = new ThreeItemsPane();
pane.setSpacing(20);
pane.setItem1(new Label("Application"));
pane.setItem2(new Label("Center"));
pane.setItem3(new Label("User"));
```

A three-part header.



Horizontal and vertical orientations.

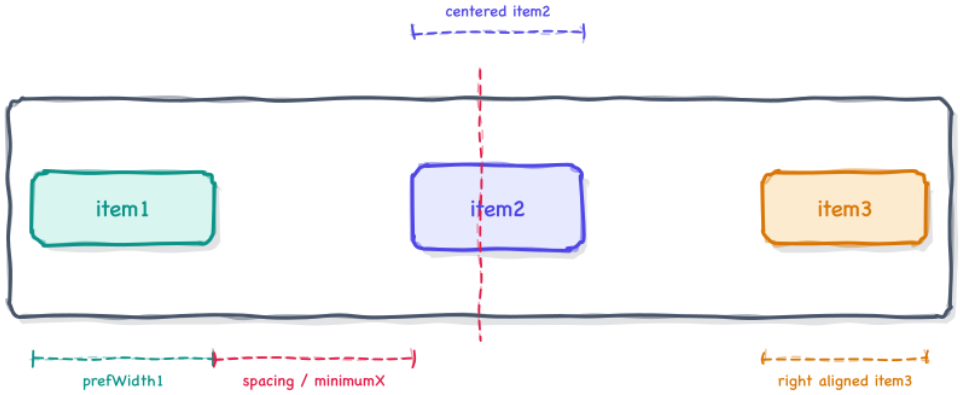
3. Anatomy



The three semantic item positions.

Property	Type	Default	Description
orientation	ObjectProperty<Orientation>	Orientation.HORIZONTAL	Chooses horizontal or vertical.
item1	ObjectProperty<Node>	null	First item: left or top.
item2	ObjectProperty<Node>	null	Center item.
item3	ObjectProperty<Node>	null	Third item: right or bottom.
spacing	DoubleProperty	0	Separation between managed items.

4. Layout algorithm

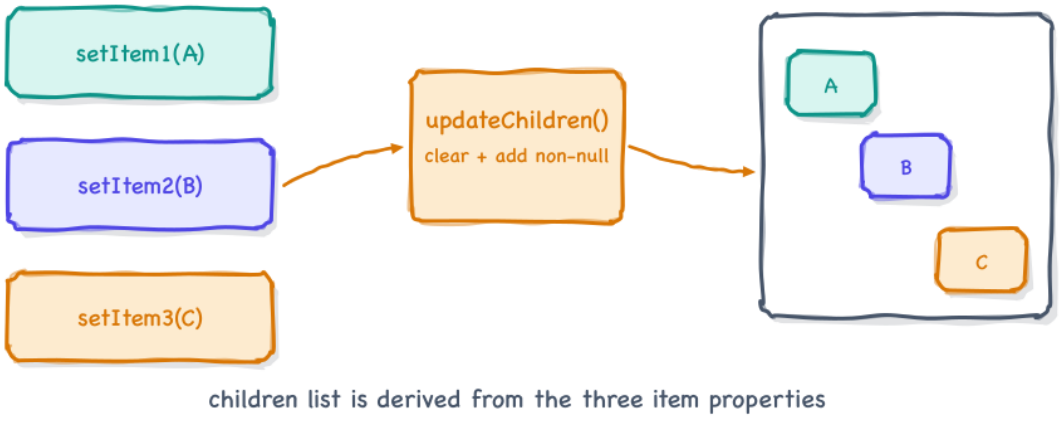


Horizontal positioning with overlap protection.

Item1 is placed at the leading inset. Item2 is centered around half the pane but shifted after item1 if needed. Item3 is aligned to the trailing inset but shifted after item2 if needed.

```
x2 = max(minimumX, width / 2 - item2.prefWidth / 2)
x3 = max(minimumXAfterItem2, width - rightInset - item3.prefWidth)
```

5. Sizing and children



Item properties rebuild the children list.

The pane clears and repopulates its children whenever item1, item2 or item3 changes. Preferred size uses child preferred sizes and insets; content bias is inherited from children.

> **Note.** The source horizontal pref / min width expression is precedence-sensitive: `getChildren().size() - 1 * getSpacing()`. This manual documents observed slot layout rather than relying on that expression.

6. Styling

ThreeItemsPane adds no style class, no user agent stylesheet and no styleable CSS properties. Add an application style class to the pane or to individual items.

```
pane.getStyleClass().add("app-header");
pane.getItem2().getStyleClass().add("header-title");
```

7. Vertical layout details

Vertical layout mirrors the horizontal algorithm. Item1 is placed at the top inset, item2 is vertically centered but not above the minimum y after item1, and item3 is bottom aligned but not above the minimum y after item2.

Item	Vertical y coordinate
item1	`insets.getTop()`
item2	`max(minimumY, height / 2 - prefHeight / 2)`
item3	`max(minimumY, height - bottomInset - prefHeight)`

All three items are horizontally centered in vertical orientation. Width computations use each child preferred width with available height as the argument.

```
pane.setOrientation(Orientation.VERTICAL);
pane.setSpacing(12);
```

8. Null items and managed children

The pane supports any combination of null and non-null item properties. The children list contains only non-null items, in item order. Layout methods still query the three properties directly, so replacing one property is the supported way to change a slot.

Slots set	Children list
item1 only	item1
item2 only	item2
item1 + item3	item1, item3
all three	item1, item2, item3

Because the class extends Pane, callers can technically mutate `getChildren()`. Do not use that as the main API: the next property change clears and rebuilds the children list from the item properties.

> **Careful.** Use item properties as the source of truth. Direct child-list edits are fragile with this pane.

9. Size computations by orientation

Preferred and minimum size methods compute each child size first. In horizontal orientation, height is the maximum child height and width is intended to be the sum of child widths plus spacing and insets. In vertical orientation, width is the maximum child width and height is the sum of child heights plus spacing and insets.

Orientation	Preferred width	Preferred height
HORIZONTAL	sum of child preferred widths plus horizontal insets and spacing expression from source	max child preferred height plus vertical insets
VERTICAL	max child preferred width plus horizontal insets	sum of child preferred heights plus vertical insets and spacing

Maximum size is unbounded along the layout axis: horizontal orientation returns `Double.MAX_VALUE` for max width, while vertical orientation returns `Double.MAX_VALUE` for max height.

10. Toolbar design patterns

The center item is visually stable as long as enough room exists between the edge items. This makes ThreeItemsPane useful for title bars where the title should remain centered in the whole window, not merely centered in the remaining space.

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.

When the edge items become too large, the source protects against overlap by shifting the center and trailing items. That is preferable to clipping, but applications should still keep header controls compact.

```
pane.setItem1(navigationButtons);
pane.setItem2(titleLabel);
pane.setItem3(accountMenu);
pane.setSpacing(16);
```

11. Choosing between alternatives

Use ThreeItemsPane when positions have meaning. Use HBox with Spacer when the goal is simply to consume extra space. Use BorderPane when five regions and resizable center content are needed.

Requirement	Best fit
Left / center / right header	ThreeItemsPane
Push right controls to the edge	HBox plus Spacer
Top / bottom / left / right / center application layout	BorderPane

Requirement	Best fit
Responsive navigation rail	ResponsivePane

This distinction keeps layout code easy to read: item properties communicate intent directly, while generic child lists require readers to infer meaning from order and spacer nodes.

12. Practical sizing checklist

A three-slot pane is usually placed in a constrained header or footer. The clearest results come from making each child report a realistic preferred size and letting the pane preserve the semantic positions.

Child type	Sizing advice
Text label	Let the label compute its preferred size; avoid unnecessary max width.
Button cluster	Wrap related buttons in an HBox and use that HBox as one item.
Avatar or status node	Set an explicit preferred size so the trailing edge is predictable.
Long title	Consider ellipsis or a maximum width before it collides with edge items.

If the center item must never shift, constrain the edge items externally. The source protects from overlap by shifting nodes, not by clipping or compressing children.

13. Common mistakes

The most common mistake is to treat ThreeItemsPane like a generic Pane and add children directly. That may appear to work until an item property changes and updateChildren clears and rebuilds the children list.

- Do not add unrelated nodes through `getChildren()`.
- Do not rely on child order after setting an item property.
- Do not expect the pane to resize or compress child nodes; it uses preferred sizes.
- Do not use it as a replacement for BorderPane when a resizable center region is required.

A good rule is simple: if a node is not item1, item2 or item3, it belongs outside this pane or inside one of the three item nodes as a nested layout.

```
HBox leftCluster = new HBox(backButton, titleIcon);
pane.setItem1(leftCluster);
// not: pane.getChildren().add(titleIcon);
```

14. Recipes

14.1 Centered title bar

```
pane.setItem1(backButton);
pane.setItem2(new Label("Details"));
pane.setItem3(settingsButton);
```

14.2 Checklist

1. Use item properties instead of direct child mutations.
2. Give each item a sensible preferred size.
3. Increase spacing when edge items approach the center item.
4. Use null to omit a slot.

15. See also

- Demo application: `com.dlsc.gemsfx.demo.ThreeItemsPaneApp` (run with `mvn javafx:run -f gemsfx-demo/pom.xml -Dmain.class=com.dlsc.gemsfx.demo.ThreeItemsPaneApp`)
- `Spacer` - flexible gap for HBox / VBox.
- `ResponsivePane` - adaptive sidebar layout.
- API documentation: https://dlsc-software-consulting-gmbh.github.io/GemsFX/api/