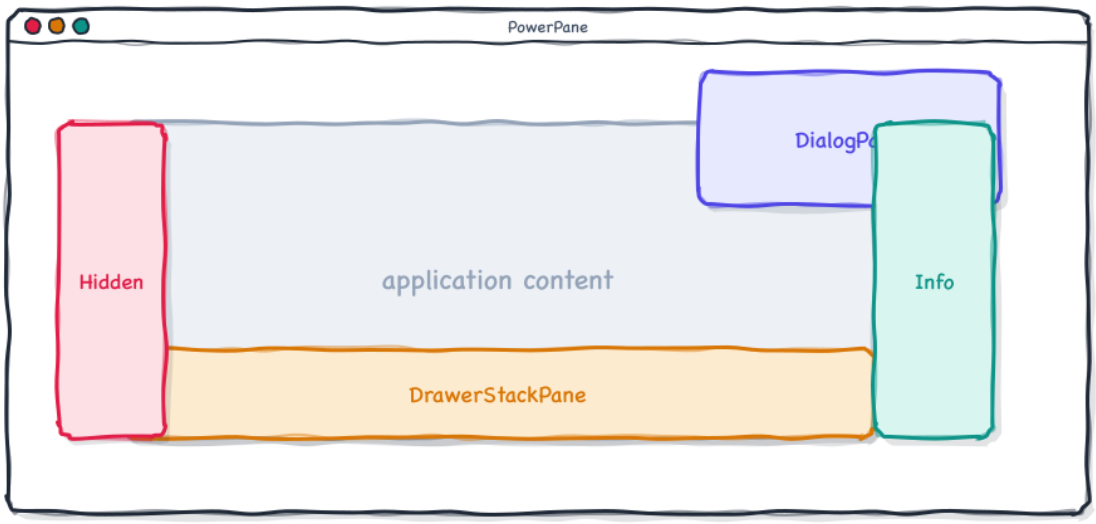
# PowerPane

## Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls




*A generated cartoon overview of PowerPane.*

PowerPane combines InfoCenterPane, DialogPane, DrawerStackPane and HiddenSidesPane into one StackPane shell and delegates features through getters for the composed panes.

# Table of Contents

# 1. Introduction

**PowerPane** is a convenience shell that composes `InfoCenterPane`, `DialogPane`, `DrawerStackPane` and `HiddenSidesPane`. It delegates behaviour through final getters.

## 1.1 Key features

- Direct child is an InfoCenterPane.
- DialogPane is stacked with DrawerStackPane inside the info center content.
- DrawerStackPane contains HiddenSidesPane.
- HiddenSidesPane content is bound to PowerPane content.
- Protected factory methods create the composed panes.

## 1.2 Maven dependency

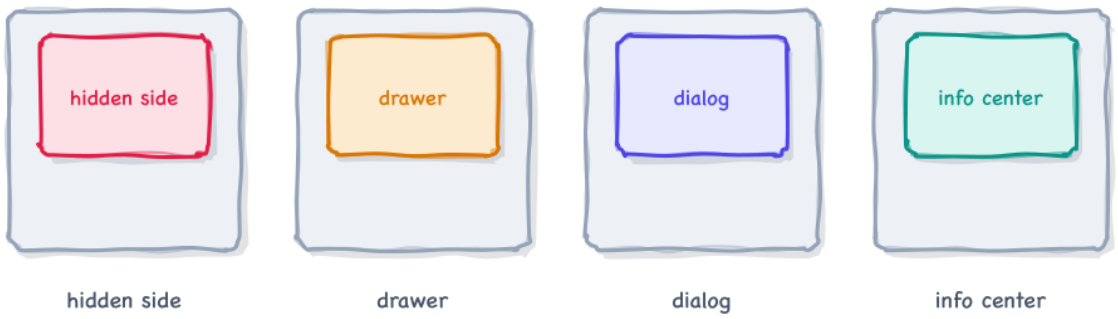
```
<dependency>
    <groupId>com.dlsc.gemsfx</groupId>
    <artifactId>gemsfx</artifactId>
    <version>4.4.1</version>
</dependency>
```

*Maven coordinates for the GemsFX control library.*

# 2. Getting started

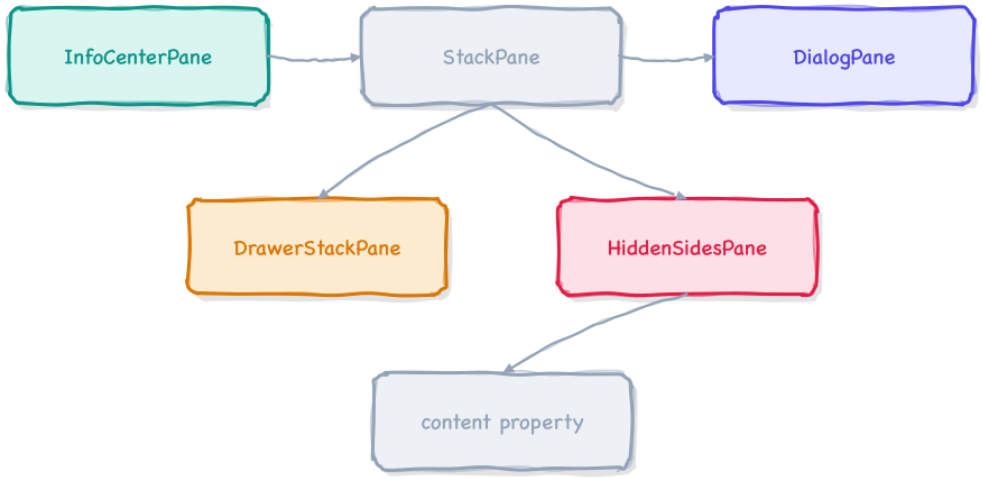
```
PowerPane powerPane = new PowerPane(mainView);
powerPane.getDialogPane().showInformation("Info", "Ready");
powerPane.getDrawerStackPane().setDrawerContent(settingsView);
powerPane.getHiddenSidesPane().setLeft(navigationRail);
```

*Using the composed panes through getters.*




*The overlay families composed by PowerPane.*

# 3. Anatomy



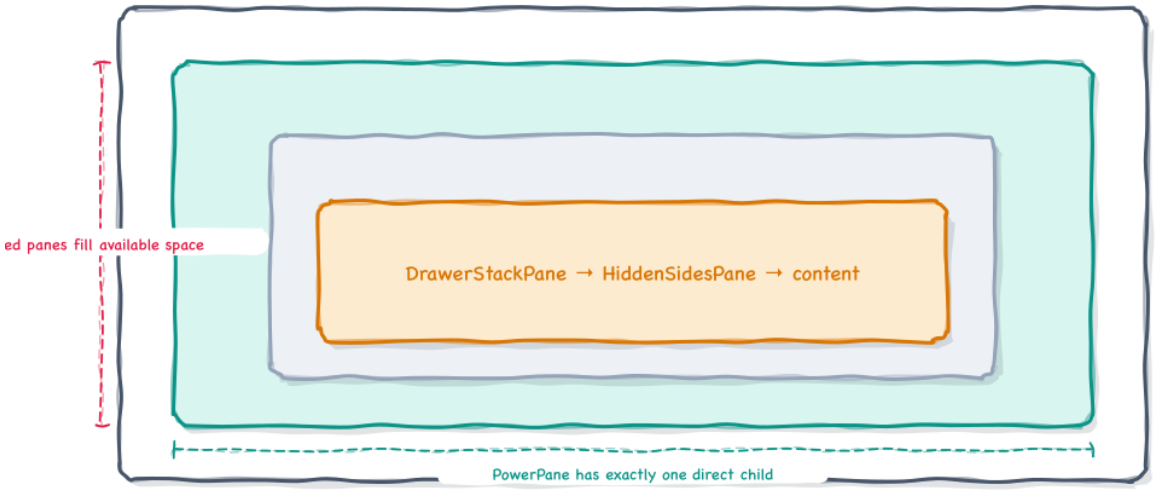

*How PowerPane nests the composed panes.*

| Part | Getter | Role |
|---|---|---|
| Info center | `getInfoCenterPane()` | Outermost notification surface. |
| Dialog pane | `getDialogPane()` | Dialog overlay. |
| Drawer stack | `getDrawerStackPane()` | Bottom drawer layer. |
| Hidden sides | `getHiddenSidesPane()` | Edge tray layer. |
| Content | `content` | Main application UI. |

# 4. Control API

| Property | Type | Default | Description |
|---|---|---|---|
| content | ObjectProperty<Node> | null | Main UI bound into HiddenSidesPane. |
| infoCenterPane | InfoCenterPane | new InfoCenterPane() | Returned by final getter. |
| dialogPane | DialogPane | new DialogPane() | Returned by final getter. |
| drawerStackPane | DrawerStackPane | new DrawerStackPane() | Returned by final getter. |
| hiddenSidesPane | HiddenSidesPane | new HiddenSidesPane() | Returned by final getter. |

# 5. Composition and layout



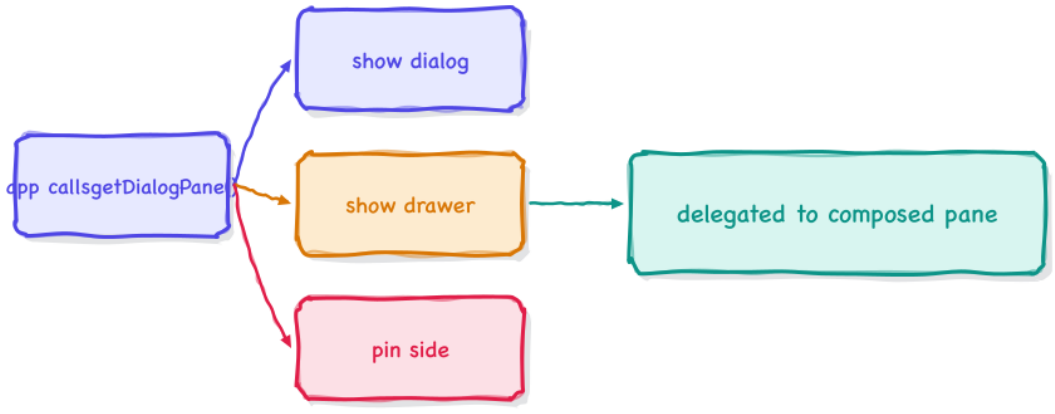

*Nested panes fill the PowerPane area.*

The constructor binds hiddenSidesPane.contentProperty() to contentProperty(), adds HiddenSidesPane to DrawerStackPane, wraps DrawerStackPane and DialogPane in a StackPane, sets that StackPane as InfoCenterPane content and finally adds InfoCenterPane as the only child.

```
hiddenSidesPane.contentProperty().bind(contentProperty());
drawerStackPane.getChildren().add(hiddenSidesPane);
infoCenterPane.setContent(new StackPane(drawerStackPane, dialogPane));
getChildren().add(infoCenterPane);
```

*The composition graph from the constructor.*

# 6. Delegation patterns




*Applications configure the composed panes.*

| Need | Use |
|---|---|
| Show a dialog | `getDialogPane()` |
| Open a drawer | `getDrawerStackPane()` |
| Install side trays | `getHiddenSidesPane()` |
| Show notifications | `getInfoCenterPane()` |

## 7. Subclassing and styling

Override the protected create methods to provide custom subclasses during construction. The composed pane fields are final after construction.

```
public class MyPowerPane extends PowerPane {
    @Override
    protected DialogPane createDialogPane() {
        DialogPane pane = new DialogPane();
        pane.setAnimationDuration(Duration.millis(120));
        return pane;
    }
}
```

PowerPane adds style class `power-pane` but has no user agent stylesheet and no styleable CSS properties.

## 8. Delegated property map

PowerPane intentionally exposes only its own content property. Everything else is configured on the composed pane that actually owns the feature. This keeps PowerPane small and avoids duplicating the APIs of DialogPane, DrawerStackPane, HiddenSidesPane and InfoCenterPane.

| Feature | Configure on |
| --- | --- |
| Drawer content, title, animation and height | `getDrawerStackPane()` |
| Hidden left / right / top / bottom trays | `getHiddenSidesPane()` |
| Dialog animation, converter and dialog display | `getDialogPane()` |
| Notification groups and info-center visibility | `getInfoCenterPane()` and its InfoCenterView |

```
powerPane.getDrawerStackPane().setShowDrawerTitle(true);
powerPane.getHiddenSidesPane().setTriggerDistance(24);
powerPane.getDialogPane().setAnimationDuration(Duration.millis(150));
```

## 9. Construction sequence

The constructor order matters for subclassing. First the four create methods are called. Then the content binding and nested containment are established. Finally the InfoCenterPane is added as the one direct child of the PowerPane.

1. `createInfoCenterPane()`
2. `createDialogPane()`
3. `createDrawerStackPane()`
4. `createHiddenSidesPane()`
5. Bind hidden sides content to PowerPane content.
6. Nest HiddenSidesPane inside DrawerStackPane.
7. Set InfoCenterPane content to a StackPane containing drawer and dialog panes.

> **Careful.** Because factory methods run from the constructor, overrides should not depend on subclass fields initialized after `super()`.

## 10. Application shell strategy

PowerPane works best as the center of the primary scene or the center of a root BorderPane. Treat it as the owner of application overlays: route dialogs, drawers, side trays and notifications through its child panes instead of creating parallel overlay stacks elsewhere.

| 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. |

| Concern | Recommended practice |
| --- | --- |
| Runtime changes | Property invalidation calls requestLayout, so batch related changes where possible. |

Because the content property is bound into the nested HiddenSidesPane, replacing content updates the main application view without reconstructing the overlay infrastructure.

```
PowerPane shell = new PowerPane();
shell.setContent(loginView);
// later
shell.setContent(mainApplicationView);
```

# 11. Layering order in practice

The composed layers are arranged so each specialized pane can do its normal job. Hidden sides and drawer content live inside DrawerStackPane; DialogPane is stacked above that drawer stack; InfoCenterPane hosts the entire stack and can show notifications over the application shell.

| Layer | Typical visual result |
| --- | --- |
| HiddenSidesPane | Edge trays around the main content. |
| DrawerStackPane | Bottom drawer over the content / hidden-side layer. |
| DialogPane | Dialog overlay above drawer and content. |
| InfoCenterPane | Notification center surface around the whole shell. |

When debugging a complex shell, inspect the individual composed panes first. PowerPane itself only creates and connects the composition graph.

# 12. Recipes

## 12.1 Install all hidden sides

```
HiddenSidesPane sides = powerPane.getHiddenSidesPane();
sides.setLeft(leftTray);
sides.setRight(rightTray);
sides.setTop(topTray);
sides.setBottom(bottomTray);
```

## 12.2 Checklist

1. Set content on PowerPane, not directly on the nested HiddenSidesPane.
2. Use getters to configure child panes.
3. Override factory methods only in subclasses.
4. Read composed pane manuals for delegated properties.

## 13. See also

- Demo application: `com.dlsc.gemsfx.demo.PowerPaneApp` (run with `mvn javafx:run -f gemsfx-demo/pom.xml -Dmain.class=com.dlsc.gemsfx.demo.PowerPaneApp`)
- `InfoCenterPane` - notification surface.
- `DialogPane` - dialog layer.
- `DrawerStackPane` - drawer layer.
- `HiddenSidesPane` - edge tray layer.
- API documentation: https://dlsc-software-consulting-gmbh.github.io/GemsFX/api/