Eroica / **javafx-fluent-theme**

Code | Issues 2 | Pull requests | Security | Insights

A custom theme for JavaFX following Windows 11's designs

Zlib license

91 stars **3** forks **4** watching Branches Activity
Tags

Public repository

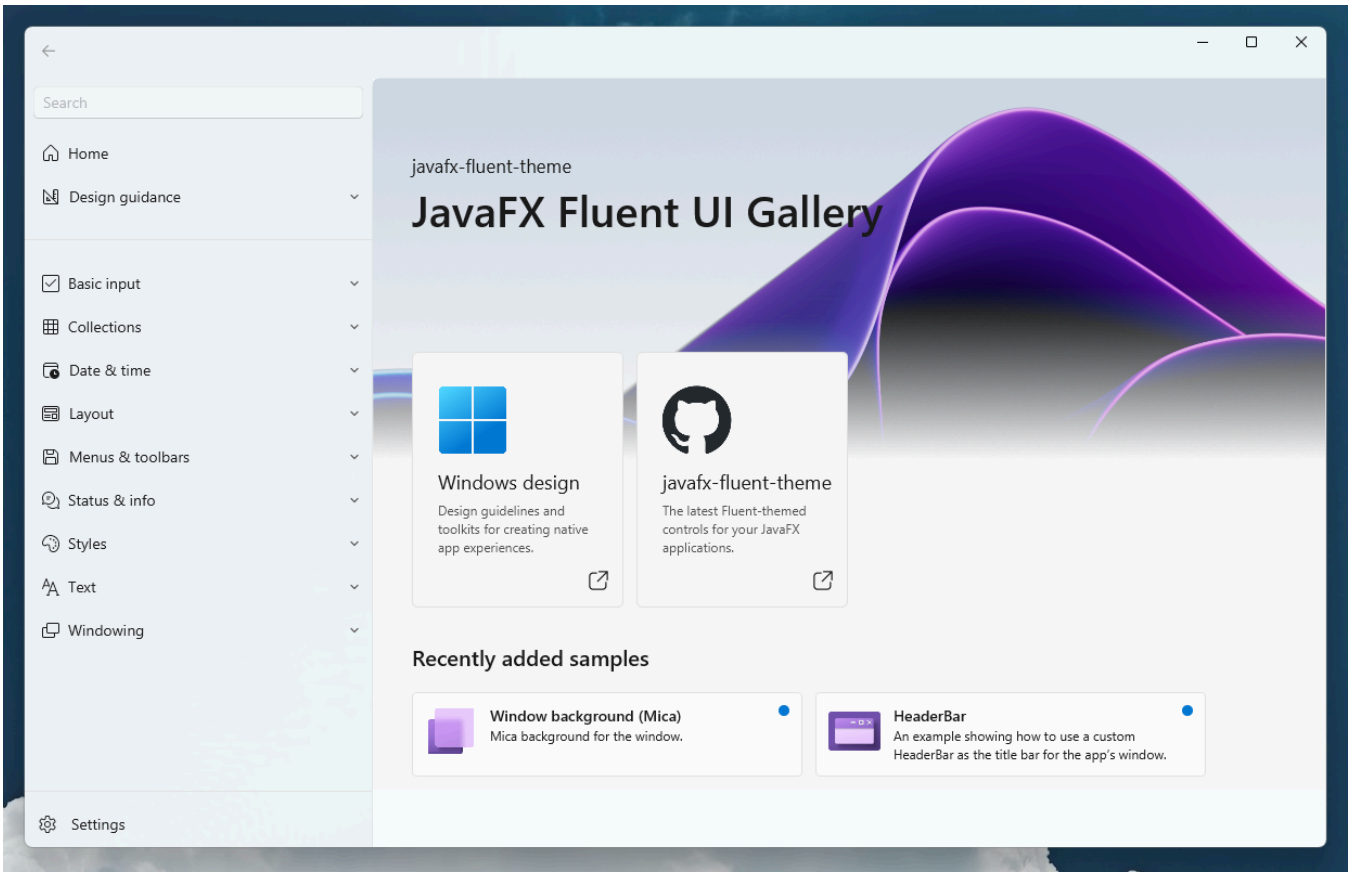
3 Branches 2 Tags Go to file t Go to file Add file Code

Eroica Update version to v2025.05 29a517f · 8 months ago

FluentLib	theme, demo: Update copyright years	last year
demo	Update version to v2025.05	8 months ago
docs	Initial commit	last year
gradle/wrapper	Update Gradle to 8.11	9 months ago
theme	Update dependencies	8 months ago
.editorconfig	Initial commit	last year
.gitignore	Initial commit	last year
CHANGELOG.md	Update CHANGELOG	last year
LICENSE	theme, demo: Update copyright years	last year
README.md	theme, demo: Update copyright years	last year
build.gradle.kts	Update version to v2025.05	8 months ago
gradle.properties	Initial commit	last year
gradlew	Update Gradle to 8.11	9 months ago
gradlew.bat	Update Gradle to 8.11	9 months ago
settings.gradle.kts	Initial commit	last year

README Zlib license

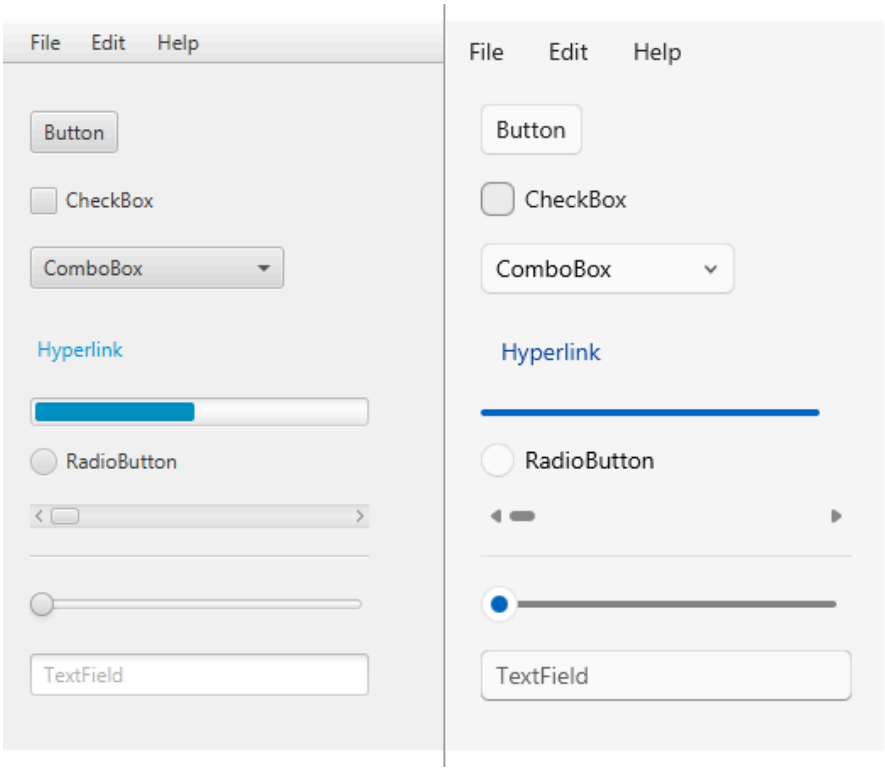
javafx-fluent-theme



This is a custom theme for JavaFX to make your application look like a Windows 11 (WinUI) program. It replaces `modena.css` with styles that follow Microsoft's *Fluent design* language. In addition, you can apply Windows 11's *Mica* effect to your window, or override Windows' default window title bar for complete client-side decorations.

Take a look at the example application, or see here how to achieve Windows' built-in effects.

See here for the list of styled controls, and to see which is still missing.



The theme and example application are written in Kotlin and built with Gradle. `FluentLib.dll` is required to call Windows' internal functions. It is created by a Visual Studio project located in `FluentLib/`.

```
theme/      # CSS theme and custom JavaFX controls which can be used as a library
FluentLib/  # Windows DLL to get access to Mica/title bar replacement
demo/       # demo application that uses the theme and DLL
```

The theme (currently, only a light theme is available) is built by SASS with files in `theme/src/main/css`.

Setup/Installation

The whole `theme` package includes CSS styling, custom skins, and a `FluentApp` abstract class that takes care of setting everything up.

- If you want to use the whole package, you need to build the `theme` package locally, and then include it in your dependencies.
- Alternatively, if you just want to change the look of some controls, you can copy the theme's CSS file into your project. However, some controls use custom skins to e.g. enable animations. See the list of controls here.

If you want to use Windows' internal functions, you need to load `FluentLib.dll` in your code. `FluentApp` takes care of that, but the DLL must be available to the application, which is usually where the process is being executed; in a Gradle environment, put it where your project's `build.gradle.kts` is located.

You can download a pre-built DLL from the Releases.

Usage

As a first step, run `FluentApp.initialize()` as soon as possible, before calling any JavaFX code. For example, in this `Main.kt` file:

```
fun main() {
    FluentApp.initialize()
    launch(YourApp::class.java)
}
```

This takes care of loading the FluentLib DLL, and does some checks so that Windows 11's Mica effect works (see Issues).

For the `YourApp` class, subclass `FluentApp`. It has a single abstract method `onCreateStage(Stage)`. In general, whatever you do in `start()`, you should now do in `onCreateStage`, with some exceptions:

- Don't change the Stage style using `initStyle`.
- Don't call `primaryStage.show()`. FluentApp takes care of that.

Manual setup

If you want to set up the theme manually without subclassing `FluentApp`, make sure to:

- Override `modena.css` as early as possible with `setUserAgentStylesheet("fluent-light.css")`
- If you want to use Mica/other effects, load the DLL as early as possible: `System.loadLibrary("FluentLib")`
- If you want to use Mica/other effects, set your stage's style to `StageStyle.UNIFIED`
- Only use methods from the FluentLib DLL **after** `primaryStage.show()`

Enabling Windows 11's Mica effect

Windows 11's "Mica" effect can be activated with this library's `Windows.setMicaFor(String, Boolean)` method (after `FluentLib.dll` is loaded). However, you need to make sure that:

- Your stage's style is `StageStyle.UNIFIED`
- In your JavaFX stage, there is no background color set wherever Mica should "shine through", e.g. any node should have `-fx-background-color: transparent`, and the `Scene` should be instantiated with `Scene(root, Color.TRANSPARENT)`
- Call `Windows.setMicaFor(String, Boolean)` providing your window title **after** the stage is `show()` n.
- You can enable or disable the effect using the same method.

Issues

While it's generally possible to get the Mica effect without using Microsoft's blessed ways, it seems it is a little unreliable under JavaFX **when using certain types of GPUs**. More info is available here: Graphic issues on certain GPUs

When you see a "glitched" Mica effect, try using software rendering first by setting this as early as possible (i.e. before calling JavaFX' `launch`):

```
System.setProperty("prism.order", "sw")
```

If Mica works with this, there is a good chance that you only need to enable a hidden flag **on non-AMD GPUs**. Remove the `prism.order` flag, and run this code as early as possible:

```
System.setProperty("prism.forceUploadingPainter", "true")
System.setProperty("javafx.animation.fullspeed", "true") // When on monitors >60Hz
```

`FluentApp.initialize()` takes care of that automatically, but my current check for AMD GPUs isn't very sophisticated. Feel free to raise any issues if e.g. the demo application looks "weird" on your setup (usually, a black background).

Removing Windows' default title bar

Windows' default title bar normally shows a small icon and the window title. You can remove this title bar and merge the content area with the window controls to use this "unused" space. Many macOS and GTK applications use a similar design which arguably looks a little more modern.

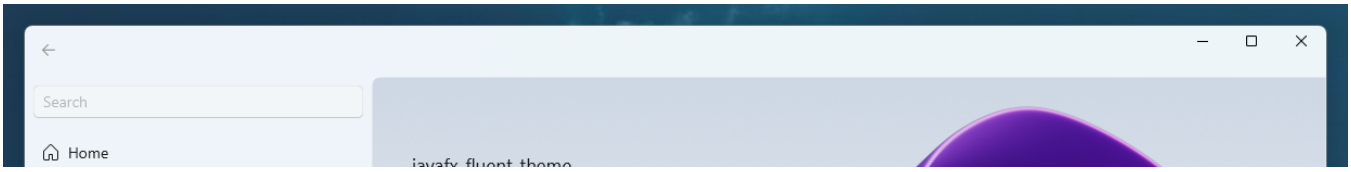
The Win32 equivalent is called `DwmExtendFrameIntoTitleBar`. (There is actually code in JavaFX that uses this, but it doesn't seem to work reliably.) With this theme, you get a more reliable way to achieve this window styling. Follow this (if you want to do it manually):

- The Stage's window style must be `StageStyle.UNIFIED`.
- **After** `primaryStage.show()`, call `Windows.setHeaderBarFor(String, Boolean)` providing your window's title to remove the title bar.
- The effect can be toggled back and forth using the same method.

Alternatively, take a look at subclassing `FluentApp`.

Depending on your scene background, the window controls at the top right might disappear. This is because the background is painted over these controls (although Windows will still react to click events there). To see window controls, make sure not to set any background or only use `transparent` where the window controls might be painted over.

Combined with the Mica effect from above (which requires a transparent background), the window controls will appear again, giving you a nice, Windows 11-styled application:



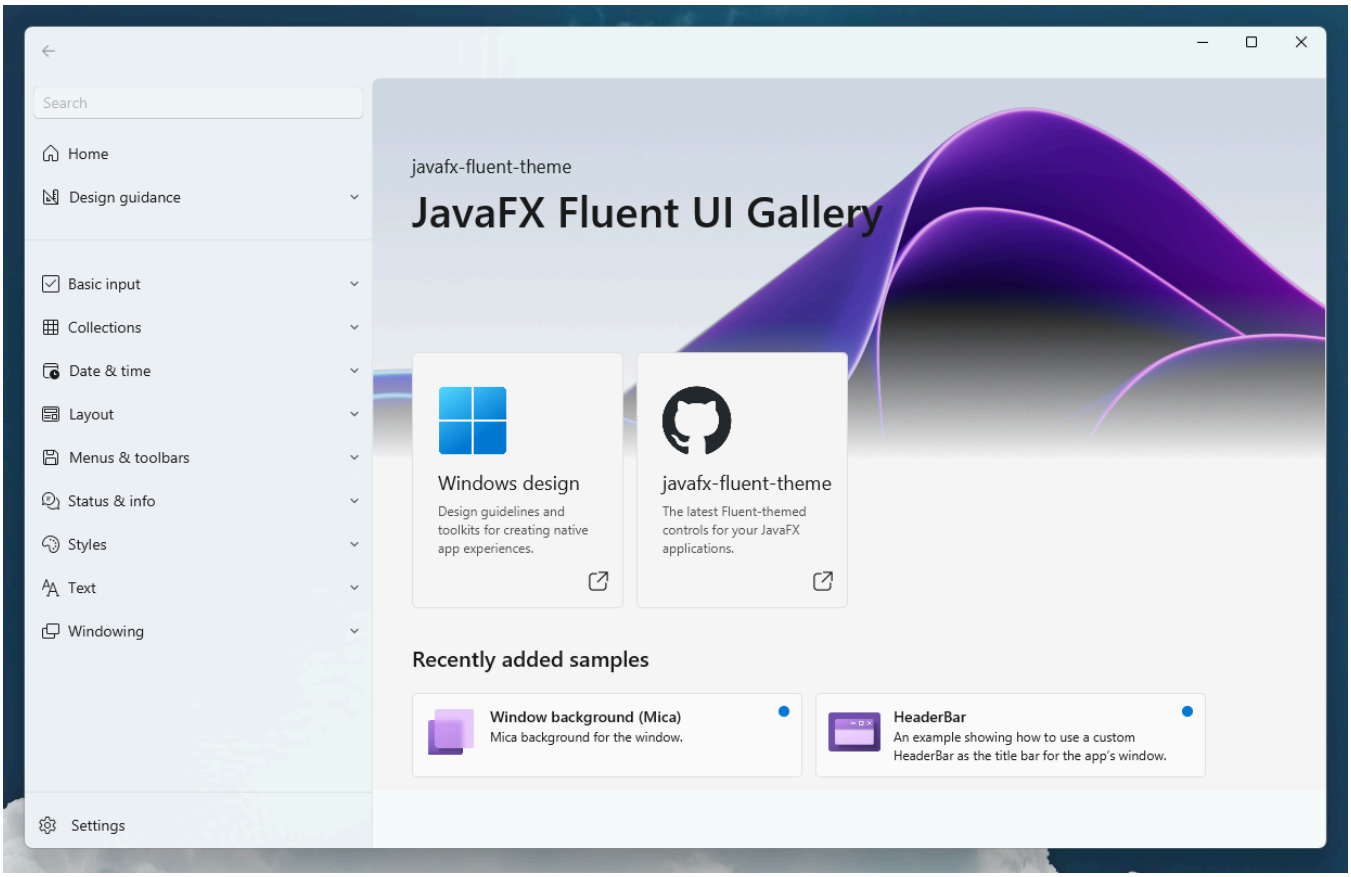
Here, the first node of my JavaFX scene is a custom `HeaderBar` node (which is basically an HBox) which just shows a button (the back arrow).

Without a title bar, you will lose the option to drag the window around with a mouse. Because of that, this library also provides a `DragPane` which you should put somewhere in your `HeaderBar` node. The nice thing about this is that it will capture Windows' native events, and e.g. trigger Windows' snap layouts when moving the window around.

An example header bar:

```
<HeaderBar>
    <padding>
        <Insets top="4.0" right="8.0" bottom="4.0" left="8.0"/>
    </padding>
    <Button text="&#xe830;" styleClass="borderless-button, back-button" disable="true"></Button>
    <DragPane HBox.hgrow="ALWAYS"/>
</HeaderBar>
```

Demo application



The `demo` application is roughly designed after "WinUI 3 Gallery". So far I mostly added the *controls* as opposed to containers like `StackPanel` / `StackPane`. Take a look at the list of controls below to see what is missing.

I renamed "System Backdrops (Mica/Acrylic)" (found under "Styles") to "Window background", and "TitleBar" (under "Windowing") to "HeaderBar".

Additionally, `Spinner` is put under "Basic input" instead of "Text" (where WinUI puts "NumberBox").

Note:

- The back button is only showcasing what you can put in place of a title bar, and not working yet.
- There is no application icon yet.

Building

The demo application requires `FluentLib.dll` in its working directory, so either build it from `FluentLib/`, or download a pre-built x64 DLL from the Releases, and put it under `demo/`.

Appendix

Design guidelines

`javafx-fluent-theme` will set grayscale antialiasing which is the default for Window applications using UWP or WinUI.

I mostly tried to follow the designs found in Microsoft's gallery application. When I first started my designs, most of them came from "WinUI 2 Gallery" and only later "WinUI 3 Gallery" caught up with Microsoft's latest designs, so there might still be some subtle differences between Microsoft's most current styles.

In my own designs, I try not to use any title case when labelling buttons or other controls, but only capitalize the first letter.

Non-goals

- Support on macOS or using JavaFX with GTK
- Compatibility with Windows 10 (Windows 11 replaces the previous *Segoe MDL2 Assets* with *Segoe Fluent Icons* which is used throughout this project, and which is not guaranteed to be available on Windows 10).

Table of JavaFX/WinUI controls

(Status: Done = styled & done; WIP = Added but not everything styled yet; Missing = Not added yet; Skipped = not planning on working on it)

JavaFX	WinUI	Status	CSS only?†
Button	Button	Done	YES
CheckBox	CheckBox	Done	YES
ChoiceBox	ComboBox	Done	YES
ColorPicker	ColorPicker	Missing	-
ComboBox	ComboBox	Done	NO‡
ContextMenu	(several)	Done	NO‡
DatePicker	CalendarDatePicker, DatePicker	Done	YES
HTMLEditor	-	Skipped	-
Hyperlink	HyperlinkButton	Done	YES

JavaFX	WinUI	Status	CSS only?†
ImageView	Image	Done	YES
Label	TextBlock	Done	YES
ListView	ListView	Done	NO‡
MediaView	MediaPlayerElement	Skipped	-
MenuBar	MenuBar	WIP	YES
MenuButon	DropDownButton	Done	YES
Pagination	-	Skipped	-
PasswordField	PasswordBox	Done	YES
ProgressBar	ProgressBar	Done	YES
ProgressIndicator	ProgressRing	Skipped	-
RadioButton	RadioButton	Done	YES
ScrollBar	ScrollBar	Done	YES
Separator	(AppBarSeparator)	Done	YES
Slider	Slider	Done	NO
Spinner	NumberBox	Done	NO
SplitMenuButton	SplitButton	Missing	-
TableView	DataGrid/DataTable	WIP	YES
TextArea	TextBox	Done	YES
TextField	TextBox	Done	YES
ToggleButton	ToggleButton	Done	YES
Tooltip	ToolTip	Done	NO‡
TreeTableView	DataGrid (I think)	Skipped	YES
TreeView	TreeView	WIP	-
WebView	WebView2	Skipped	-

†This means that no custom `Skin` is used to add any custom behavior. You could simply copy-paste the CSS attributes to get the design.

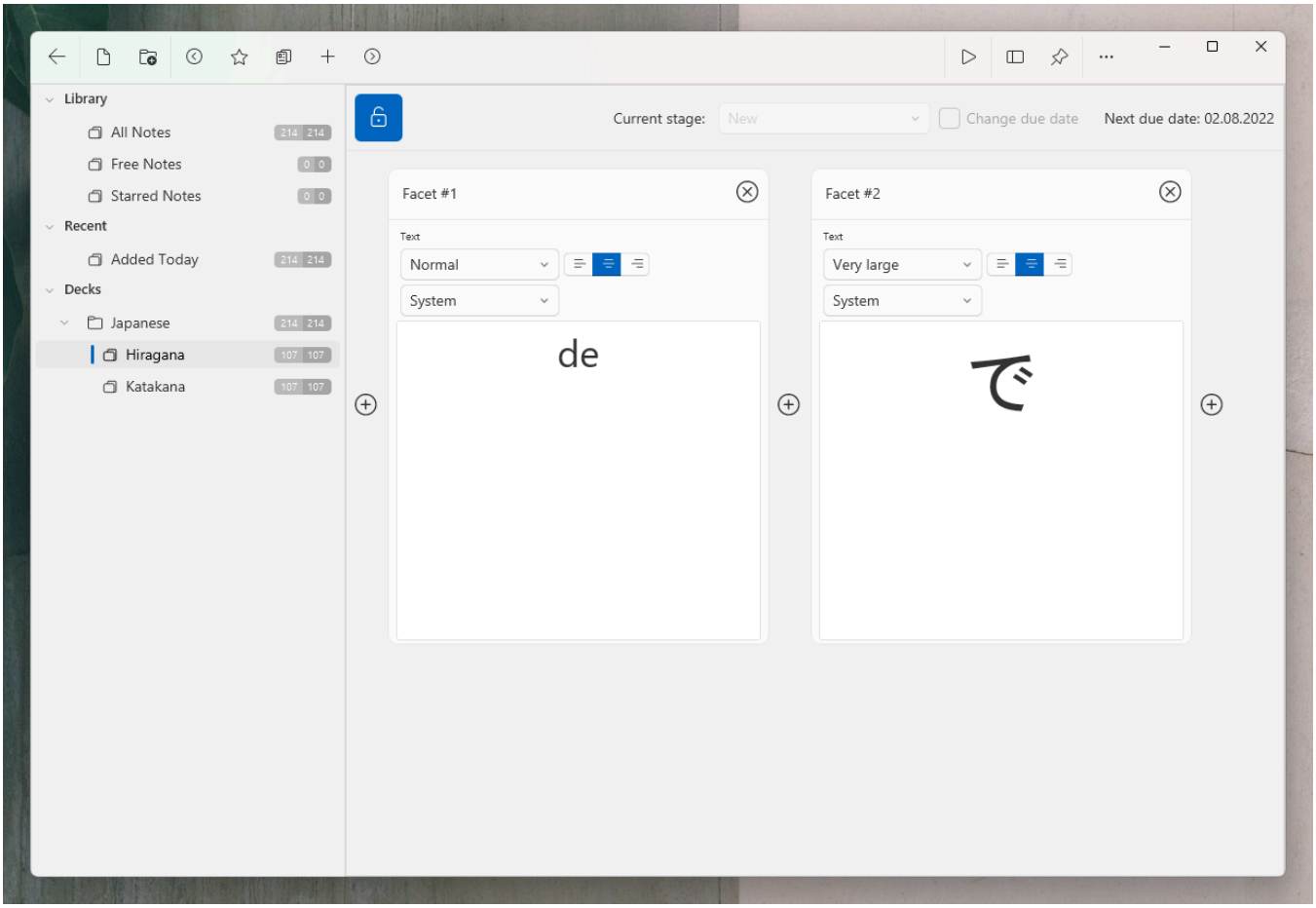
‡The control can be styled with CSS only, but in addition, you can instantiate `Fluent[Name]` to have "fluent animations".

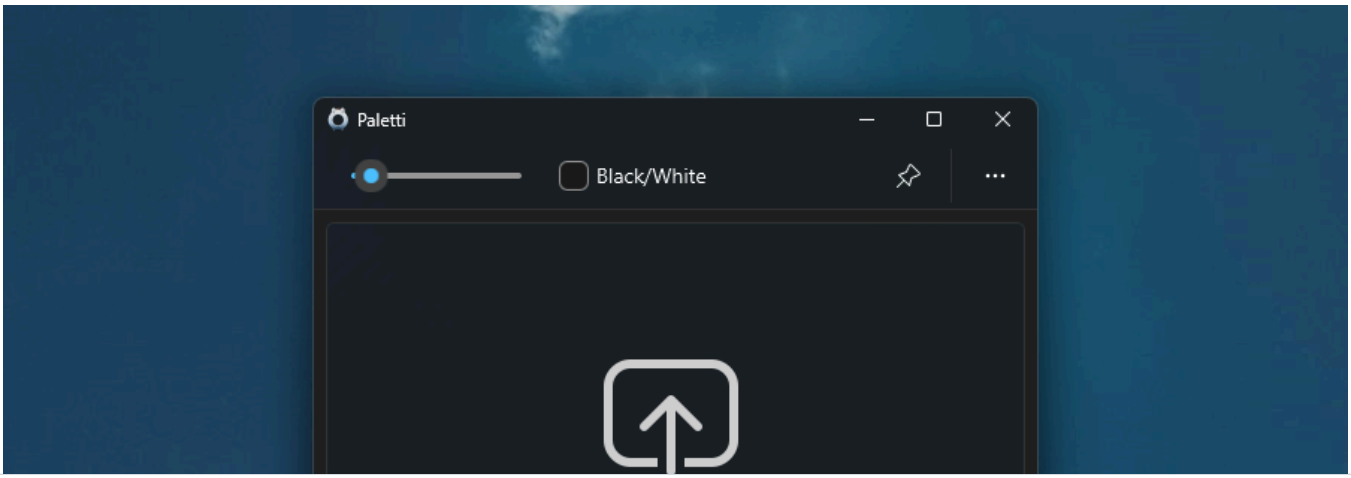
Additionally, the following container styles are available or currently in preparation:

JavaFX	WinUI	Status	CSS only?
Dialog	ContentDialog	WIP	YES
SplitPane	SplitView	Missing	-
TabPane	TabView	Missing	-
Expander	Expander	WIP	-
ScrollPane	ScrollView	WIP	-
ToolBar	CommandBar	Missing	-
InfoBar	InfoBar	Done	NO

Gallery

Here are my JavaFX applications that I styled with this theme:





Releases

2 tags

Languages

SCSS 67.3% Kotlin 20.1% C++ 11.6% C 1.0%