Slint Learn

- author: syf20020816@outlook.com•

updateDate: 20230902

- github: https://github.com/syf20020816/slint_learn

How to study this document

learning sequence

The learning sequence in this document is basically top-to-bottom, in order from front to back Know that you have encountered this ☒ sign, which will lead you to a change in the learning sequence!

symbolise

- ☒: indicates that the learning sequence will

be sent to change (probably) or prompts• 👎:

indicates that it is not recommended

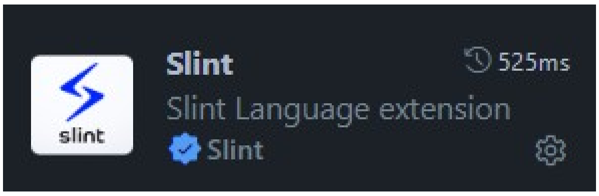
instructions

There are some differences between this document and the official document, and it is not a translation of the official document, the content of the official document may have some discrepancies with the content of this document (terminology, terminology designation, labelling, etc.), perhaps you can find the correspondence in the following table.

official (relating a government office)	This article was renamed
Builtin Elements	General Components
colour	Color.
brush	Color.brush
physical-length	Length.phx
length	Length.size
relative-font-size	Length.rem
Builtin Elements	General Components
Builtin Callbacks	life cycle

Slint With VSCode

I recommend using VSCode for Slint development, VSCode provides plugins that are very Slint friendly, the plugins are as follows:

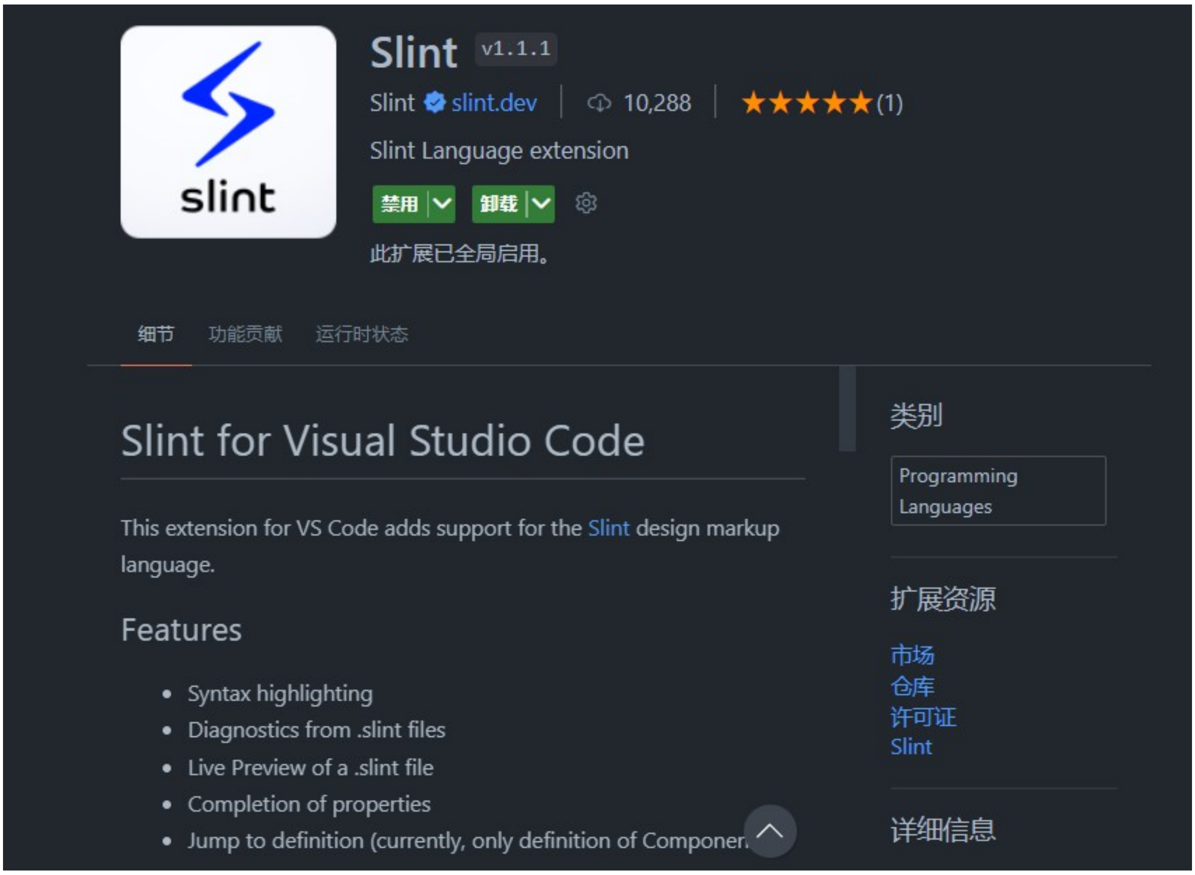


Slint

525ms

Slint Language extension

Slint



Slint for Visual Studio Code

This extension for VS Code adds support for the Slint design markup language.

Features

- Syntax highlighting
- Diagnostics from .slint files
- Live Preview of a .slint file
- Completion of properties
- Jump to definition (currently, only definition of Componen

Slint With Rust

dependencies

```
cargo add slint
```

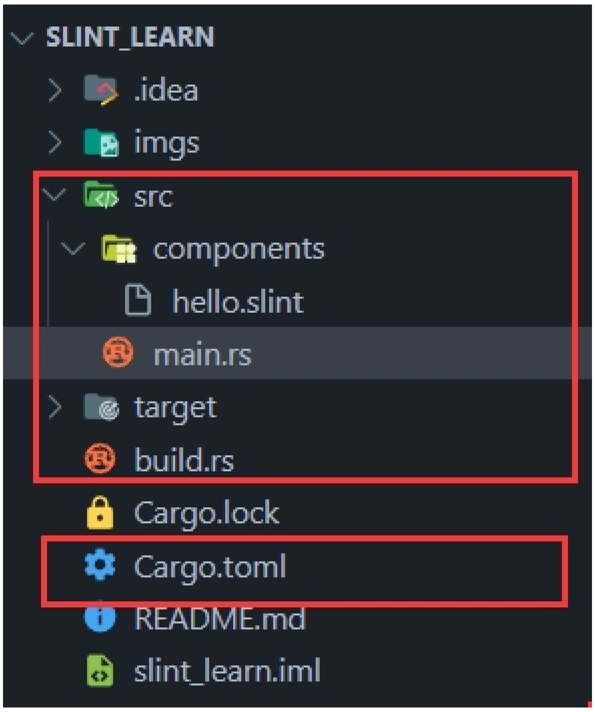
Defining Macros

Used to define a component so that it can then be
written in the rs file

```
slint::slint!{}
```

Slint and Rust Separation

In fact, the recommended and better way would be to separate the slint file from the rs file.



```
SLINT_LEARN
  .idea
  imgs
  src
    components
      hello.slint
    main.rs
  target
  build.rs
  Cargo.lock
  Cargo.toml
  README.md
  slint_learn.iml
```

1.Add build dependencies (**slint-build**)

```
[package]
name =
"slint_learn"
version = "0.1.0"
edition = "2021"
[dependencies]
slint =
"1.1.1"

// Add compilation
dependencies
[build-dependencies]
slint-build = "1.1.1"
```

2.Preparation of **slint** files

```
export component MainWindow inherits Window {
  Text{
    text: "Hello Slint";
   }
}
```

3.Write build.rs

```
fn main() {
  slint_build::compile("src/components/hello.slint").unwrap();
}
```

4.Write main.rs

```rust
//Introduction of modules
slint::include modules!();

fn main() {
    MainWindow::new().unwrap().run().
}
```

General Components

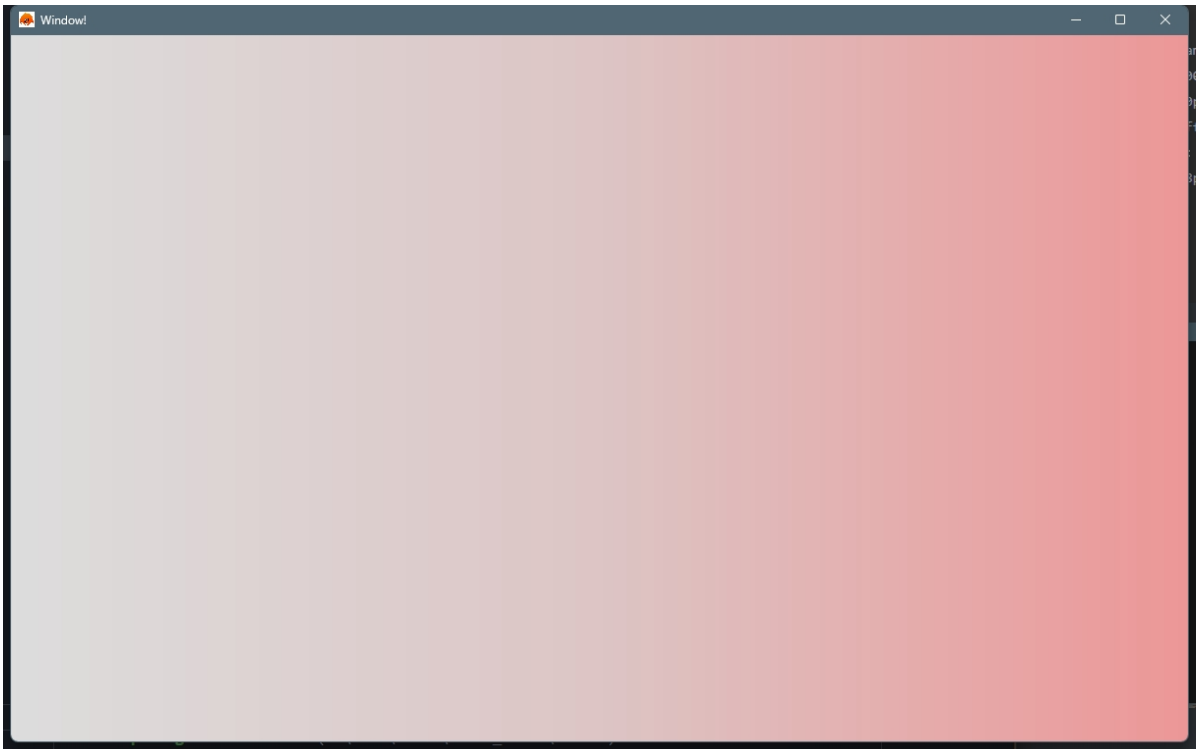
Components need to `be` declared using `comment` and exported using `export`.

Main Form Window

Forms need to inherit (inherits) Window

example

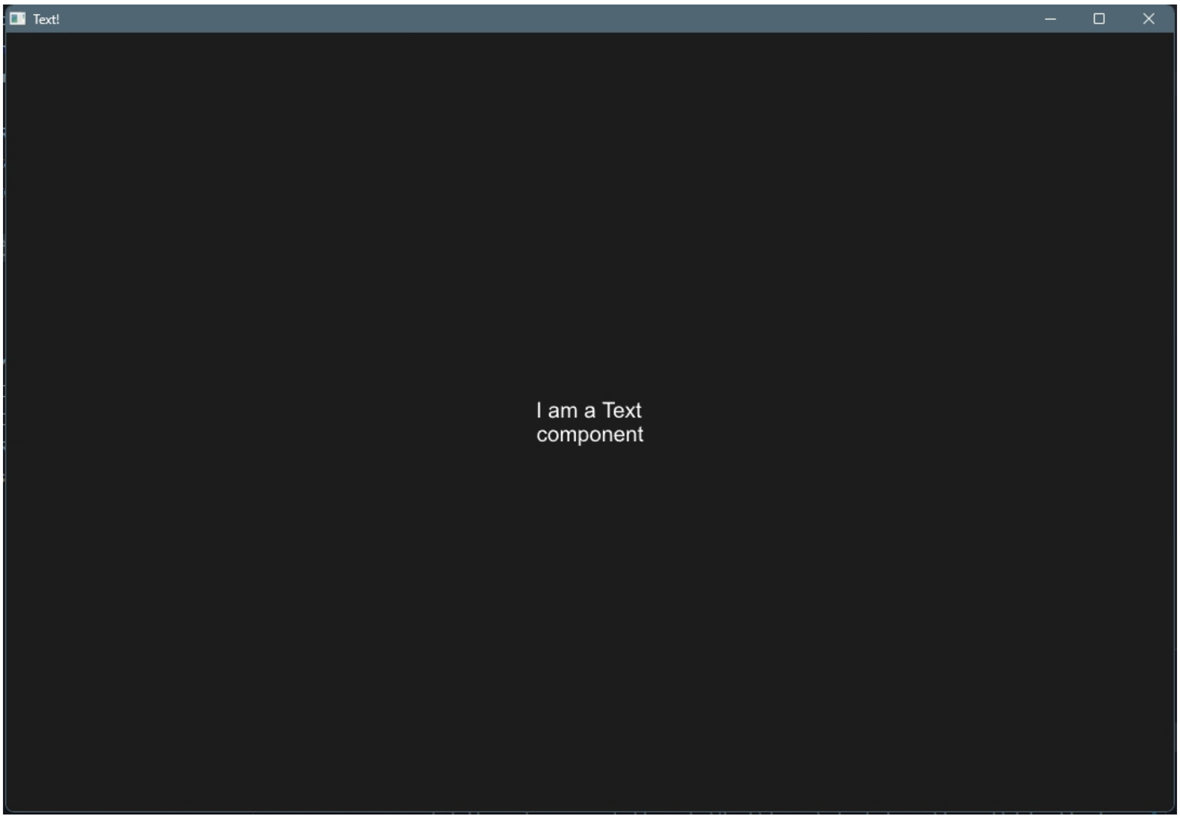
```
export component MainWindow inherits Window {
  default-font-family: "Helvetica,Verdana,Arial,sans-serif";
  default-font-size: 16px;
  default-font-weight: 700;
  background: @linear-gradient(90deg,#ddd 0%,#ddc5c5 50%,#ed9797 100%);
  always-on-top: true;
  no-frame: false;
  icon: @image-url("... /... /imgs/rust.png");;
  title: "Window!";
  height: 720px;
  width: 1200px;
}
```



Text

example

```
export component MainWindow inherits Window {
  height: 720px;
  width: 1080px;
  title: "Text!";
  Text {
    text: "I am a Text component";
    height: 200px;
    width: 100px;
    // Text line feed
    wrap: word-wrap;
    colour: #fff;
    font-size: 20px;
    padding: 8px;
    letter-spacing: 2px;
    //Horizontal alignment
    horizontal-alignment:centre;
    //Vertical alignment
    vertical-alignment: centre;
    overflow: elide;
   }
}
```

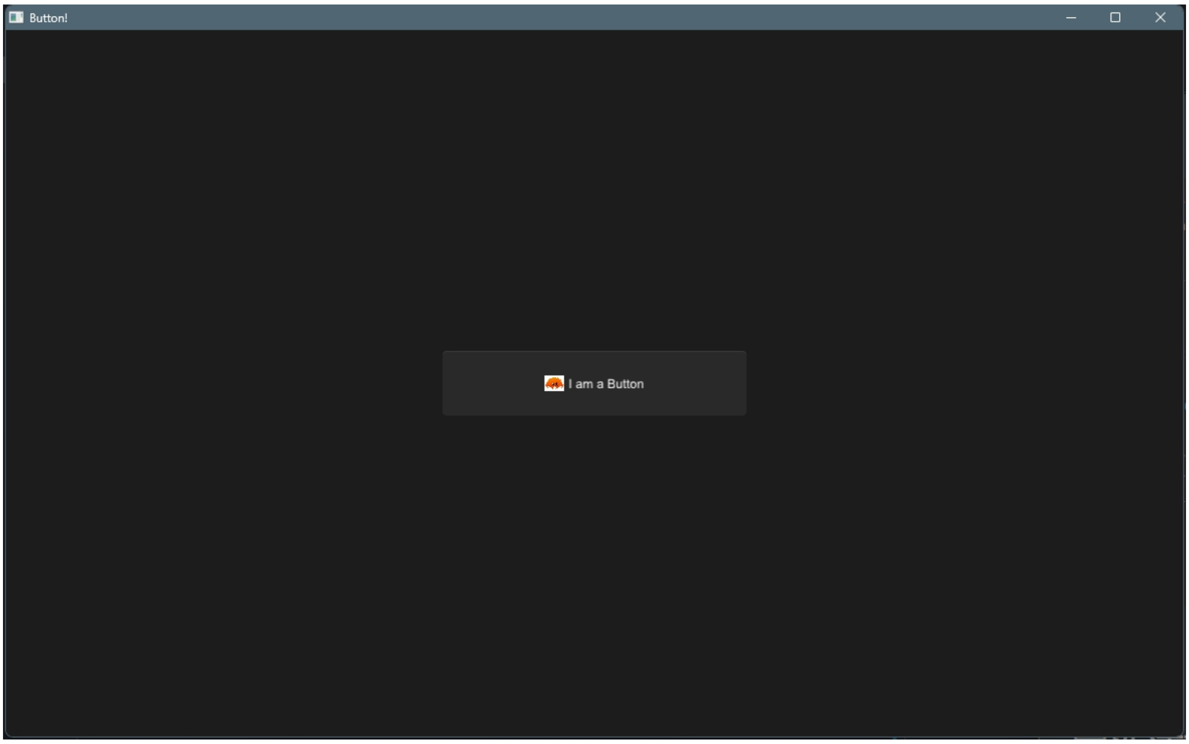


I am a Text component

Button

example

```
import { Button } from "std-widgets.slint" ;
export component MainWindow inherits Window {
  height: 720px;
  width: 1200px;
  title: "Button!";
  Button {
    height: 66px;
    width: 310px;
    icon: @image-url("... /... /imgs/rust.png");;
    text: "I am a Button";
    clicked => {
      self.text = "Clicked!";
      self.width = 360px;
     }
   }
}
```



functions

event name	instructions
clicked	button click event

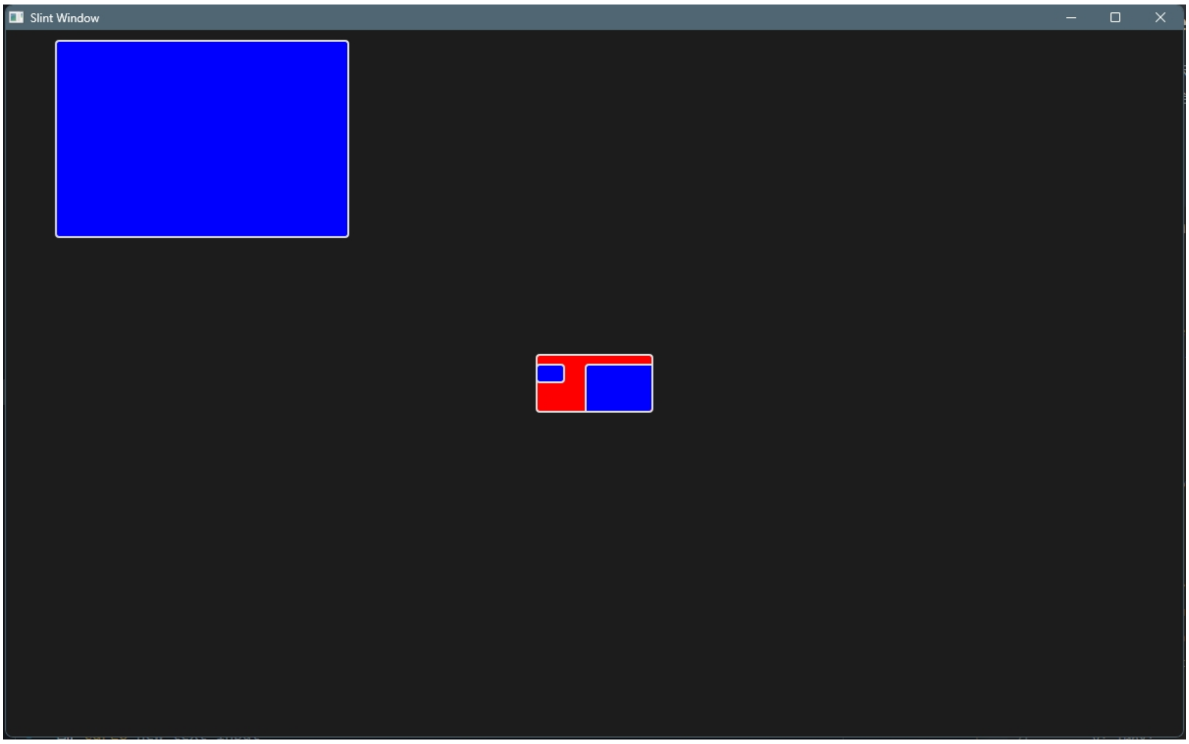
```
Button {
  height:
  66px; width:
  310px;
  text: "I am a Button";
  clicked => {
    self.text =
    "Clicked!"; self.width
    = 360px;
  }
}
```

Rectangle box element

A Rectangle is simply an empty item that does not display any content. Rectangles can be drawn on the screen by setting a colour or configuring a border. When not part of a layout, its width and height default to 100% of the parent element.

example

```
export component MainWindow inherits Window {
  height: 720px;
  width: 1200px;
  Rectangle {
    background: red;
    border-color: #ddd;
    border-radius: 4px;
    border-width: 2px;
    height: 60px;
    width: 120px;
    //like overflow clip indicates if the container is out of range.
    clip: true;
    Rectangle {
      background: blue;
      border-color: #ddd;
      border-radius: 4px;
      border-width: 2px;
      height: 20px;
      width: 30px;
      x: 0px;
      y: 10px;
    }
    Rectangle {
      background: blue;
      border-color: #ddd;
      border-radius: 4px;
      border-width: 2px;
      height: 202px;
      width: 300px;
      x: 50px;
      y: 10px;
    }
  }
  Rectangle {
    background: blue;
    border-color: #ddd;
    border-radius: 4px;
    border-width: 2px;
    height: 202px;
    width: 300px;
    x: 50px;
    y: 10px;
  }
}
```

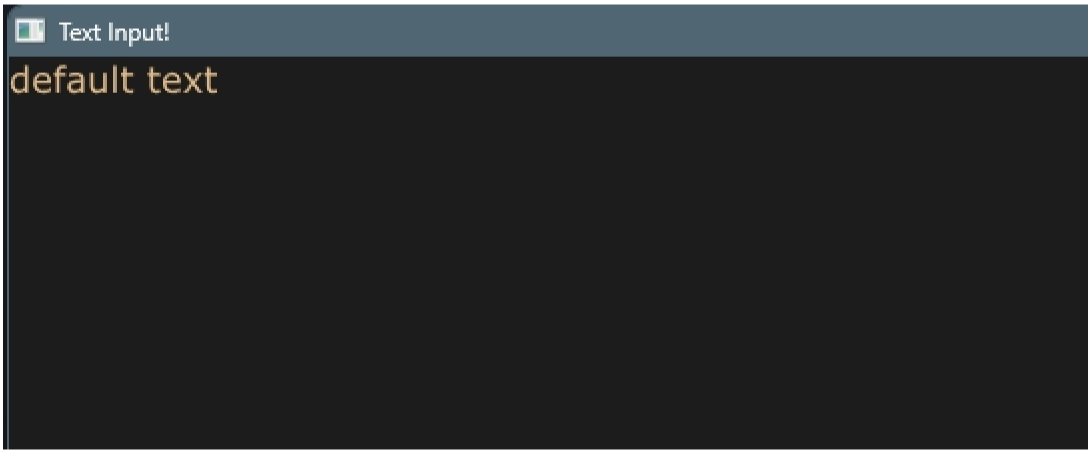


TextInput

This is a low-level input box that will directly use the parent container's width and height, and cannot set its own

example

```
export component MainWindow inherits Window {
  height: 720px;
  width: 1200px;
  title: "Text Input!";
  TextInput {
    colour: burlywood.
    font-family: "Verdana".
    font-size: 18px;
    font-weight: 400;
    horizontal-alignment: left;
    input-type: text;
    read-only: false;
    selection-background-color: blue;
    selection-foreground-color: red;
    single-line: false;
    wrap: word-wrap;
    text: "default text";
    text-cursor-width:8px;
    }
}
```

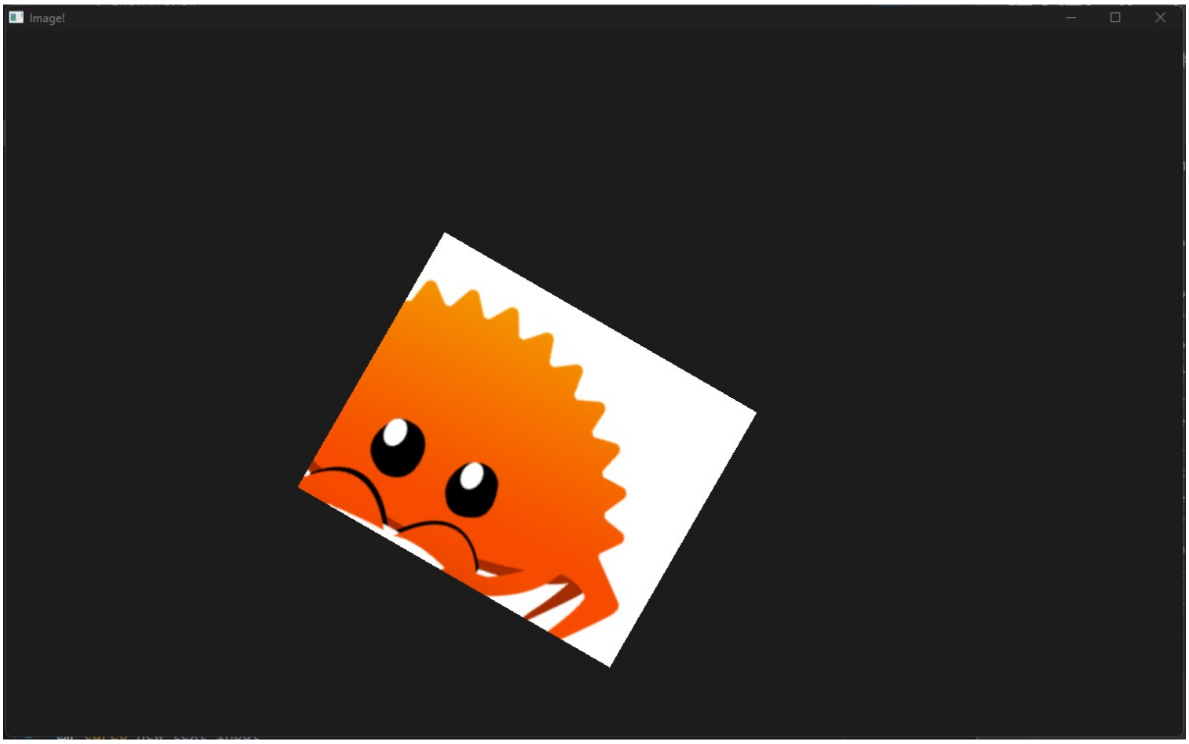


default text

Image

example

```
export component MainWindow inherits Window {
  height: 720px;
  width: 1200px;
  title: "Image!";
  Image {
    source: @image-url("... /... /imgs/rust.png");
    image-fit:fill;
    image-rendering: smooth;
    //Set the centre of rotation
    rotation-origin-x: 23px;
    rotation-origin-y: 56px;
    rotation-angle: 30deg;
    source-clip-height: 200;
    source-clip-x: 100;
    height: 300px;
   }
}
```

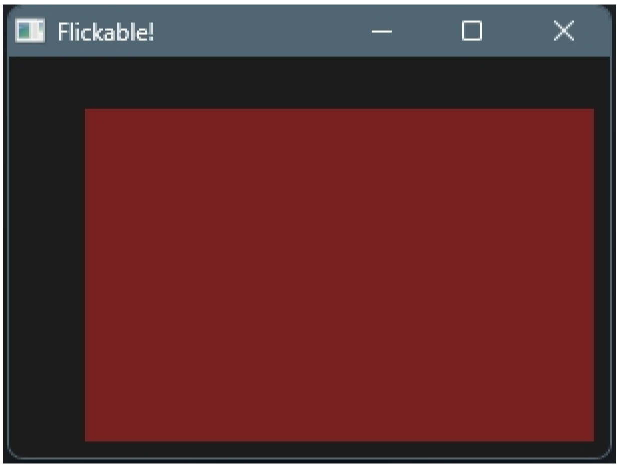


Scroll Window **Flickable**

Instead of producing a scrollable window, Flickable adds scrollable containers to an element. Because he is for the parent container, the child container is scrollable, not making the parent container scrollable

example

```
export component MainWindow inherits Dialog {
  height: 200px;
  width: 300px;
  title: "Flickable!";
  Flickable {
      interactive: true;
      viewport-height: 300px;
      viewport-width: 400px;
      viewport-x: 0px;
      viewport-y: 0px;
     Rectangle {
      height: 200px;
      width: 300px;
      background: #792121;
      }
   }
}
```



GridLayout

Elements that use a grid layout are added to the

- col: column•

  row: row

- colspan: Columns

  as a percentage•

  rowspan: Rows as a

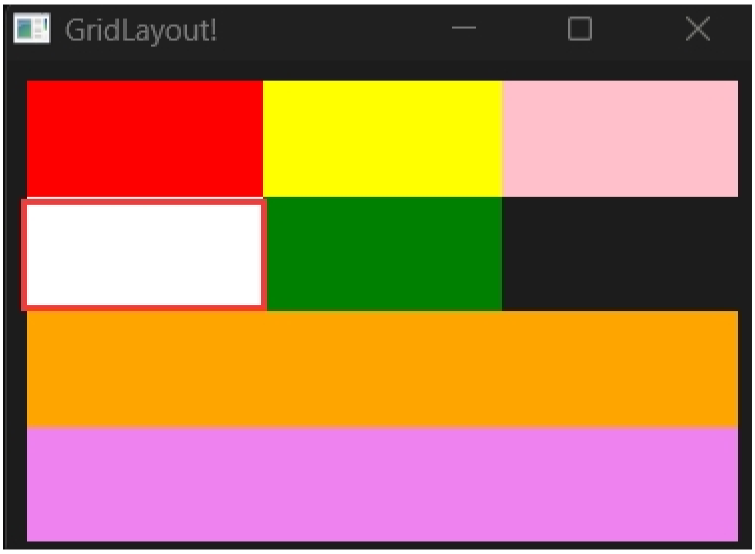
  percentage

These four attributes, which are typically used to control the

You can also use `Row{}` for row declarations to place all elements on the same row under a single Row.

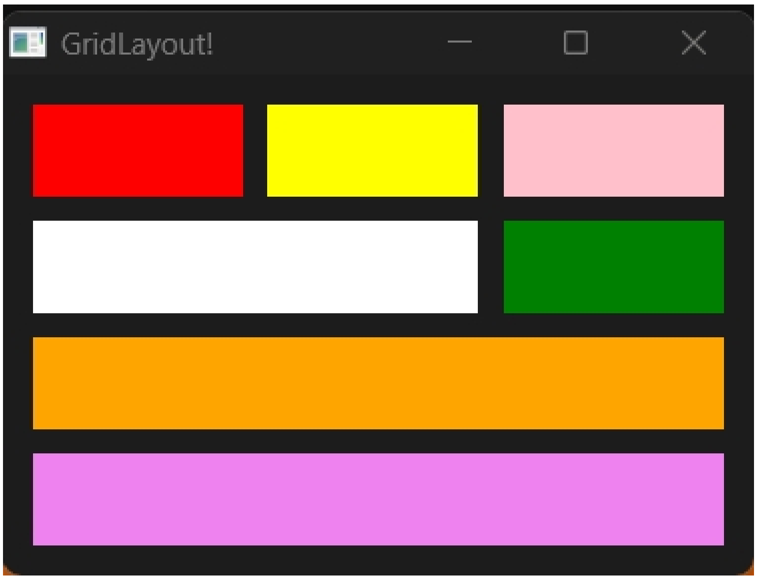
> Worth noting: personally, I think there are some problems with Slint's grid layout, and I

look forward to fixing it in subsequent releases (column occupancy and where columns are located need to be strongly specified, weak specification can lead to speculation errors)

> The questions are as follows:
>
> Here the white Rectangle column in the second row should be 2 but it shows 1, after stacking check the other half of the white box is covered by the green one, so the weak specification can't infer that the green box should actually be in the 3rd column, you need to specify `col:2` manually.
>
> 
>

example

```
export component MainWindow inherits Dialog {
  height: 200px;
  width: 300px;
  title: "GridLayout!";
  GridLayout {
    spacing: 10px;
    padding: 4px;
    // Row declaration using Row
    Row{
      Rectangle { background: red; }
      Rectangle { background: yellow;}
      Rectangle { background: pink; }
     }
    Row{
      Rectangle { background: white; colspan: 2; }
      Rectangle { background: green; colspan: 1; col: 2;}
     }
    Rectangle { background: violet; colspan: 3;row:3;}
    Rectangle { background: orange; colspan: 3;row:2;}
   }
}
```



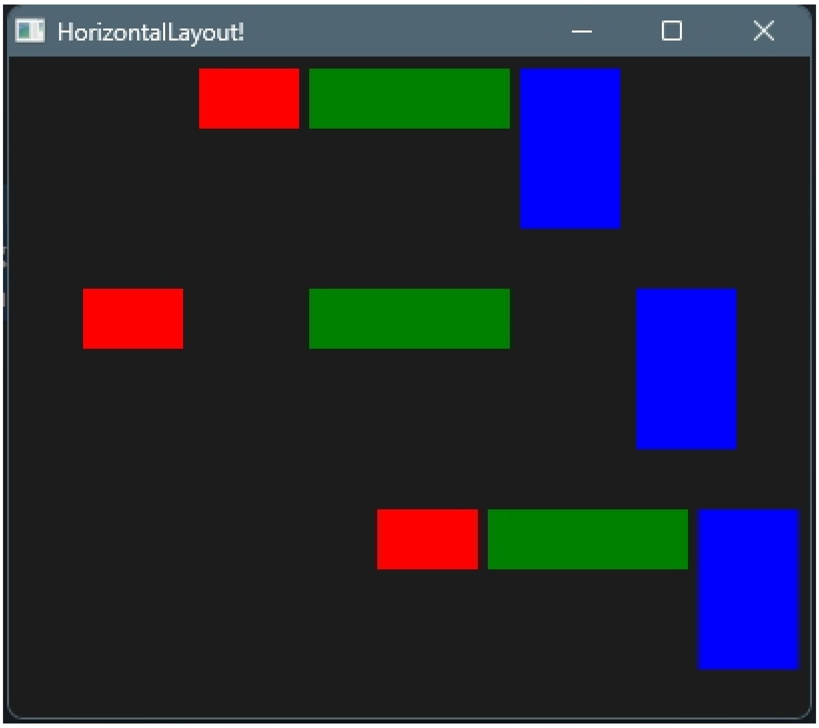
HorizontalLayout | VerticalLayout

The alignment of elements is controlled by the alignment attribute. Horizontal and vertical layouts are often used in combination with each other

HorizontalLayout

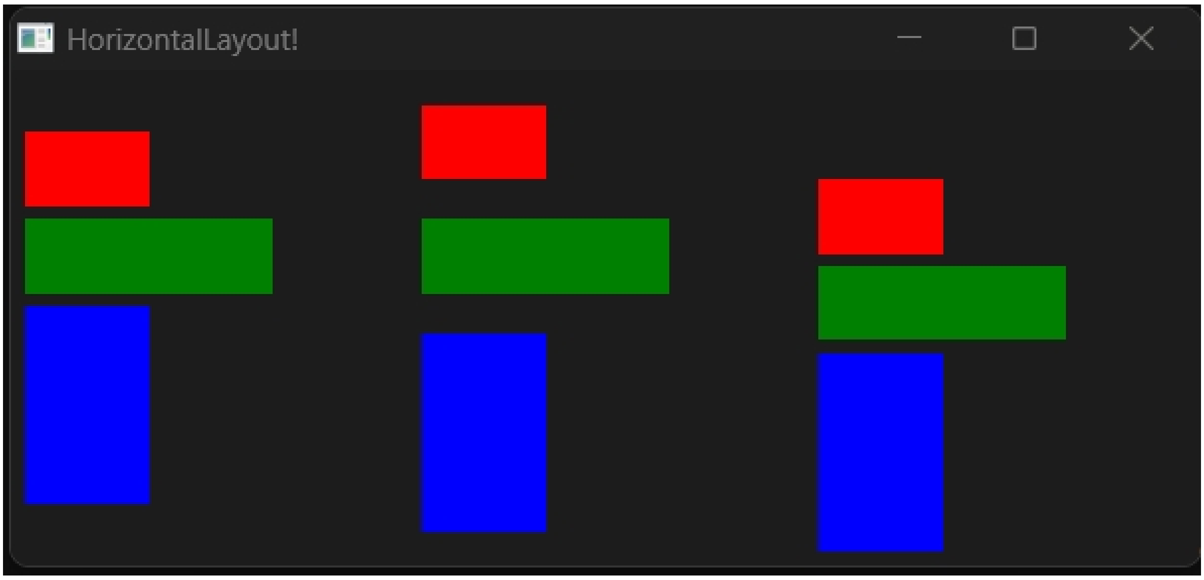
Horizontal layout means that the elements are all arranged on the same line

```
export component MainWindow inherits Window {
  height: 330px;
  width: 400px;
  title: "HorizontalLayout!".
  HorizontalLayout {
    spacing: 5px;
    padding: 6px;
    height: 100px;
    width: 400px;
    x: 0px;
    y: 0px;
    alignment: centre;
    Rectangle {background: red;height: 30px;width: 50px;}
    Rectangle {background: green; height: 30px;width: 100px;}
    Rectangle {background: blue; height: 80px;width: 50px;}
   }
  HorizontalLayout {
    spacing: 5px;
    padding: 6px;
    height: 100px;
    width: parent.width;
    x: 0px;
    y: 110px;
    alignment: space-around;
    Rectangle {background: red;height: 30px;width: 50px;}
    Rectangle {background: green; height: 30px;width: 100px;}
    Rectangle {background: blue; height: 80px;width: 50px;}
   }
  HorizontalLayout {
    spacing: 5px;
    padding: 6px;
    height: 100px;
    width: parent.width;
    x: 0px;
    y: 220px;
    alignment: end;
    Rectangle {background: red; height: 30px;width: 50px;}
    Rectangle {background: green; height: 30px;width:
    100px;}
  } Rectangle {background: blue; height: 80px;width: 50px;}
}
```



VerticalLayout

```
export component MainWindow inherits Window {
  height: 200px;
  width: 480px;
  title: "HorizontalLayout!".
  VerticalLayout {
    spacing: 5px;
    padding: 6px;
    height: root.height;
    width: 150px;
    x: 0px;
    y: 0px;
    alignment: centre;
    Rectangle {background: red;height: 30px;width: 50px;}
    Rectangle {background: green; height: 30px;width: 100px;}
    Rectangle {background: blue; height: 80px;width: 50px;}
   }
  VerticalLayout {
    spacing: 5px;
    padding: 6px;
    height: root.height;
    width: 150px;
    x: 160px;
    y: 0px;
    alignment: space-around;
    Rectangle {background: red; height: 30px;width: 50px;}
    Rectangle {background: green; height: 30px;width:
    100px;} Rectangle {background: blue; height: 80px;width:
    50px;} Rectangle {background: blue; height: 80px;width:
    50px;} height: 80px;width: 50px;}
  }
  VerticalLayout {
    spacing: 5px;
    padding: 6px;
    height: root.height;
    width: 150px;
    x:
    320px;
    y: 0px.
    alignment: end;
    Rectangle {background: red; height: 30px;width: 50px;}
}   Rectangle {background: green; height: 30px;width:
    100px;} Rectangle {background: blue; height: 80px;width:
    50px;} Rectangle {background: blue; height: 80px;width:
    50px;} height: 80px;width: 50px;}
  }
```



Drawing Board Path

Drawing shapes by tracing, which I call

the drawing board• Drawing using

Slint's path command

- Drawing with SVG's path command

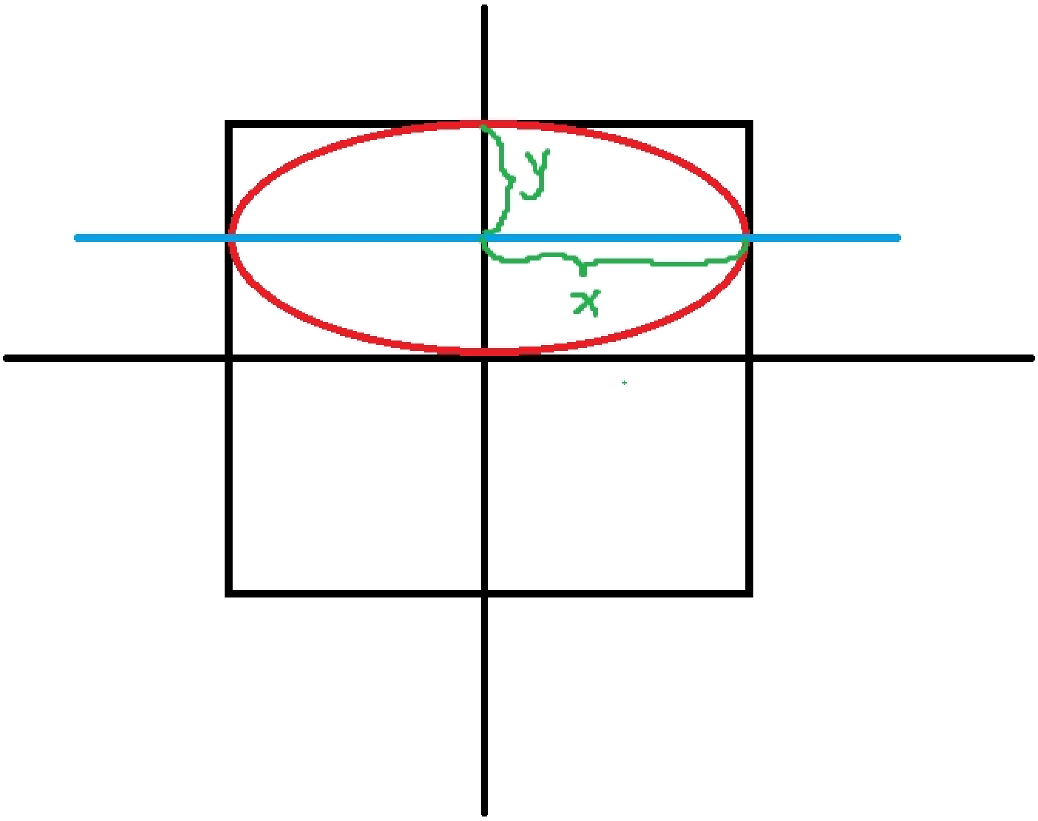
SVG Path Command and Slint Path Command

If the command letter is uppercase, e.g. M, the coordinate position is absolute; if the command letter is lowercase, e.g. m, the coordinate position is relative.

Declare it using the commands attribute (written as a function below to aid understanding):

```
commands: "M ..."
```

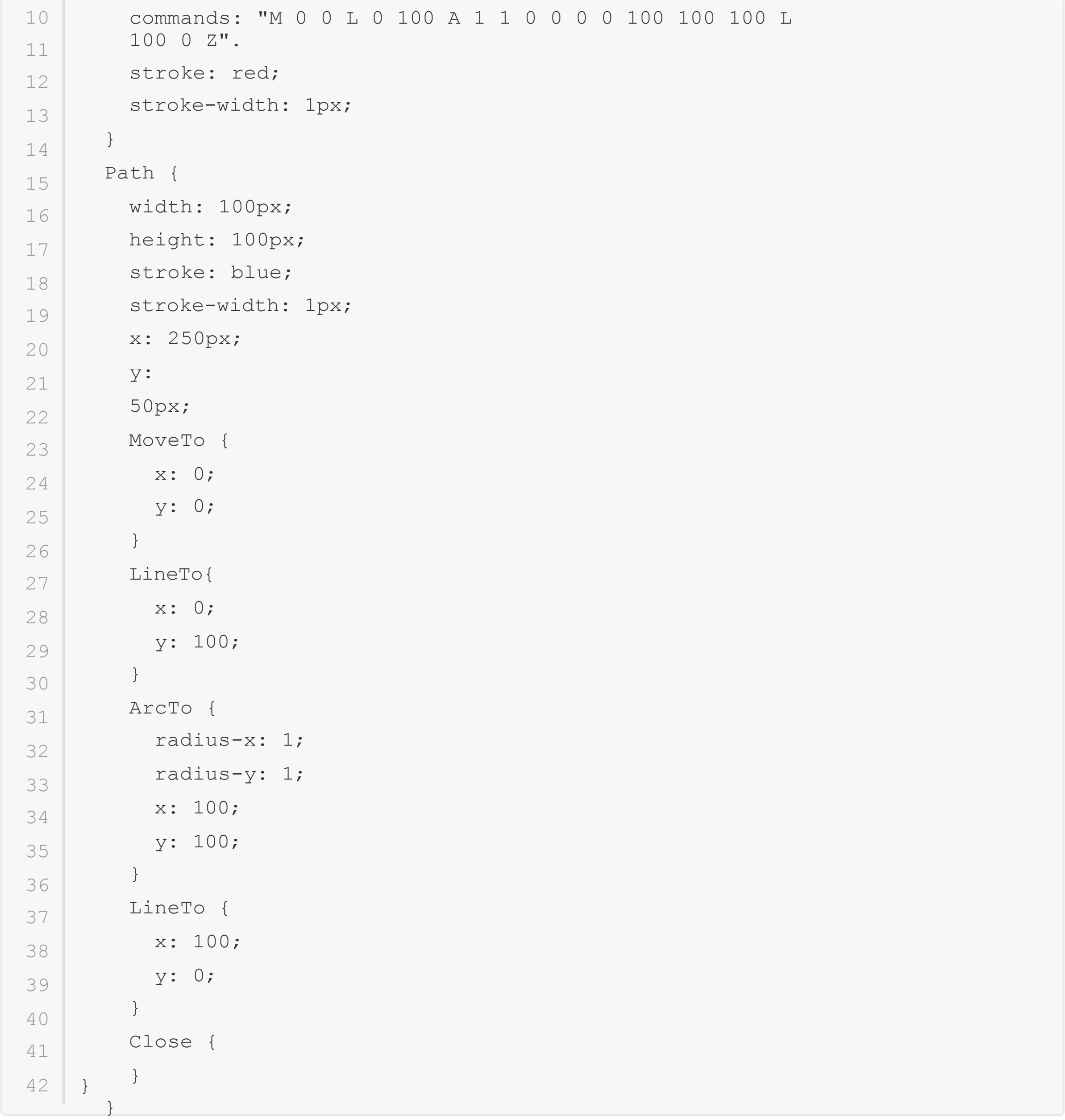
- `M(x:float,y:float)` : `MoveTo` indicates that this is a new starting point and will close the previous drawing. Example: `M 0 100`

- `L(x:float,y:float)` : `LineTo` Indicates from the previous point to the current point, this will draw a straight line segment. Example: `L 100 100`
- `A(radius_x:float,radius_y:float,large_arc:bool,sweep:bool,x_rotation:float,x:float,y:float)` : `ArcTo`
  - radius_x : radius of tangent ellipse cross length
  - radius_y : longitudinal radius of the tangent ellipse
  - 

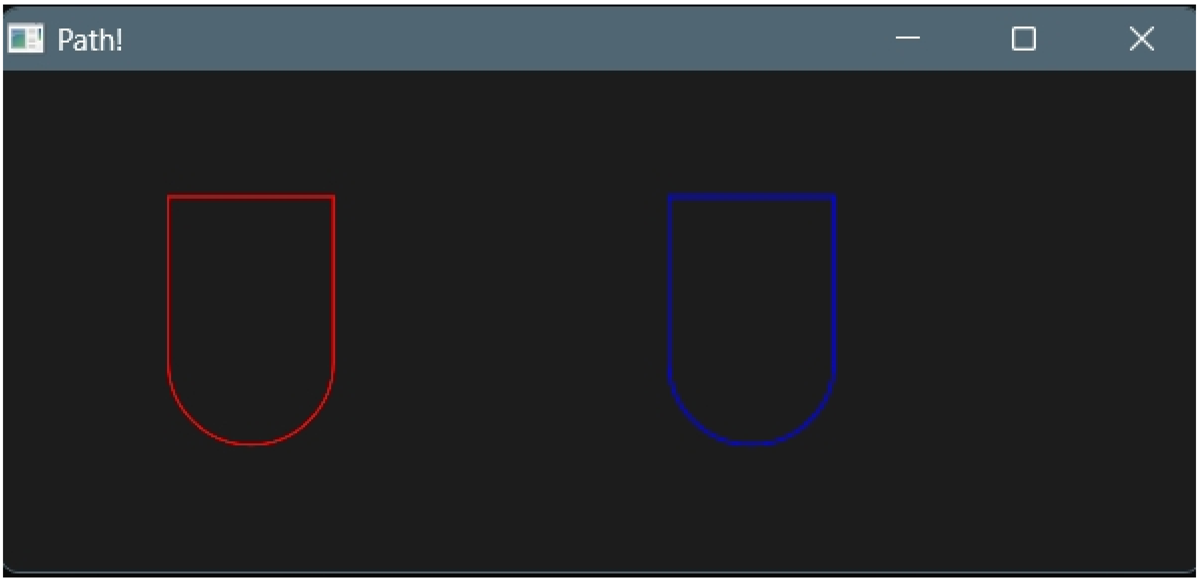
  - large_arc: of the two arcs of the closed ellipse, this flag selects the larger arc to be rendered. If the attribute is false, the shorter arc will be rendered
  - sweep: draw clockwise or counterclockwise (true is
  - clockwise) x_rotation: the number of degrees the tangent ellipse is rotated along the x-axis
- `C(control_1_x:float,control_1_y:float,control_2_x:float,control_2_y:float,x:float,y:float)` : `CubicTo` ,Smooth Bessel Curve
  - control_1_x: horizontal coordinates of control point 1, the same as the next, not all written here
- `Q(control_x:float,control_y:float,x:float,y:float)` : `QuadraticTo` Quadratic Bessel Curve
- `Z()` : `Close` closes the current subpath and connects it from the current position to the starting point.

example

```
export component MainWindow inherits Window {
  height: 200px;
  width: 480px;
  title: "Path!";
  Path {
    width: 100px;
    height: 100px;
    x: 50px;
    y: 50px;
```



```
    commands: "M 0 0 L 0 100 A 1 1 0 0 0 0 100 100 100 L
    100 0 Z".
    stroke: red;
    stroke-width: 1px;
  }
  Path {
    width: 100px;
    height: 100px;
    stroke: blue;
    stroke-width: 1px;
    x: 250px;
    y:
    50px;
    MoveTo {
      x: 0;
      y: 0;
    }
    LineTo{
      x: 0;
      y: 100;
    }
    ArcTo {
      radius-x: 1;
      radius-y: 1;
      x: 100;
      y: 100;
    }
    LineTo {
      x: 100;
      y: 0;
    }
    Close {
    }
}
  }
```



Flag

When you see this, it means that you have finished the introductory chapter, next in order that you can better understand the advanced components, please move to the basics, after learning the basics of advanced components to learn!

basics

When you see here that the ordinary components have been mastered, in order to allow you to learn advanced components and other subsequent knowledge without obstacles, please be patient to learn the basics, the basics of some of the terms after my modification is not the same as the translation of the name of the term, if you find a term you can not understand, please query the description table.

Slint file writing sequence and structure

The slint file is written in the same order as js, from top to bottom, which means that components in the lower box need to be defined in the upper box before they can be used.
(custom component), so the code below is wrong!

```
import { Button } from "std-widgets.slint";
export component MainWindow inherits Window
{
  MyButton{
    height:
    50px; width:
    50px;
  }
component MyButton inherits Button
  { text: "My Button";
}
```

Correct Code

Simply move the MyButton declaration to the front

```
import { Button } from "std-widgets.slint"
;
component MyButton inherits Button {
  text: "My Button";
}

export component MainWindow inherits Window
  { MyButton{
    height:
    50px; width:
    50px.
  }
}
```

Slint component structure

The component structure of slint is a tree structure, and each slint file can define one or more components.

Component Access and Naming

Component Access

Knowing that the structure of the component is a tree structure, it is obvious that we can access the component hierarchy through the tree. slint obviously takes this into account, so the component hierarchy is accessed in slint in the following way:

1. `root` : the root component of the tree, that is, the outermost component of the component, the root of all subcomponents.
2. `self` : the current component, through self can directly access all the properties and methods of the current self
3. `parent` : the parent of the current component

Identifiers (naming conventions)

Like most other language specifications, it consists of `(a~z)`, `(A~Z)`, `(0~9)`, `(_,-)`, and cannot start with a number or -, and for slint, `_ is the first character in the alphabet.`
`Same as -` in a non-beginning position to act as a canonical name, meaning: `my_button == my-button`

named component

Get a reference to a component by naming it with `:=!`

```
export component MainWindow inherits Window {
  height: 300px;
  width: 300px;
  text1:=Text {
    text: "Hello" + num;
   }
}
```

exegesis

- `//` : Single-line comment
- `/* ... */` : Multi-line comments

Types in Slint

> ❗Note: I made a slight modification in the type

typology	instructions	default value
int	signed integer	0
float	Signed 32-bit floating point numbers (f32)	0
bool	boolean	false
string	string (computer science)	""
Color.	RGB colours with Alpha channels, each with 8-bit precision, can also be 16 binary system of weights and measures	transparent

Color.brush	Special colours, which can be graded or changed from the base colour, are used more widely	transparent
Length.phx	Quantity used for unit conversion, length = `integer` * 1phx	0phx
typology	**instructions**	**default value**
Length.size	Commonly used units of length, divided into `px,pt,in,mm,cm` (`pt:1/72` inch , `in(`inch`):2.54cm`)	0px
Length.rem	Follow the component font size unit	0rem
duration	The unit of time, used in animation, is divided into: `ms,s`	0ms
angle	Angle unit, mostly used for rotation, gradient. The units are: `deg,rad,turn` (`1turn).` `= 360deg = 2Πrad`)	0deg
easing	The animation rate is divided into: `ease, ease_in, ease_in_out, ease_out,linear), which` is often referred to as ease in, ease out, linear.	linear
image	images, using `@image-url()`	empty image
percent	Percentage with `per cent`	0 per cent

colour

There is a difference between the normal colour Color.color and the special colour Color.brush. brush uses a brush to fill in elements or draw outlines. And
Brush has a few more ways:

- `brighter(factor: float) -> brush`

  Returns a new colour derived from this colour, but with its brightness increased by the specified factor. For example, if the factor is 0.5 (or for example 50%), the returned colour is 50% brighter. Negatives Reduces the brightness.

- `darker(factor: float) -> brush`

  Returns a new colour derived from this colour, but whose brightness has been reduced by the specified factor. For example, if the factor is .5 (or, for example, 50%), the returned colour is 50% darker. Negative factor Increases the brightness.

- `mix(other: brush, factor: float) -> brush`

  Returns a new colour which is this colour `andother` , with a proportionality factor given by a factor (which will be limited to `0.0` and `1.0`).

- `transparentize(factor: float) -> brush`

  Returns a new colour with its opacity reduced by `factor` . Transparency is achieved by multiplying the alpha channel by `(1 - factor)` .

- `with alpha(alpha: float) -> brush`

  Return alpha value set to `alpha` (between 0 and 1)

linear gradient

```
// Syntax:
@linear-gradient(gradient-angle, colour-percentage, colour-percentage,...) ;

@linear-gradient(90deg, #3f87a6 0%, #ebf8e1 50%, #f69d3c 100%);
```

radial gradient

```
// Syntax
@radial-gradient(circle, colour %, colour %, ...) ;

@radial-gradient(circle, #f00 0%, #0f0 50%, #00f 100%);
```

Flag

Before learning about custom types please move to properties for learning, which will facilitate your understanding of custom properties

Custom Types

constructor

Complex types can be declared by means of custom structures, which in general can no longer be called properties, but rather internal data! (according to the role) But in this article they are still called attributes, but please be strict in this identification.

```
struct User {
  name:string,
  age:int,
}

export component MainWindow inherits Window {
  height: 300px;
  width:
  300px; Text
  {
    property <User> user:{name: "I am
    Mat",age:16}; text: user.name;
  }
}
```

anonymous structured object

An anonymous structure is a structure that cannot be reused if it is declared directly and not externally named.

```
export component MainWindow inherits Window {
  height: 300px;
  width: 300px;
  Text {
    property <{name:string,age:int}> user:{name: "I am Mat",age:16};
    text: user.name;
   }
}
```

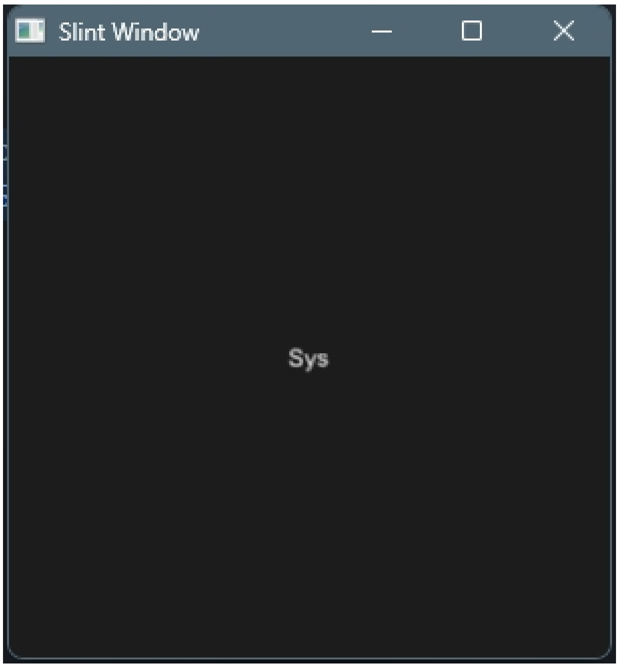
enumeration

```
enum CompoentType{
  System.
  Define
}

```

```

export component MainWindow inherits Window {
  height: 300px;
  width: 300px;
  Text {
    property <CompoentType> type : CompoentType.System ;
    text: type == CompoentType.System? "Sys": "Define";
   }
}
```



arrays

Arrays are simply declared with `[data type]` and are accessed using `[index]` .

```
export component MainWindow inherits Window {
  height: 300px;
  width: 300px;
  property <[colour]> colours:[#fff,#dc3b3b,#eee] ;
  background: colours[1];
}
```

causality

All components have properties. Properties are an important part of a component, and there are default and custom properties. Properties have four access levels corresponding to their visibility.

Attribute Visibility

- `private` : can only be accessed from within the component, it is the default
- The `in`: attribute is an input. It can be set and modified by the user of the component, e.g. by binding or by assignment in a callback. Components can provide default bindings, but they cannot assign them
- `out` : an output property that can only be set by the component and can be retrieved externally
- `in-out` : Properties of public reads and writes

Custom Properties

```
export component MainWindow inherits Window {
  in property <int> num1;
  in-out property <int> num2;
  out property <int> num3;
  // property <int> num4
  private property <int> num4.
}
```

Attribute assignment (one-way binding of attributes)

By setting a value directly after a declared property, the value is the default value of the property and represents a one-way binding to the property.

```
export component MainWindow inherits Window {
  in property <int> counter : 10;
}
```

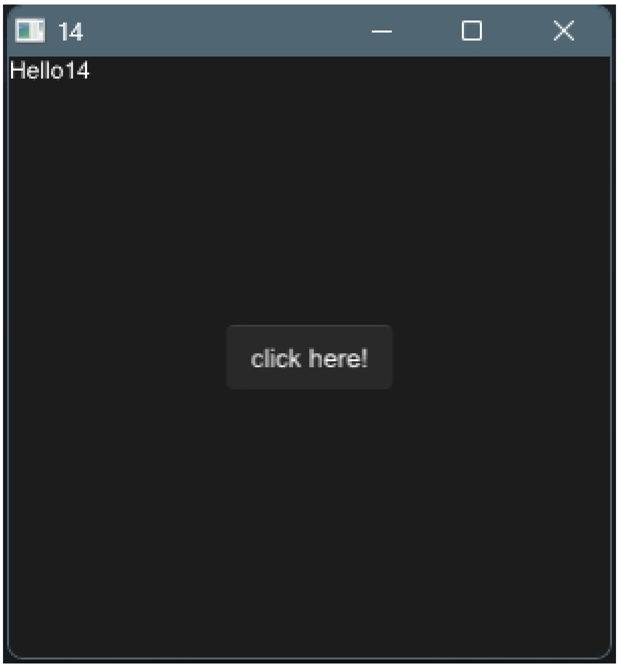
Bidirectional binding of properties

Bidirectional binding of attributes enables responsive modification of attributes by using `<=> for` bidirectional binding.

`private` property access

By combining two-way binding and component naming `private` properties can also be accessed.

```
import { Button } from "std-widgets.slint" ;
export component MainWindow inherits Window {
  height: 300px;
  width: 300px;
  property <int> root-num <=> text1.num;
  title: root-num;
  text1:=Text {
    x: 0px;
    y: 0px;
    property <int> num : 10;
    text: "Hello" + num;
   }
  Button {
    text: "click here!";
    clicked => {
      parent.root-num +=2;
     }
   }
}
```



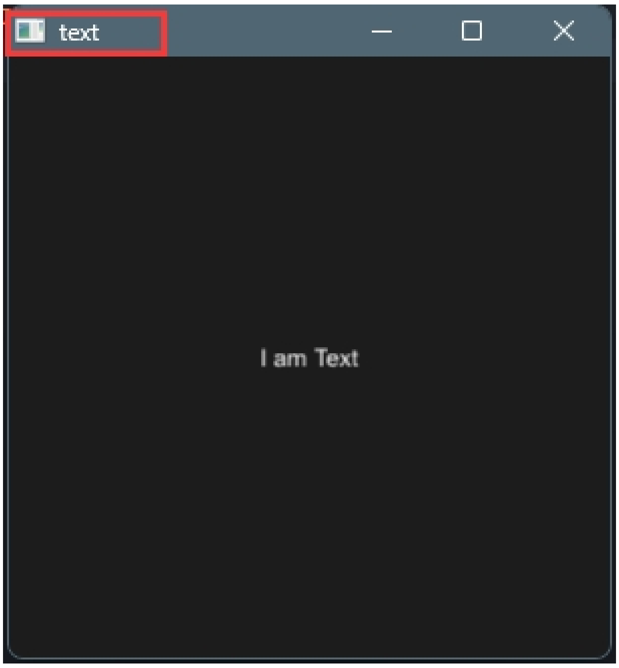
Hello14

Functions and Callbacks

function (math.)

We can define functions within a component to help the component do its job, in the case of a function it is defined within the component, in Slint if a function is called from within a component's property definition it must be pure, i.e. always return the same result given the same input parameters. Pure functions need to be declared using the `pure` keyword, and pure functions are also known as a kind of callback (or so I think).

```
 export component MainWindow inherits Window {
   height: 300px;
   width: 300px;
   title: text.get-name();
   text:=Text {
     text: "I am Text";
     property<string> name : "text";
     pure public function get_name()-> string {
       self.name.
      }
    }
}
```



I am Text

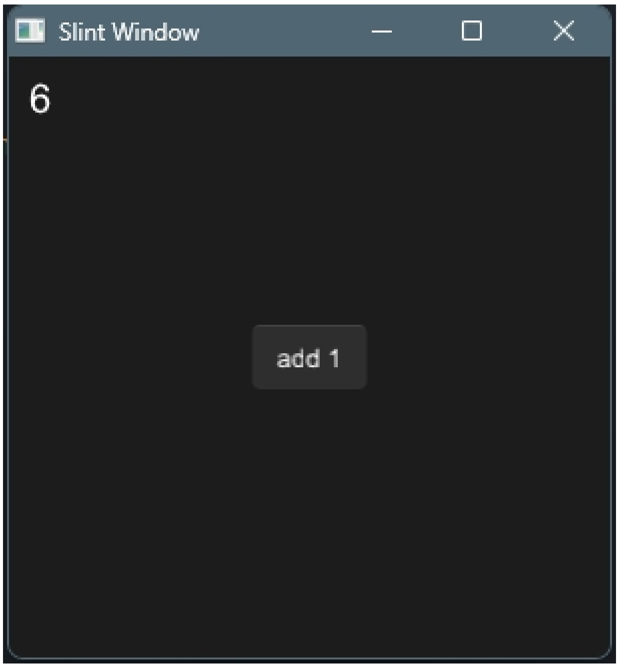
pull back (of a currency)

Components can declare callbacks that are used to pass state changes outside the component. For callbacks we usually need to define them in 2 steps:

- Declaring callbacks: declaring them using the `callback` keyword
- Writing callbacks: declaring them using arrow functions

Callbacks are special functions, so callbacks can also have an input and a return value, please control this when declaring the callbacks

```
import { Button } from "std-widgets.slint"
;
component MyBtn inherits Text{
  in-out property <int> num:0;
  callback click;
  click => {
    self.num += 1;
  }
}
  -         -                  herits Window
{
  height: 300px;
  width: 300px;
  Button {
    text: "add 1";
    clicked => {
      btn.click()
    }
   }
   btn:=MyBtn {
     x: 10px;
     y: 10px;
    font-size: 20px;
    text: self.num;
   }
```



callback alias

Callbacks can also have aliases, and you can set up callback aliases using two-way binding

```
export component Example inherits Rectangle {
    callback clicked <=> area.clicked;
    area := TouchArea {}
}
```

conditional on a loop

prerequisite

The construction of conditional statements in slint is the same as in other languages `if - else if - else`

```
if(condition){}
else if (condition){}
else{}
```

Of course conditions can also be used on constructed components to determine the component state

```
if Condition : Component
```

ternary expression

Conditional statements can be made with ternary expressions.

```
Condition? Match success return value:Failure return value
```

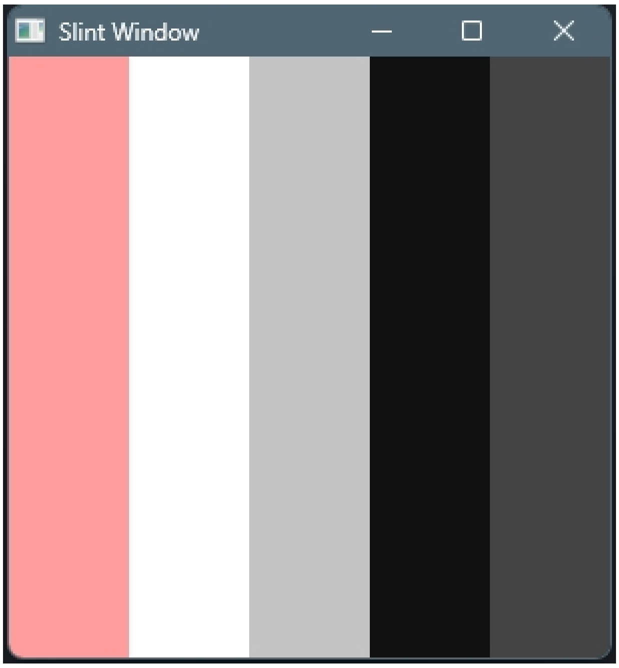
cycle

Component replication can be achieved through the use of loops, where item is an element of the loop object (order), `[index] the index` ofthe current item, I personally think that this syntax is more inappropriate, I prefer, such as: `for (item,index) in` this kind of

```
for item[index] in loop object
```

example

```
export component MainWindow inherits Window {
  height: 300px;
  width: 300px;
  property <[colour]> colours : [#ff9d9d,#fff,#c4c4c4,#111,#444] ;
  for color[index] in colors: Rectangle {
    height: root.height;
    width: root.width / colours.length;
    x:self.width * index;
    background: colour;
   }
}
```



anime

Define the animation by `animate`, the parameters to be defined in the animation are as follows:

- `delay` : the amount of time to wait before starting the animation
- `duration` : the time it takes for the animation to complete
- `iteration-count` : The number of times the animation should run. Negative values specify infinite replay of the animation.
- `easing` : animation rate, can be `linear`, `ease`, `ease-in`, `ease-out`, `ease-in-out`, `cubic-bezier(a, b, c, d)`

```
export component MainWindow inherits Window
  { width: 300px;
  height: 300px;
  background: area.pressed ? #fff : red;
  animate background {
      duration: 100ms;
  }

  area := TouchArea {

  }
}
```

Flag

When you see this, most of the basics have been mastered, so move on to the advanced components to learn until the next Flag!

Advanced knowledge

state of affairs

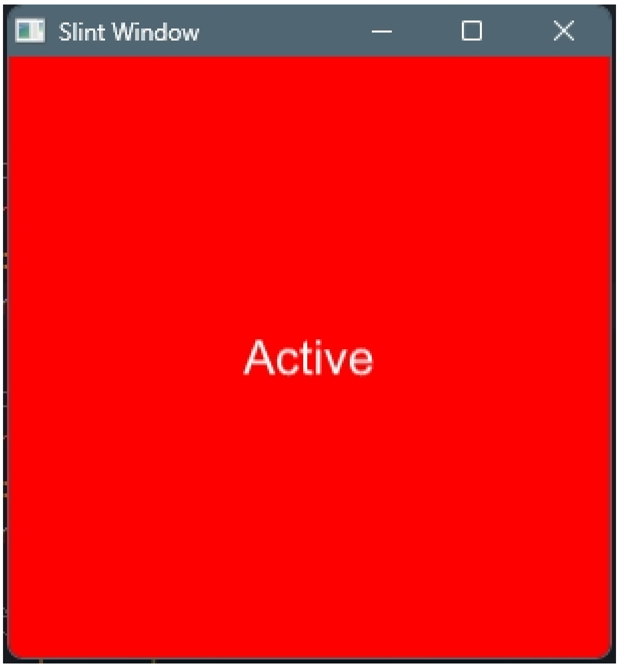
For components, multiple states can be declared, each with different judgement rules, and states need to be declared using `states[]`, with specific syntax:

```
states[
    State 1 when Condition {}
    State 2 when Condition {}
    ...
]
```

example

```
export component MainWindow inherits Window
  { width: 300px;
  height: 300px;
  default-font-size: 24px;
  property <bool> active: true;
  label := Text { }
  area := TouchArea {
      clicked => {
          active = !active;
      }
  }

  states [
    // Declare the active-click state.
      active-click when active && !area.has-hover:
          { label.text: "Active";
          root.background: blue;
      }
      // Declare the active-hover state
      active-hover when active && area.has-hover: {
          label.text: "Active Hover";
          root.background: green;
      }
      // Declare the clicked state
      clicked when !active: {
          label.text: "Clicked";
          label.colour:#000;
          root.background: #fff;
      }
  ]
}
```



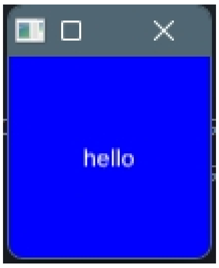
Change animation by state

Here is a modification of the official case, which gives two states, disabled and down, and changes the animation by using the out and in keywords to change the animation inwards and outwards, where * denotes a wildcard (all).

```
export component AnStates inherits Window
  { width: 100px;
  height: 100px;

  text := Text { text: "hello";
  } in-out property<bool>
  pressed;
  in-out property<bool> is-
  enabled; TouchArea {
    clicked => {
      root.is-enabled = !root.is-enabled;
      root.pressed = !root.pressed
    }
  }
  states [
      disabled when !root.is-enabled : {
          background: grey; // same as root.background:
          grey; text.colour: white;
          out {
              animate * { duration: 800ms; }
```

```
            }
        }
        down when pressed : {
            background: blue;
            in {
                animate background { duration: 300ms; }
            }
        }
    ]
}
```



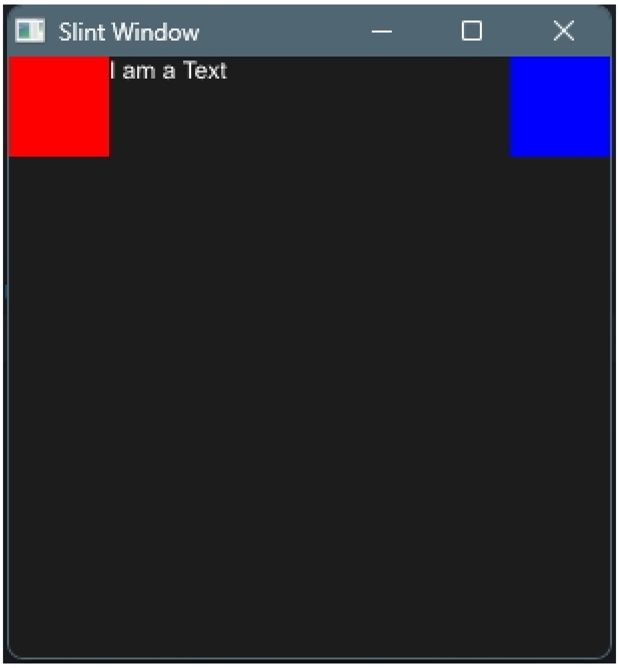
slots

Slots are used to insert the required children into a part of the component, using `@children` in the slint to specify the insertion location.

```
component MyComponent inherits HorizontalLayout {
  height: 300px;
  width: 300px;
  Rectangle {height: 50px;width: 50px;background:
  red;} @children
  Text {
    text: "I am a Text";
  }
}

export component MainWindow inherits Window
  { width: 300px;
  height: 300px;

  MyComponent {
    Rectangle {height: 50px;width: 50px;background:
    blue;}
  }
}
```



Import and export of modules

The role of import and export is to allow better reuse of components or data, so we know these keywords:

1. global: global variable
2. export: export
3. import: import
4. from: file address

global variable

To make a property or structure or enumeration globally available you need to define it using the `global` keyword, so that it can be used throughout the project

```
global MyColors {
  in-out property <color> red : #e24949;
  in-out property <color> green :
  #6de249; in-out property <color> blue :
  #4989e2.
}
export component MainWindow inherits Window
  { width: 300px;
  height: 300px;
  background: MyColors.green;

}

export {
  MyColors
}
```

derive

Export keyword `export` There are several ways to export:

1. `export{...}` : Exported content, selectable
2. `export component ...` : Export a single
3. `export * from "slint file address"` : export all

Export Rename

Export items can be renamed using the `as` keyword when exporting.

```
export {MyColors as DefaultColors};
```

import (data)

Use the `import` keyword in conjunction with `from to import a` module file.

```
  import {MyColors} from ". /colors.slint";
```

example

```
import { MyColors } from ".
/14 global.slint".


component Example inherits Window
  { height: 100px;
  width: 100px;
  background: MyColors.red;
}
```

Flag

When you see this markup move to Lifecycle

Advanced Components

TouchArea

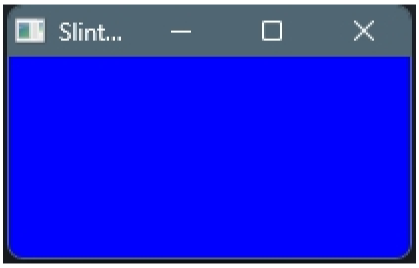
Use TouchArea to control what happens when the area it covers is touched or interacted with using the mouse. When not part of a layout, its width or height defaults to 100% of its parent element

example

```
export component Example inherits Window {
  width: 200px;
  height: 100px;
  background: area.pressed?red:blue;
  area := TouchArea {
      width: parent.width;
      height: parent.height;
      clicked => {
          root.background = #777
      }

  }
}

```



functions

- clicked(): called on click, presses the mouse and releases the element.
- moved(): the mouse has been moved. Called only when the mouse is pressed. • pointer-event(PointerEvent): called when the button is pressed or released.

`PointerEvent`

Represents an event sent by the window system to the pointer. Generate this structure and pass it to the `pointer-event`'s callback `TouchArea` element

- `button` (*PointerEventButton*): button pressed or released
- `kind` (**PointerEventKind**): button type
- `modifiers` (*KeyboardModifiers*): Keyboard modifiers pressed during the event.

`PointerEventButton`

This enumeration describes the different types of pointer events for the buttons, usually in the mouse

- `other` : A button that is not left, right or centre. For example, this is used for the fourth button on a mouse with many buttons.
- `left` : Left button.
- `right` The right button.
- `middle` : The centre button.

`PointerEventKind`

The enumeration report occurs in the `PointerEventButton`

- `cancel` : The operation is cancelled.• `down` : The button is pressed.• `up` : The button is released.

FocusScope

FocusScope exposes callbacks to intercept critical events. Note that FocusScope will only call them on has-focus.

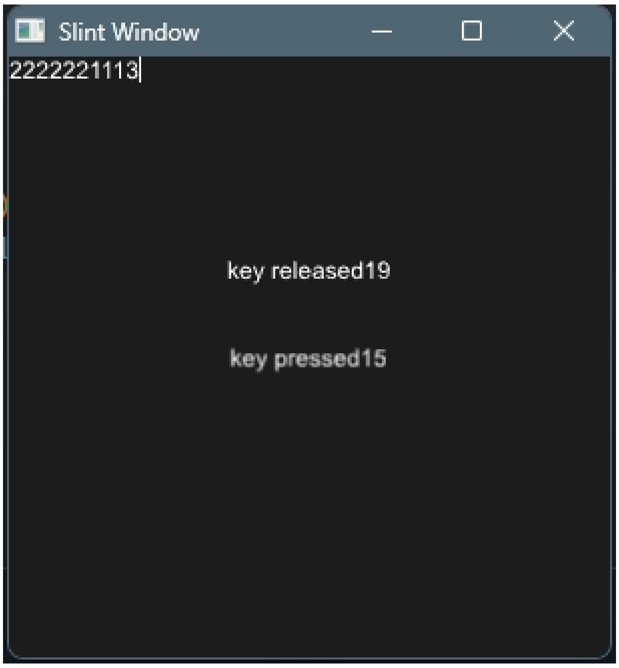
KeyEvent has a text property which is the character of the key entered. When a non-printable key is pressed, that character is either a control character or is mapped to a private Unicode character. The mapping of these special non-printable characters is available in the Key namespace

example

```
export component MainWindow inherits Window
  { width: 300px;
  height: 300px;
  text1:=Text {

  }
  text2:=Text{
    y:100px;
  }
  FocusScope {
    property <int> press:0;
    property <int> release:0;
    TextInput {}
    key-pressed(e) => {
      press+=1;
      text1.text = "key pressed" + press;
      accept
    }
    key-released(e) => {
      release+=1;
      text2.text = "key released"
      +release; accept
    }
  }

}
```



2222221113

key released19

key pressed15

functions

- key-pressed(KeyEvent) -> EventResult: called when a key is pressed, argument is a KeyEvent structure. (Called only if you enter one of the 4 keys in KeyboardModifiers)
- key-released(KeyEvent) -> EventResult: called when the key is released, argument is a KeyEvent structure. (Called on any input)

  typical example

KeyEvent

This structure is generated and passed to the FocusScope element's key press and release callbacks. Contains fields:

- text (string): the string representation of the key.
- modifiers (KeyboardModifiers): keyboard modifiers that were pressed during the event

EventResult

This enumeration describes whether the event was rejected or accepted by the event handler.

- reject: the event is rejected by this event handler and may

then be handled by the parent• accept: the event has

been accepted and will not be processed further

KeyboardModifiers

This structure is generated as part of KeyEvent to indicate which modifier keys were pressed

during the generation of the key event. Contains fields:• control (bool): true if the control

key was pressed. on macOS this corresponds to the command key.

- alt (bool): true if the alt key is pressed.
- shift (bool): true if the Shift key is pressed.
- meta (bool): true if the Windows key is pressed on Windows, or the control key is pressed on macOS.

PopupWindow

A kind of low-level popup box, it is not possible to

externally access the components in the popup box

to display the popup window via the `show`

method.

example

```
import { Button } from "std-widgets.slint";
export component MainWindow inherits Window
{
  width: 300px;
  height:
  300px.
  popup := PopupWindow {
    Text {
      text: "I am Popup";
    }
    x:
    20px;
    y: 20px.
    height:
    50px;
  Button {width:
    50px.
  }
```

```
    text: "Show
    Popup"; clicked =>
    {
      popup.show()
  } }
}
```

functions

- show: show popups

Dialog dialogue box

A type of dialogue box, which you might think is similar to a pop-up box, but dialogue boxes are qualified in that they can have any number of `StandardButton` or other button with the `dialog-button-role` property.

```
import { Button , StandardButton} from "std-widgets.slint";
export component MainWindow inherits Dialog {
  height: 720px;
  width: 1200px;
  title: "Dialog!";
  icon: @image-url("... /... /imgs/rust.png");;
  //Main Element
  Text {
    font-size: 30px;
    text: "This is a dialog";
   }
  StandardButton {
    kind: ok;
   }
  StandardButton {
    kind: cancel;
   }
  Button {
    width: 120px;
    text: "info";
    // Pretend to be a dialog-button element.
    dialog-button-role: action;
   }
}
```

life cycle

Each component has an init initialisation lifecycle, which is activated when the component is initialised (rendered).

```
init => {//init...}
```

Attribute Quick Check

popular

height

```
height: 200px;
```

Width **width**

```
width:200px;
```

Position **x** and **y**

The position of an element relative to its parent

```
x:20px;
```

Stacking level **z**

The stacking order of elements in the same level, default value is 0

```
z:1111.
```

Grid layout **col, row, colspan, rowspan**

```
Rectangle { background: green; colspan: 1; col: 2;}
```

Stretch **horizontal-stretch** and **vertical-stretch**

```
horizontal-stretch: 2;
```

Maximum size of an element **max-width** and **max-height**

```
max-width:1000px;
```

Minimum size of an element **min-width** and **min-height**

```
min-width:120px;
```

Preferred size of elements **preferred-width** and **preferred-height**

```
preferred-height:100px;
```

Whether to show **visible**

Visibility, default true

```
visible:false;
```

Transparency **opacity**

Default value is 1 (0 is fully transparent, 1 is fully opaque)

```
opacity:0.5;
```

Accelerated rendering **cache-rendering-hint**

Default false

drop-shadow-blur

The degree of blurring of the shadows, default value is 0

```
    drop-shadow-blur:2.
```

drop-shadow-color

Shadow position **drop-shadow-offset-x** and **drop-shadow-offset-y**

Horizontal and vertical distance of the shadow from the frame of the element, if negative, the shadow is located to the left and above the element

```
drop-shadow-offset-x:2px;
```

Window Params

causality	Description (type)	typical example
default-font-family	Default Text Type (String)	default-font-family: "Helvetica,Verdana,Arial,sans-serif";
default-font-size	Default text size (Size)	default-font-size: 16px;
default-font-weight	Default text thickness (Int)	default-font-weight:700
background	Background (Colour.brush)	background: @linear-gradient(90deg,#ddd 0%,#ddc5c5 50%,#ed9797 100%);
always-on-top	Always on top of other pages (Bool)	always-on-top: true;
no-frame	No border, default false (Bool)	no-frame: false;

causality	Description (type)	typical example
icon	Window Icon (Image)	icon: @image-url("... /... /imgs/rust.png");;
title	Window Title (String)	title: "Window!";

Text Params

causality	Description (type)	typical example
horizontal-alignment	Horizontal alignment (TextHorizontalAlignment)	default-font-family: "Helvetica,Verdana,Arial,sans-serif";
vertical-alignment	vertical alignment (TextVerticalAlignment)	default-font-size: 16px;
wrap	TextWrap	default-font-weight:700
overflow	TextOverflow strategy	overflow: elide;
font-size	Text size (Length.size)	font-size: 20px;
colour	Text colour (Color.colour)	colour: #fff;
font-weight	Text thickness (Int)	font-weight:700;
letter-spacing	Text interval size (Length.size)	letter-spacing:2px;
text	Text content (String)	text: "I am a Text component";

TextOverflow

This enumeration describes how text is displayed if it is too wide to fit into the Text width.

• clip: the text will be simply clipped. • elide: text will be elided as...

TextHorizontalAlignment

This enumeration describes the different types of content for which text is aligned along the horizontal axis of the Text element.

• left: the text will be aligned with the left edge of the containing box. • centre: the text will be centred horizontally in the containing box. • right: the text will be aligned on the right side of the containing box.

TextVerticalAlignment

This enumeration describes the different types of content for which text is aligned along the vertical axis of the Text element.

- top: the text will be aligned with the top of the containing box.

- centre: the text will be centred vertically in the containing box.
- bottom: the text will be aligned with the bottom of the containing box.

TextWrap

This enumeration describes how to wrap text when it is too wide to fit in the Text width.

- no-wrap: text will not wrap, but will overflow.
- word-wrap: text will wrap with word boundaries.

Input Box PropertiesTextnput Params

Contains text properties (Text Param)

causality	Description (type)	typical example
input-type	Input box type (InputType)	input-type: text;
read-only	Read-only or not (Bool)	read-only: false;
selection-background-colour	Background colour of text when typing (Colour)	selection-background-color: blue;
selection-foreground-colour	The colour of the text at the time of input (Color)	selection-foreground-color: red;
single-line	Whether it is a single line, i.e. no line breaks (Bool)	single-line: false;
text-cursor-width	Width of the cursor (Length.size)	text-cursor-width:8px;

InputType

This enumeration is used to define the type of input field. Currently, this only distinguishes between text and password input, but in the future it can be extended to define which type of virtual keyboard should be displayed, for example.

- text: the default value. This will render all characters normally
- password: this renders all the characters, which default to * .

Image Params

causality	Description (type)	typical example
colourize	Override foreground colour (Color)	colourize:Colors.aliceblue;
source	Image source (Image)	source: @image- url("... /... /imgs/rust.png");
image-fit	Image fill type (ImageFit)	image-fit:fill;
image-rendering	Image scaling method (ImageRendering)	image-rendering: smooth;

rotation-origin-x, rotation-origin-y	Setting the position of the centre of rotation (Length.size)	rotation-origin-x: 23px;
rotation-angle	Angle of rotation	rotation-angle: 30deg;
causality	**Description (type)**	**typical example**
source-clip-height, source-clip-width	Crop Height|Width (Length.size)	source-clip-height: 200;
source-clip-x, source-clip-y	Crop position (Length.size)	source-clip-x: 100;

ImageFit

This enumeration defines how the source image fits into the Image element.

- fill: scales and stretches the source image to fit the width and height of the Image element.
- contain: the source image is scaled to fit the size of the Image element while preserving the aspect ratio.
- cover: the source image is scaled to cover to the size of the Image element while preserving the aspect ratio. If the aspect ratio of the source image does not match that of the element, then the image is cropped to fit.

ImageRendering

This enumeration specifies how the source image is scaled.

- smooth: scale the image using a linear interpolation algorithm.
- pixelated: scales the image using the nearest neighbour algorithm.

Flickable Params

causality	Description (type)	typical example
interactive	Input box type (InputType)	interactive: true;
viewport-height, viewport-width	Scroll window size (Length.size)	viewport-height: 300px;
viewport-x, viewport-y	Position of child elements relative to the scroll window (Length.size)	viewport-x: 0px;

Grid Layout GridLayOut

causality	Description (type)	typical example
spacing	Element spacing (Length.size)	spacing: 10px;
padding (left,right,top,bottom)	Layout inner margins	padding: 4px;

	(Length.size)	

HorizontalLayout | VerticalLayout

causality	Description (type)	typical example
spacing	Element spacing (Length.size)	spacing: 10px;
padding (left,right,top,bottom)	Layout inner margins (Length.size)	padding: 4px;

causality	Description (type)	typical example
alignment	Element alignment (LayoutAlignment)	alignment: end

LayoutAlignment

An enumeration representing the alignment properties of HorizontalBox, VerticalBox, HorizontalLayout or VerticalLayout.

- stretch: uses the minimum size of all elements in the layout, allocating the remaining space between all elements according to the element stretch attribute.
- centre: uses the preferred size of all elements, distributing the remaining space evenly before the first element and after the last element. • start: uses the preferred size of all elements, placing the remaining space after the last element.
- end: use the preferred size for all elements, placing the remaining space before the first element.
- space-between: use the preferred size for all elements, distributing the remaining space evenly between elements.
- space-around: Use the preferred size of all elements, distributing the remaining space evenly before the first element, after the last element, and between elements.

TouchArea

causality	Description (type)	typical example
has-hover	Mouse contact event (out Bool)	
mouse-cursor	MouseCursor events	
mouse-x, mouse-y	Mouse position in TouchArea	
pressed-x, pressed-y	The position of the TouchArea when the mouse was last pressed.	
pressed	Mouse long press event (out bool)	

MouseCursor

This enumeration represents the different types of mouse cursors. It is a subset of the mouse cursors available in CSS. For more information and pictograms, see the MDN documentation for cursors. Depending on the backend and the operating system used, one-way resize cursors may be replaced by two-way cursors.

- default: the system default cursor.
- none: no cursor is shown.
- help: Cursor for help information.
- pointer: pointer to the link.
- PROGRESS: The programme is busy, but it is still possible to interact with it.
- WAIT: The programme is busy.
- crosshair.
- text: indicates the cursor for selectable text.
- alias: an alias or shortcut is being created.
- copy: a copy is being created.
- MOVE: Something needs to be moved.
- no-drop: Some things can't be dropped here.

- not-allowed: not allowed to take action
- grab: something is grabbable.
- GRABBING: Something is caught.
- col-resize: indicates that a column can be resized horizontally.
- row-resize: means a row can be resized vertically.
- n-resize: one-way resize to the north.
- e-resize: resize east in one direction.
- s-resize: resize south in one direction.
- w-resize: resize one-way to west.
- ne-resize: resize north-east in one direction.
- nw-resize: one-way resize north-west.
- se-resize: southeast resize.
- sw-resize: resize southwest in one direction.
- ew-resize: resize in both east and west directions.
- ns-resize: resize in both directions.
- nesw-resize: resizes NE-SW in both directions.
- nwse-resize: bi-directional northwest-southeast resize.

Dialog

causality	Description (type)	typical example
icon	Window Icon (Image)	
title	Window Title (String)	

accessibility

I think it's a characteristic, not an attribute.

- accessible-role: element role (defaults to none for most elements, but text for text elements)
- accessible-checkable: whether the element can be checked or not
- accessible-checked: whether the element is checked or not - corresponds to the "checked" state of checkboxes, radio buttons and other widgets.
- accessible-description: description of the current element
- accessible-has-focus: set to true when the current element currently has focus.
- accessible-label: the label of the interactive element (defaults to null for most elements, or the value of the text attribute of a text element)
- accessible-value-maximum: maximum value

- accessible-value-minimum: minimum value
- accessible-value-step: the smallest increment by which the current value can be changed
- accessible-value: the current value.

Flag

When you see this mark it means that you are 85% of the way through slint, for the next 15% check out the System Custom Components.md document, which has a release date of 20230904