Slint UI Framework
Layout-System Dokumentation

Tutorial

26. Oktober 2025

Inhaltsverzeichnis

1 Einführung

Slint ist ein modernes UI-Framework für Rust, C++ und JavaScript. Es ermöglicht die Erstellung deklarativer Benutzeroberflächen mit einer eigenen Domain-Specific Language (DSL).

1.1 Grundstruktur

Ein Slint-Programm besteht aus zwei Hauptteilen:

- **UI-Definition**: Deklarative Beschreibung der Oberfläche in Slint-Syntax
- **Rust-Code**: Logik und Event-Handling

Listing 1: Minimales Beispiel

```
slint::slint! {
    export component MainWindow inherits Window {
        width: 400px;
        height: 300px;

        Text {
            text: "Hallo Welt!";
        }
    }
}
```

2 Layout-Systeme

Slint bietet verschiedene Layout-Container zur Positionierung von UI-Elementen.

2.1 VerticalLayout

Ordnet Elemente vertikal (von oben nach unten) an.

Listing 2: VerticalLayout Beispiel

```
VerticalLayout {
    padding: 20px;
    spacing: 10px;
    alignment: center;

    Text { text: "Erstes Element"; }
    Text { text: "Zweites Element"; }
    Text { text: "Drittes Element"; }
}
```

Wichtige Eigenschaften:

`padding`	Innenabstand um alle Kindelemente
`spacing`	Abstand zwischen den Kindelementen
`alignment`	Ausrichtung: `start`, `center`, `end`, `stretch`

2.2 HorizontalLayout

Ordnet Elemente horizontal (von links nach rechts) an.

Listing 3: HorizontalLayout Beispiel

```
HorizontalLayout {
    spacing: 15px;
    padding: 10px;

    Button { text: "Links"; }
    Button { text: "Mitte"; }
    Button { text: "Rechts"; }
}
```

Die Eigenschaften sind identisch zu `VerticalLayout`, nur die Ausrichtung ist horizontal.

2.3 GridLayout

Ermöglicht die Anordnung von Elementen in einem Raster mit Zeilen und Spalten.

Listing 4: GridLayout Beispiel

```
GridLayout {
    spacing: 5px;

    Button { text: "1,1"; row: 0; col: 0; }
    Button { text: "1,2"; row: 0; col: 1; }
    Button { text: "2,1"; row: 1; col: 0; }
    Button { text: "2,2"; row: 1; col: 1; colspan: 2; }
}
```

Positionierung:

`row`	Zeilenindex (beginnt bei 0)
`col`	Spaltenindex (beginnt bei 0)
`colspan`	Anzahl der Spalten, über die sich das Element erstreckt
`rowspan`	Anzahl der Zeilen, über die sich das Element erstreckt

3 Alignment (Ausrichtung)

Die `alignment`-Eigenschaft steuert, wie Elemente innerhalb ihres Containers ausgerichtet werden.

4 Absolute Positionierung

Neben Layouts können Elemente auch absolut positioniert werden.

Wert	Beschreibung
`start`	Elemente werden am Anfang ausgerichtet (oben bei VerticalLayout, links bei HorizontalLayout)
`center`	Elemente werden zentriert
`end`	Elemente werden am Ende ausgerichtet (unten bzw. rechts)
`stretch`	Elemente werden gestreckt, um den verfügbaren Platz auszufüllen

Tabelle 1: Alignment-Optionen

Listing 5: Absolute Positionierung

```
Rectangle {
    width: 200px;
    height: 150px;

    Rectangle {
        width: 50px;
        height: 50px;
        background: red;
        x: 10px;
        y: 20px;
    }

    Rectangle {
        width: 50px;
        height: 50px;
        background: blue;
        x: parent.width - self.width - 10px;
        y: parent.height - self.height - 10px;
    }
}
```

Relative Referenzen:

- `parent.width` / `parent.height`: Dimensionen des Elternelements
- `self.width` / `self.height`: Eigene Dimensionen
- `root.width`: Dimensionen des Hauptfensters

5 Properties und Bindings

Properties ermöglichen die Kommunikation zwischen UI und Rust-Code.

5.1 Property-Typen

Listing 6: Property-Deklarationen

```
export component MainWindow inherits Window {
    in property <string> input-text;       // Nur lesbar von Slint
    out property <int> result;             // Nur schreibbar von
       Slint
    in-out property <bool> is-active;      // Lesen und Schreiben
}
```

5.2 Bidirektionale Bindings

Der `<=>` Operator erstellt eine bidirektionale Verbindung zwischen Properties.

Listing 7: Bidirektionales Binding

```
in-out property <string> user-input: "Standard";

TextInput {
    text <=> user-input;  // Synchronisiert automatisch
}
```

6 Standard-Widgets

Slint bietet vorgefertigte Widgets, die importiert werden müssen.

Listing 8: Widget-Import

```
import { Button, CheckBox, SpinBox, LineEdit } from "std-widgets.
   slint";

export component MainWindow inherits Window {
    VerticalLayout {
        Button { text: "Klick mich"; }
        CheckBox { text: "Option aktivieren"; }
        SpinBox { value: 42; }
        LineEdit { placeholder-text: "Text eingeben..."; }
    }
}
```

Häufig verwendete Widgets:

- `Button`: Schaltfläche mit Click-Event
- `CheckBox`: Kontrollkästchen
- `LineEdit`: Einzeiliges Textfeld
- `SpinBox`: Numerische Eingabe mit Auf/Ab-Buttons
- `Slider`: Schieberegler
- `ComboBox`: Dropdown-Menü

7 Callbacks und Events

Callbacks ermöglichen die Reaktion auf Benutzerinteraktionen.

7.1 Inline-Callbacks

Listing 9: Inline-Callback

```
in-out property <int> counter: 0;

Button {
    text: "Erhöhen";
    clicked => {
        counter += 1;
    }
}
```

7.2 Rust-Callbacks

Listing 10: Callback von Rust aus

```
let ui = MainWindow::new().unwrap();

ui.on_button_clicked({
    let ui_weak = ui.as_weak();
    move || {
        let ui = ui_weak.unwrap();
        println!("Button wurde geklickt!");
        ui.set_counter(ui.get_counter() + 1);
    }
});

ui.run().unwrap();
```

8 Styling

Slint erlaubt umfangreiches Styling von UI-Elementen.

Listing 11: Styling-Beispiel

```
Rectangle {
    background: #3498db;
    border-width: 2px;
    border-color: #2980b9;
    border-radius: 8px;

    Text {
        text: "Gestylter Text";
        color: white;
        font-size: 18px;
```

```
        font-weight: 700;
    }
}
```

Wichtige Style-Eigenschaften:

- `background`: Hintergrundfarbe (Hex, RGB oder benannt)
- `border-width`, `border-color`, `border-radius`: Rahmen
- `font-size`, `font-weight`: Textformatierung
- `horizontal-alignment`, `vertical-alignment`: Textausrichtung

9 Best Practices

1. **Komponenten modular gestalten**: Wiederverwendbare Komponenten erstellen
2. **Layouts bevorzugen**: Absolute Positionierung nur wenn nötig
3. **Properties verwenden**: Trennung von UI und Logik
4. **Responsive Design**: Relative Größen statt fixer Pixel-Werte
5. **Widgets importieren**: Standard-Widgets explizit importieren

10 Zusammenfassung

Slint bietet ein leistungsfähiges und intuitives Layout-System:

- Drei Hauptlayouts: Vertical, Horizontal, Grid
- Flexible Ausrichtungsoptionen mit `alignment`
- Properties für UI-Logik-Kommunikation
- Umfangreiche Standard-Widget-Bibliothek
- Deklarative Syntax für sauberen, wartbaren Code