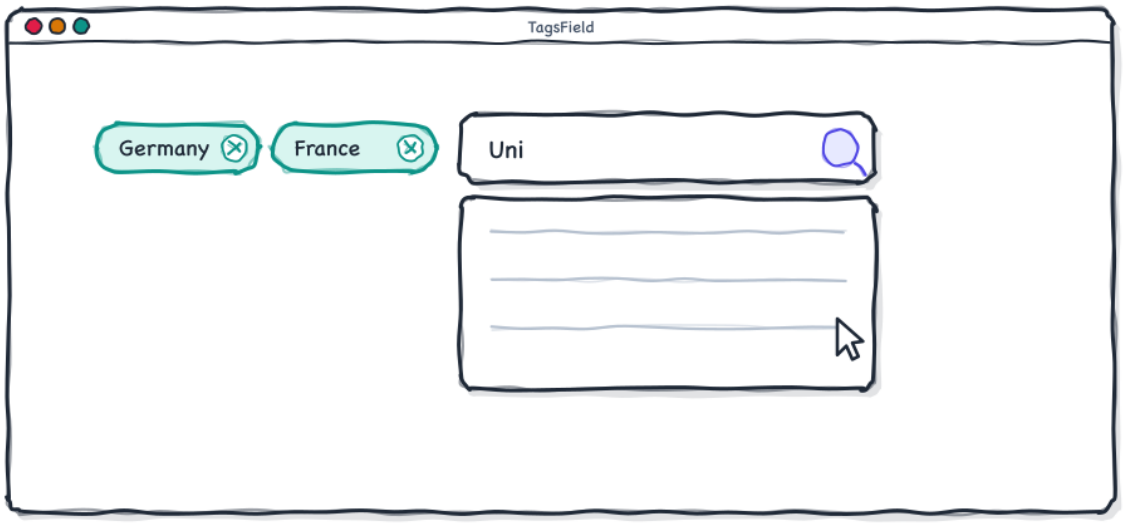
TagsField

Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls



Generated cartoon overview of TagsField.

A SearchField specialization that turns committed suggestions into removable tags shown before the editor. It owns an observable tag list and a separate selection model for tag nodes.

Table of Contents

1. Introduction

TagsField A SearchField specialization that turns committed suggestions into removable tags shown before the editor. It owns an observable tag list and a separate selection model for tag nodes.

1.1 Key features

- ENTER or RIGHT with a selected suggestion adds a tag and clears the editor.
- BACK_SPACE removes selected tags, or the last tag when the editor is empty.
- API and styling details below are verified against the control, skin, CSS and resource bundles.

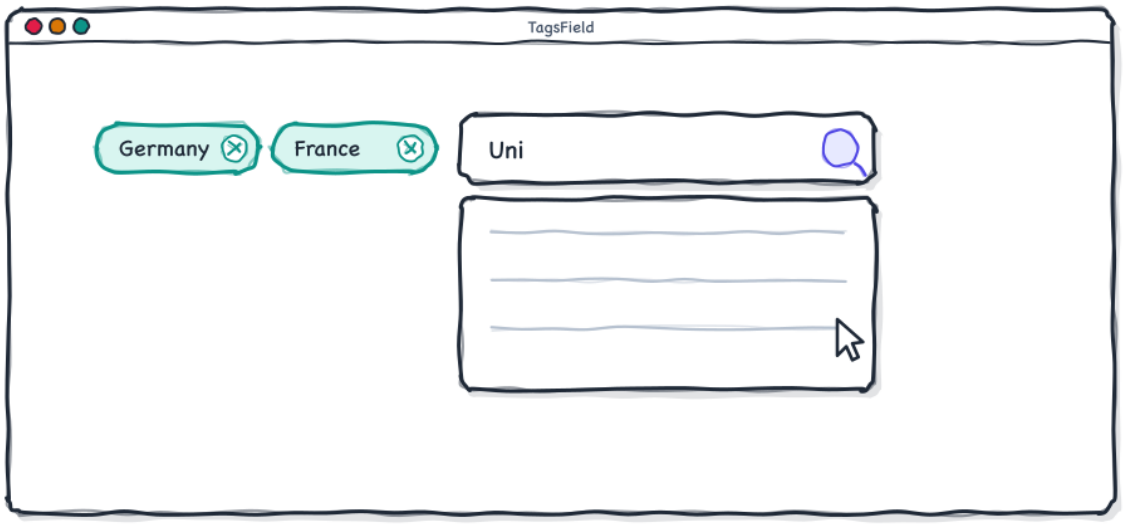
1.2 Maven dependency

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

2. Getting started

```
TagsField<Country> field = new TagsField<>();
field.setSuggestionProvider(request -> countries.stream().filter(c -> c.name().contains(requ
est.getUserText())).toList());
field.setConverter(countryConverter);
field.addTags(new Country("Germany"));
```

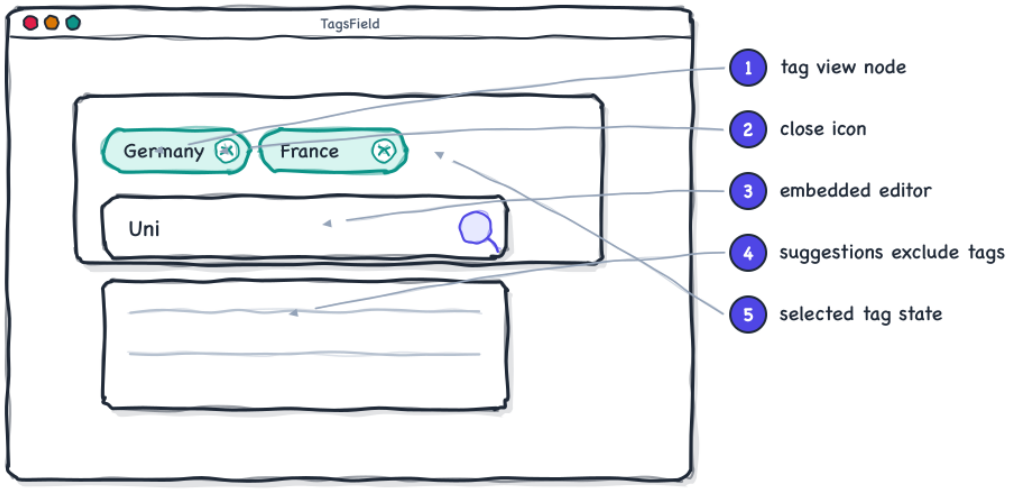
A compact setup for TagsField.



TagsField in a typical application context.

3. Anatomy

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



The parts of TagsField.

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

4. Control API

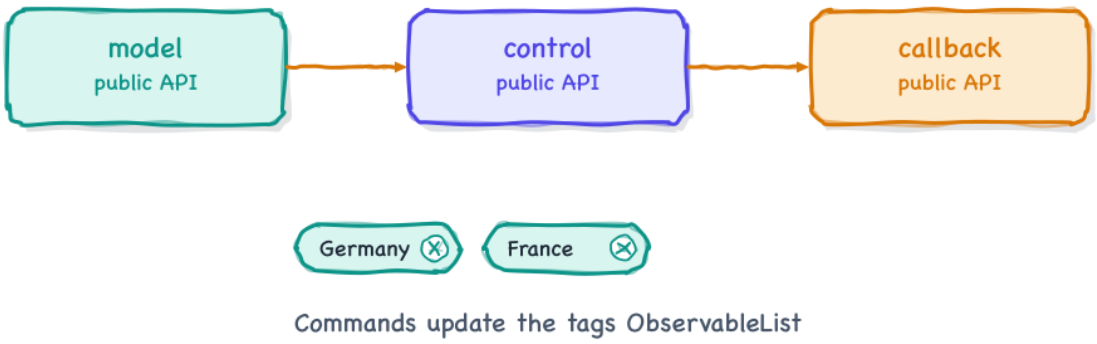
4.1 Properties and callbacks

Property	Type	Default	Description
tags	ListProperty<T>	empty list	Values currently shown as tags.
tagViewFactory	ObjectProperty<Callback<T, Node>>	Label with close icon	Creates tag nodes.
tagSelectionModel	ObjectProperty<MultipleSelectionModel<T>>	TagFieldSelectionModel	Selection model for tags; default MULTIPLE.
editorMinWidth	DoubleProperty	20	Editor width while empty.
editorPrefWidth	DoubleProperty	200	Editor width while typing.
graphic	ObjectProperty<Node>	null	TagsField removes inherited history graphic.
suggestionProvider / converter / matcher / comparator	inherited SearchField properties	same as SearchField	Configure as for SearchField.

5. Behaviour

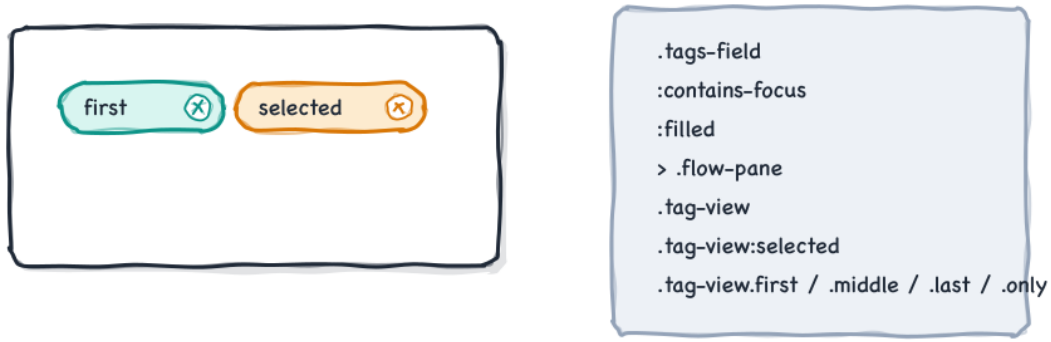
Important runtime states of TagsField.

- ENTER or RIGHT with a selected suggestion adds a tag and clears the editor.
- BACK_SPACE removes selected tags, or the last tag when the editor is empty.
- shortcut+Z and shortcut+shift+Z undo and redo tag commands.
- LEFT and RIGHT at caret zero move through tag selection.



How application data flows through TagsField.

6. Styling



Style hooks for TagsField.

6.1 Style classes and pseudo classes

Selector / pseudo class	Purpose
`.tags-field`	Verified in source, skin or CSS.
`:contains-focus`	Verified in source, skin or CSS.
`:filled`	Verified in source, skin or CSS.
`> .flow-pane`	Verified in source, skin or CSS.
`.tag-view`	Verified in source, skin or CSS.
`.tag-view:selected`	Verified in source, skin or CSS.
`.tag-view.first / .middle / .last / .only`	Verified in source, skin or CSS.
`.close-icon > .close`	Verified in source, skin or CSS.

6.2 Styleable CSS properties

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

```
.tags-field {
    /* start with the documented root selector */
}
```

7. Localization

The verified source does not use ResourceBundleManager for TagsField.

8. Accessibility

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

9. Recipes

9.1 Programmatic configuration

```
TagsField<Country> field = new TagsField<>();
field.setSuggestionProvider(request -> countries.stream().filter(c -> c.name().contains(requ
est.getUserText())).toList());
field.setConverter(countryConverter);
field.addTags(new Country("Germany"));
```

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