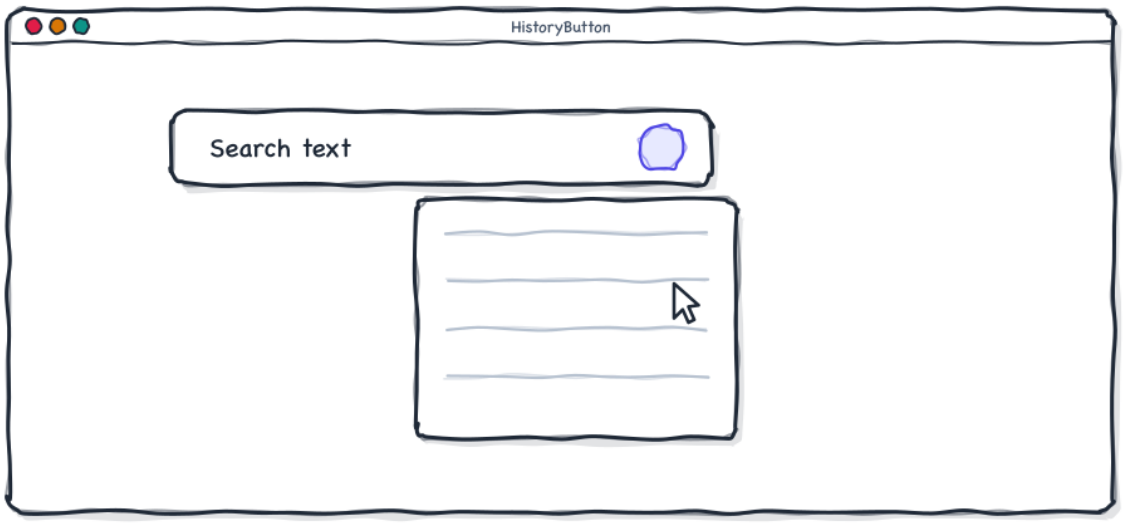
HistoryButton

Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls



Generated cartoon overview of HistoryButton.

A Button that opens a popup ListView bound to a HistoryManager. It can be embedded in an input field or used standalone.

Table of Contents

1. Introduction

HistoryButton A Button that opens a popup ListView bound to a HistoryManager. It can be embedded in an input field or used standalone.

1.1 Key features

- Clicking the button toggles the popup.
- If owner is set and not focused, owner.requestFocus() runs before showing.
- API and styling details below are verified against the control, skin, CSS and resource bundles.

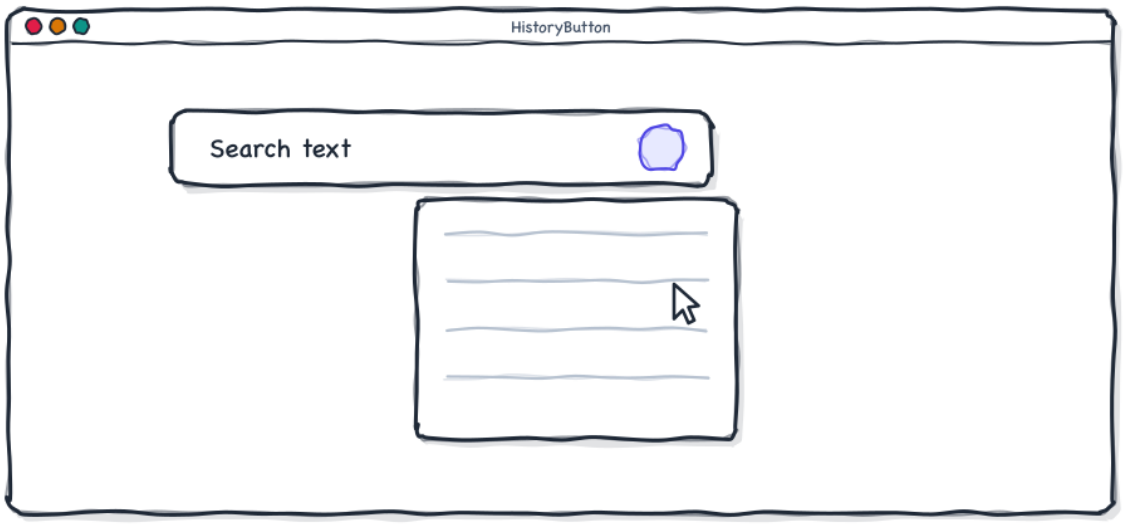
1.2 Maven dependency

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

2. Getting started

```
HistoryButton<String> button = new HistoryButton<>(textField);
StringHistoryManager manager = new StringHistoryManager(Preferences.userNodeForPackage(MyApp
.class), "recent-searches");
button.setHistoryManager(manager);
button.setOnItemSelected(textField::setText);
```

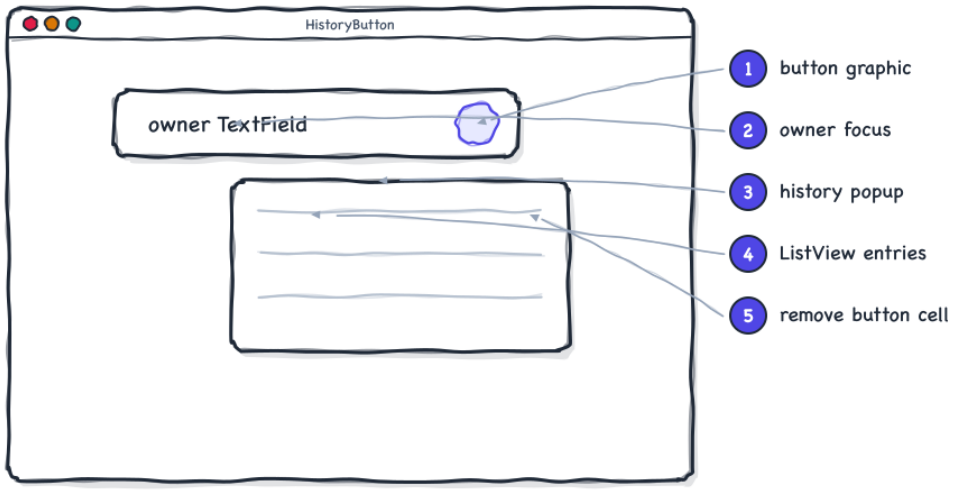
A compact setup for HistoryButton.



HistoryButton in a typical application context.

3. Anatomy

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



The parts of HistoryButton.

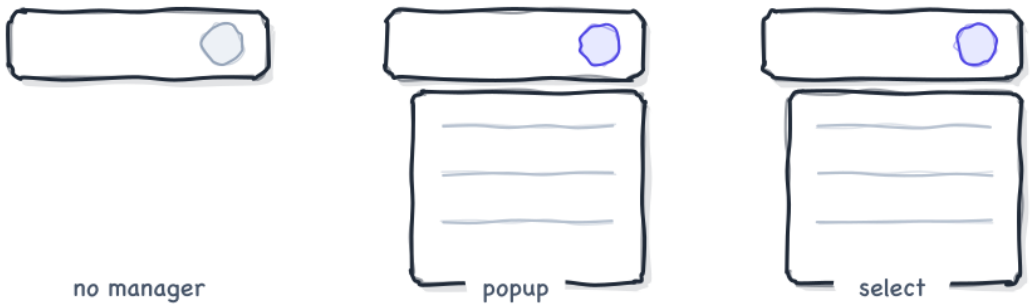
Topic	Verified source detail
Stylesheet	`com/dlsc/gemsfx/history-button.css`
Root style class	`.history-button`
Graphics	Generated SVGs; no screenshots.

4. Control API

4.1 Properties and callbacks

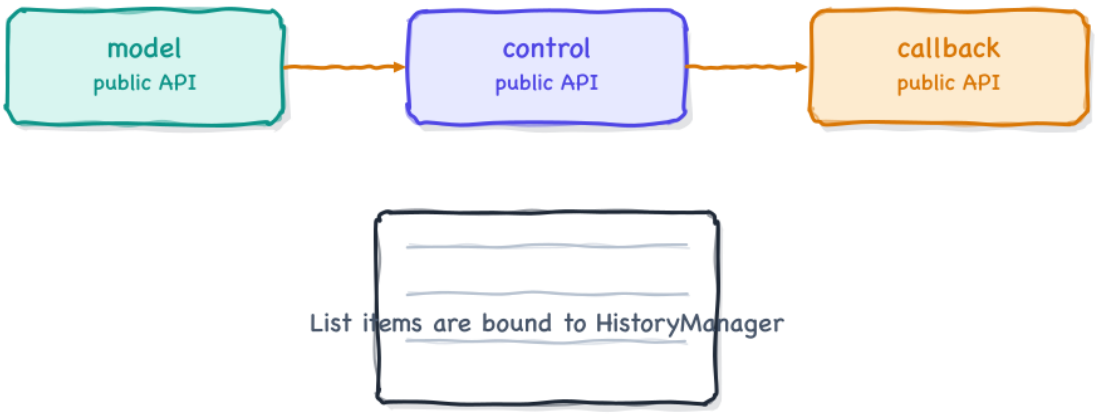
Property	Type	Default	Description
historyManager	ObjectProperty<HistoryManager<T>>	null	Source of popup entries; null disables popup.
owner	ObjectProperty<Node>	null	Optional owner that receives focus first.
placeholder	ObjectProperty<Node>	null	History ListView placeholder.
onItemSelected	ObjectProperty<Consumer<T>>	null	Called on primary click or ENTER.
popupShowing	ReadOnlyBooleanProperty	false	True while popup is open.
listDecorationLeft/right/top/bottom	ObjectProperty<Node>	null	Popup BorderPane decorations.
cellFactory	ObjectProperty<Callback<ListView<T>, ListCell<T>>>	RemovableListCell	Factory for history cells.

5. Behaviour



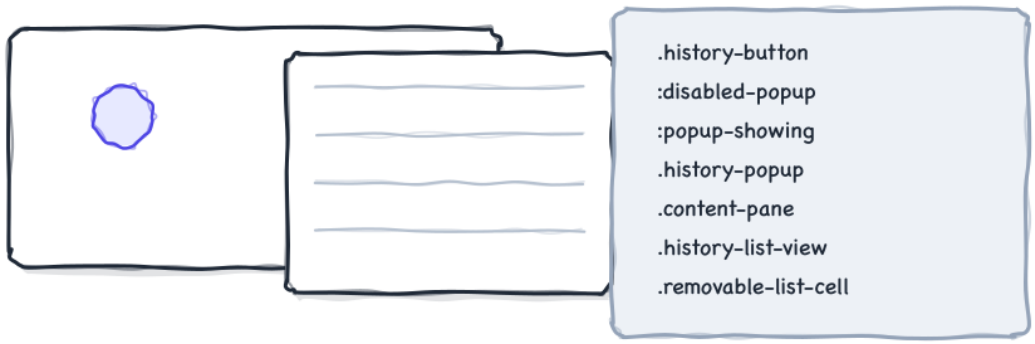
Important runtime states of HistoryButton.

- Clicking the button toggles the popup.
- If owner is set and not focused, owner.requestFocus() runs before showing.
- The popup is auto-fixing, auto-hiding and closes on ESCAPE.
- Primary click or ENTER confirms the selected ListView item.
- The default remove button calls historyManager.remove(item).



How application data flows through HistoryButton.

6. Styling



Style hooks for HistoryButton.

6.1 Style classes and pseudo classes

Selector / pseudo class	Purpose
`.history-button`	Verified in source, skin or CSS.
`:disabled-popup`	Verified in source, skin or CSS.
`:popup-showing`	Verified in source, skin or CSS.
`.history-popup`	Verified in source, skin or CSS.
`.content-pane`	Verified in source, skin or CSS.
`.history-list-view`	Verified in source, skin or CSS.
`.removable-list-cell`	Verified in source, skin or CSS.
`.remove-button`	Verified in source, skin or CSS.
`.history-popup.round`	Verified in source, skin or CSS.

6.2 Styleable CSS properties

This control declares no additional styleable CSS properties beyond inherited JavaFX properties.

```
.history-button {
    /* start with the documented root selector */
}
```

7. Localization

The verified source does not use ResourceBundleManager for HistoryButton.

8. Accessibility

Sets AccessibleRole.BUTTON. No custom accessible text binding is installed.

9. Recipes

9.1 Programmatic configuration

```
HistoryButton<String> button = new HistoryButton<>(textField);
StringHistoryManager manager = new StringHistoryManager(Preferences.userNodeForPackage(MyApp
.class), "recent-searches");
button.setHistoryManager(manager);
button.setOnItemSelected(textField::setText);
```

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