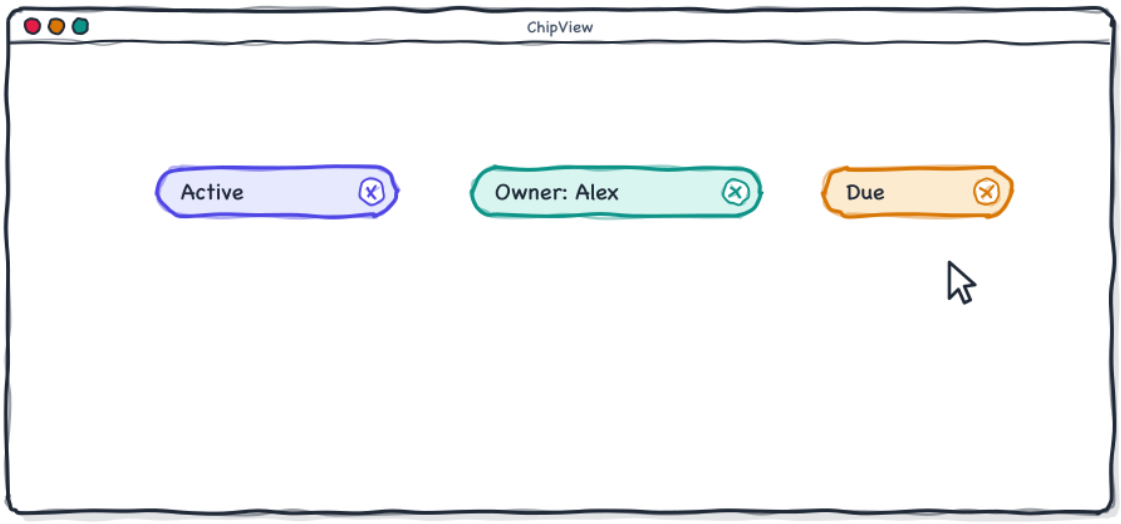
ChipView

Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls



Generated cartoon overview of ChipView.

A small pill-shaped Control representing one model object. The skin displays text and an optional graphic, and shows a close icon only when an onClose consumer is set.

Table of Contents

1. Introduction

ChipView A small pill-shaped Control representing one model object. The skin displays text and an optional graphic, and shows a close icon only when an onClose consumer is set.

1.1 Key features

- The close icon is visible and managed only when onClose is non-null.
- Clicking close invokes the consumer with getValue().
- API and styling details below are verified against the control, skin, CSS and resource bundles.

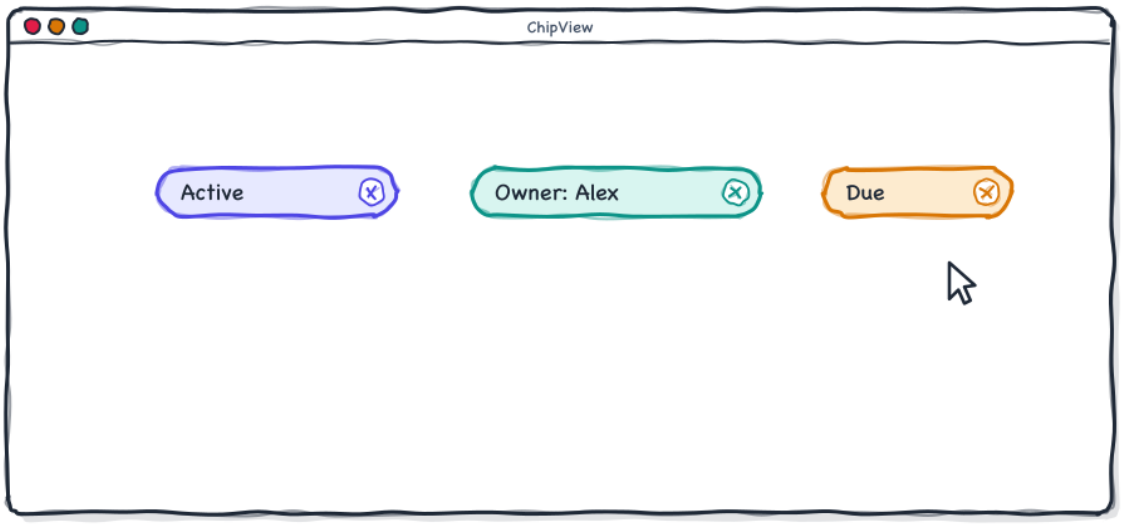
1.2 Maven dependency

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

2. Getting started

```
ChipView<Filter> chip = new ChipView<>();
chip.setValue(filter);
chip.setText(filter.label());
chip.setGraphic(new FontIcon(MaterialDesign.MDI_FILTER));
chip.setOnClose(value -> activeFilters.remove(value));
```

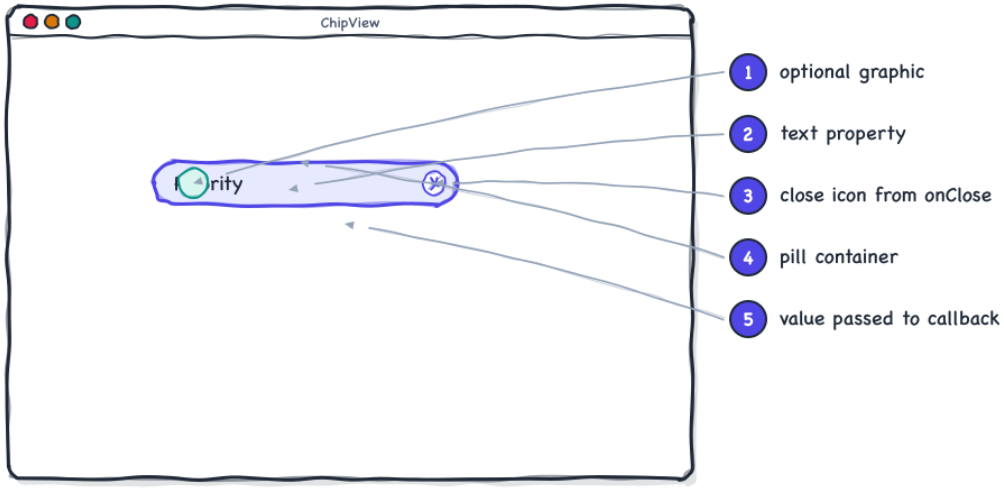
A compact setup for ChipView.



ChipView in a typical application context.

3. Anatomy

The diagram identifies the nodes and state holders created by the implementation.



The parts of ChipView.

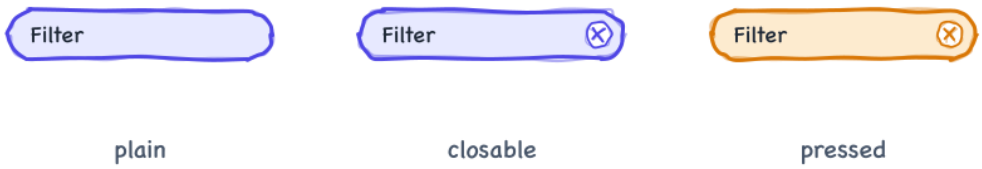
Topic	Verified source detail
Stylesheet	`com/dlsc/gemsfx/chip-view.css`
Root style class	`.chip-view`
Graphics	Generated SVGs; no screenshots.

4. Control API

4.1 Properties and callbacks

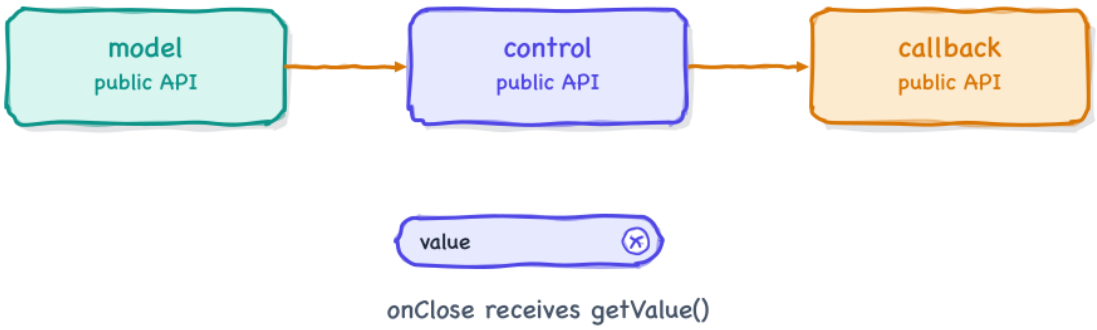
Property	Type	Default	Description
value	ObjectProperty<T>	null	Model object represented by the chip.
text	StringProperty	localized "Untitled"	Text shown by the chip.
graphic	ObjectProperty<Node>	null	Optional label graphic.
contentDisplay	ObjectProperty<ContentDisplay>	LEFT	Graphic/text placement; styleable.
onClose	ObjectProperty<Consumer<T>>	null	Called with value when the close icon is clicked.

5. Behaviour



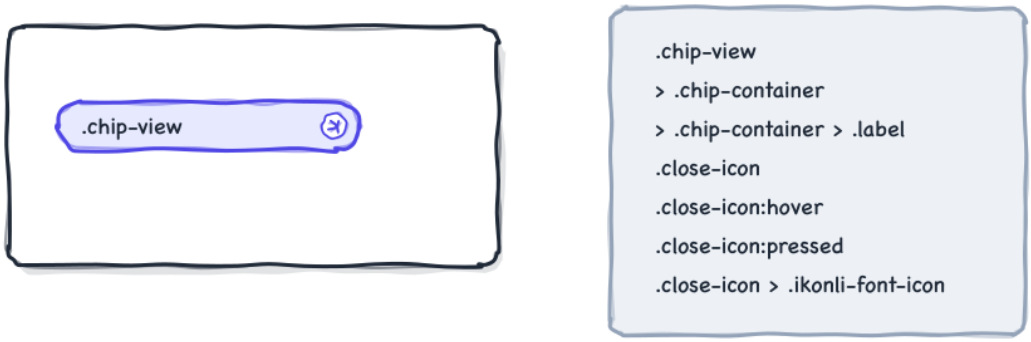
Important runtime states of ChipView.

- The close icon is visible and managed only when onClose is non-null.
- Clicking close invokes the consumer with getValue().
- The control does not remove itself; update the owning model in the callback.



How application data flows through ChipView.

6. Styling



Style hooks for ChipView.

6.1 Style classes and pseudo classes

Selector / pseudo class	Purpose
`.chip-view`	Verified in source, skin or CSS.
`> .chip-container`	Verified in source, skin or CSS.
`> .chip-container > .label`	Verified in source, skin or CSS.
`.close-icon`	Verified in source, skin or CSS.
`.close-icon:hover`	Verified in source, skin or CSS.
`.close-icon:pressed`	Verified in source, skin or CSS.
`.close-icon > .ikonli-font-icon`	Verified in source, skin or CSS.

6.2 Styleable CSS properties

CSS property	Type	Default
`-fx-content-display`	ContentDisplay	LEFT

```
.chip-view {
    /* start with the documented root selector */
}
```

7. Localization

Key	English text
`default.text.untitled`	Untitled
`accessible.role-description`	chip

8. Accessibility

Sets AccessibleRole.BUTTON with localized role description chip. No accessible text binding is installed.

9. Recipes

9.1 Programmatic configuration

```
ChipView<Filter> chip = new ChipView<>();
chip.setValue(filter);
chip.setText(filter.label());
chip.setGraphic(new FontIcon(MaterialDesign.MDI_FILTER));
chip.setOnClose(value -> activeFilters.remove(value));
```

9.2 Checklist

1. Use the public properties listed in the API chapter.
2. Style through documented selectors and CSS properties only.
3. Keep application model updates in callbacks such as onClose, onClear or onItemSelected.
4. Do not depend on private skin nodes except for documented style selectors.

10. Integration notes

10.1 Use public API boundaries

The implementation exposes enough properties for normal integration. Keep application state in observable lists, converters and callbacks instead of querying skin children.

10.2 Validation checklist

Concern	Recommendation
Model updates	Perform them in the documented callback or observable list.
Styling	Start at the root style class and keep selectors scoped.
Localization	Override the documented resource bundle keys only when they exist.
Accessibility	Preserve the role set by the control and add application-specific accessible text when needed.

```
// Integration pattern
// 1. Configure model properties.
// 2. Configure callbacks.
// 3. Style through documented selectors.
// 4. Leave skin internals private.
```

11. See also

- Demo application: `com.dlsc.gemsfx.demo.SimpleFilterViewApp` (run with `mvn javafx:run -f gemsfx-demo/pom.xml -Dmain.class=com.dlsc.gemsfx.demo.SimpleFilterViewApp`)
- API documentation: https://dlsc-software-consulting-gmbh.github.io/GemsFX/api/