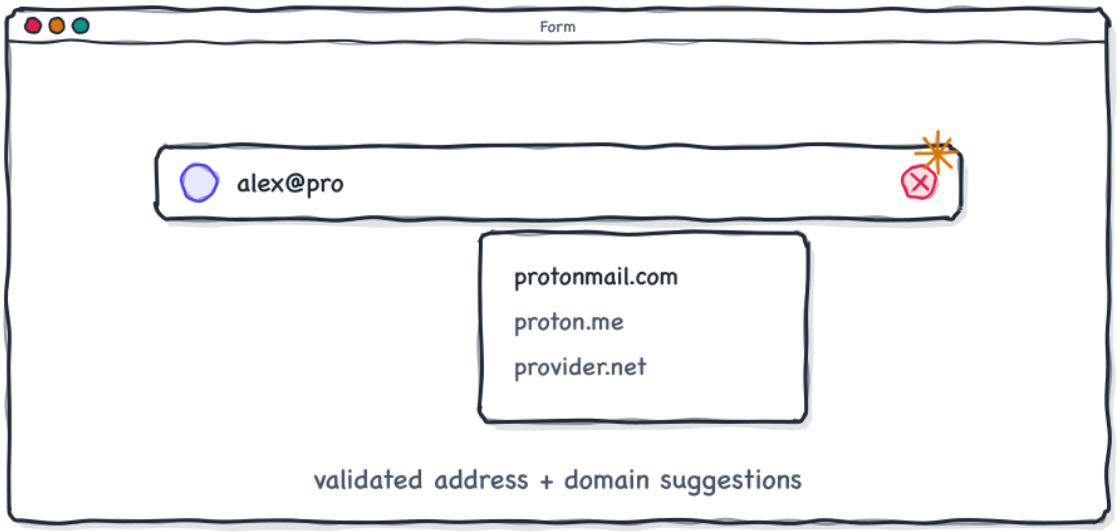
EmailField

Developer Manual

GemsFX • com.dlsc.gemsfx • JavaFX Custom Controls



A generated cartoon overview of EmailField.

EmailField wraps a CustomTextField with email validation, optional mail and warning icons, localized invalid text and a popup of matching email domains.

Table of Contents

1. Introduction

EmailField is a control for entering one email address, or a comma-separated list of addresses. It validates text with a pluggable `Predicate<String>`, exposes only valid values through its model properties and can suggest common domains after the at sign.

1.1 Key features

- Built-in domain list and auto completion popup after `@`.
- Optional mail icon on the left and validation icon on the right.
- Single-address and multiple-address modes.
- Read-only `valid` state with `:valid` and `:invalid` pseudo classes.
- Localized invalid tooltip text.

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

Create the control, set a prompt, decide whether the field is required and observe the validated output property.

```
EmailField field = new EmailField();
field.setPromptText("Email address");
field.setRequired(true);

field.validProperty().addListener((obs, old, valid) -> {
    if (valid) {
        System.out.println(field.getEmailAddress());
    }
});
```

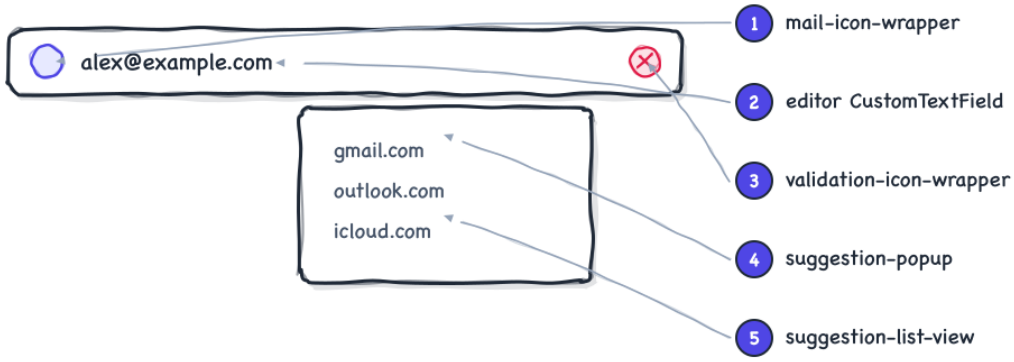
A complete single-address setup.

required=false lets an empty field be valid; required=true does not

Optional, invalid and valid states of the field.

3. Anatomy

The skin puts the internal `CustomTextField` into the control and installs the left mail icon, the right validation icon, the tooltip and the domain popup.



Parts of an EmailField.

Part	Node / value	Description
Root	`.email-field`	Control style class and validity pseudo classes.
Editor	`CustomTextField`	Receives focus and stores the user text.
Domain popup	`PopupControl`	Shows filtered domains while the editor is focused.
Suggestion list	`ListView<String>`	Uses `domainListCellFactory`.

4. Control API

4.1 Validation and values

Property	Type	Default	Description
emailValidator	ObjectProperty<Predicate<String>>	EmailValidator.getInstance()::isValid	Predicate used for each individual address.
required	BooleanProperty	false	Blank text is invalid only when required is true. Styleable.
valid	ReadOnlyBooleanProperty	derived	True when the current text satisfies the current mode and validator.
emailAddress	StringProperty	null	Valid single address, or null while invalid.
multipleEmailAddresses	ListProperty<String>	empty list	Valid addresses in multiple-address mode.

4.2 Editor and suggestions

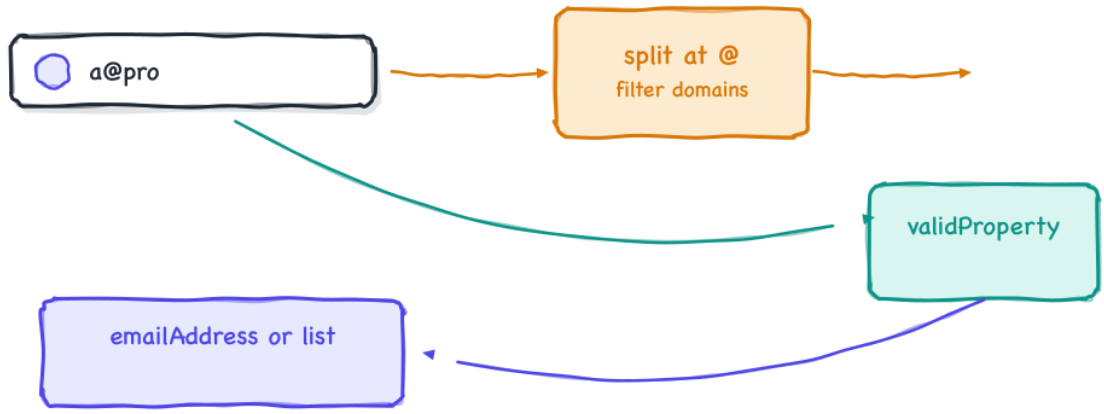
Property	Type	Default	Description
domainList	ListProperty<String>	gmail.com, yahoo.com, outlook.com, hotmail.com, icloud.com, aol.com, mail.com, protonmail.com, gmx.com, zoho.com, qq.com, 163.com, 126.com, yeah.net, msn.com, live.com, me.com	Known domains used by the popup.
autoDomainCompletionEnabled	BooleanProperty	true	Enables the popup. Styleable.
domainListCellFactory	ObjectProperty<Callback<ListView<String>, ListCell<String>>>	null	Custom cells for domain suggestions.
supportingMultipleAddresses	BooleanProperty	false	Comma-separated address mode. Styleable.
promptText	StringProperty	null	Bound to the internal editor prompt.

4.3 Icons and tooltip

Property	Type	Default	Description
showMailIcon	BooleanProperty	true	Shows the mail icon. Styleable.

Property	Type	Default	Description
showValidationIcon	BooleanProperty	true	Shows the validation icon when invalid. Styleable.
invalidText	StringProperty	Invalid email address.	Tooltip text installed on the validation icon.

5. Validation and suggestions



How typed text becomes suggestions and validated values.

The popup appears only when the editor is focused, auto completion is enabled, text contains an at sign, at least one domain starts with the typed suffix and no domain matches it exactly. Selecting a row replaces the suffix after the last at sign.

In multiple-address mode the text is tokenized by comma. Every trimmed token must pass `emailValidator`; the list property is updated with the valid tokens collected so far.

```
field.setSupportingMultipleAddresses(true);
field.setText("ada@example.com, grace@example.org");
ObservableList<String> addresses = field.getMultipleEmailAddresses();
```

6. Styling

The user agent stylesheet is `email-field.css`.

6.1 Style classes and pseudo classes

Selector	Description
`.email-field`	root control
`.email-field:valid`	valid state
`.email-field:invalid`	invalid state

Selector	Description
`.mail-icon-wrapper / .mail-icon`	left icon nodes
`.validation-icon-wrapper / .validation-icon`	right validation icon
`.suggestion-popup .content-pane .suggestion-list-view`	domain popup list

6.2 Styleable CSS properties

CSS property	Type	Default
`-fx-auto-domain-completion-enabled`	boolean	true
`-fx-required`	boolean	false
`-fx-show-mail-icon`	boolean	true
`-fx-show-validation-icon`	boolean	true
`-fx-supporting-multiple-addresses`	boolean	false

```
.email-field {
    -fx-required: true;
    -fx-show-mail-icon: false;
}

.email-field:invalid > .custom-text-field {
    -fx-border-color: red;
}
```

7. Localization

The invalid tooltip is loaded through `ResourceBundleManager` from `email-field.properties`.

Key	English text
`validation.invalid-email`	Invalid email address.

8. Accessibility

The constructor sets `AccessibleRole.TEXT_FIELD` and binds accessible text to `emailAddressProperty()`. Automatic accessible-text updates stop if application code sets accessible text itself.

9. Recipes

9.1 Replace the domain list

```
field.getDomainList().setAll("example.com", "example.org", "company.test");
```

9.2 Use a stricter validator

```
field.setEmailValidator(address -> address.endsWith("@example.com"));
```

9.3 Customize suggestion rows

```
field.setDomainListCellFactory(view -> new ListCell<>() {
    @Override protected void updateItem(String item, boolean empty) {
        super.updateItem(item, empty);
        setText(empty ? null : "@" + item);
    }
});
```

9.4 Checklist

1. Decide whether blank input is valid by setting `required`.
2. Use `emailAddress` only in single-address mode.
3. Use `multipleEmailAddresses` only in multiple-address mode.
4. Replace `invalidText` for product-specific wording.

10. See also

- Demo application: `com.dlsc.gemsfx.demo.EmailFieldApp` (run with `mvn javafx:run -f gemsfx-demo/pom.xml -Dmain.class=com.dlsc.gemsfx.demo.EmailFieldApp`)
- `SearchTextField` - another text-field control with an embedded popup.
- `CustomTextField` - the embedded editor used by the skin.
- API documentation: https://dlsc-software-consulting-gmbh.github.io/GemsFX/api/