

# Introduction

**ZikoUIElement** is the constructor class of every UI element in zikojs.

It encapsulates the properties and behaviors of a UI element, including its HTML representation, styling, attributes, and event handling...

You can create a new UI element using the following syntax :















```
const UI = new ZikoUIElement(element);  
// element can be an HTML tag string or a DOM element.
```



Alternatively, you can simply use the [Built-in UI Elements](#) provided by zikojs UI module."

In general, ZikoUIElement supports:






- Nesting UI elements within each other.
- Applying styling to UI elements using object notation.
- Event handling
- Manipulating the DOM by adding, removing, and modifying child elements.

























































## Methodes & Getters



















- **Content :**
  -  `.html` : Get the HTML content of the UI element
  -  `.text` : Get the Text content of the UI element
- **DOM :**
  -  `.clone()` : Clone the ZikoUIElement
  -  `.render()` : Render the UI element to the DOM
  -  `.renderAfter(delay)` : Render the UI element after a specified delay
  -  `.unrender()` : Remove the UI element from the DOM.
  -  `.unrenderAfter(delay)` : Remove the UI element from the DOM after a specified delay.
  -  `.setTarget(target)` :
- **Attributes :**
  -  `.setAttr(name,values)` : Set attribute(s) on the UI element.
  -  `.removeAttr(...names)` : Remove attribute(s) from the UI element.
  -  `.setId(value)` : Set the ID attribute of the UI element.
  -  `.setClasses(...classes)` : Set CSS classes on the UI element.
  -  `.addClasses(...classes)` : Add CSS classes to the UI element.
  -  `.attr` : Get the attributes of the UI element.

-  `.id` : Get the Id of the UI element.
-  `.classes` : Get the classes of the UI element.

## • CSSOM :

-  `.style(styleObject)` : Set the Style of the UI element.
-  `.size(width,height)` : Set the size of the UI element by specifying its width and height.
-  `.st.show()` : Make the UI element visible.
-  `.st.hide()` : Hide the UI element.
-  `.st` : Returns an instance of ZikoStyle, a class that provides a comprehensive set of methods for manipulating and querying the styles of the UI element.

-   `.st.styles`
-   `.st.add`
-   `.st.delete`
-   `.st.color()`
-   `.st.background()` : Set the background of the UI element, including background images and gradients.
-   `.st.backgroundColor()` : Set the background color of the UI element.
-   `.st.display()`
-   `.st.margin()` : Set the margin around the UI element.
-   `.st.marginTop()` : Set the top margin of the UI element.
-   `.st.marginBottom()` : Set the bottom margin of the UI element.
-   `.st.marginRight()` : Set the right margin of the UI element.
-   `.st.marginLeft()` : Set the left margin of the UI element.
-   `.st.padding()` : Set the padding inside the UI element.
-   `.st.paddingTop()` : Set the top padding of the UI element.
-   `.st.paddingBottom()` : Set the bottom padding of the UI element.
-   `.st.paddingRight()` : Set the right padding of the UI element.
-   `.st.paddingLeft()` : Set the left padding of the UI element.
-   `.st.width()` : Set the width of the UI element.
-   `.st.height()` : Set the height of the UI element.
-   `.st.border()` : Set the border of the UI element.
-   `.st.borderTop()` : Set the top border of the UI element.
-   `.st.borderBottom()` : Set the bottom of the UI element.
-   `.st.borderRight()` : Set the right border of the UI element.
-   `.st.borderLeft()` : Set the left border of the UI element.
-   `.st.cursor()` : Set the cursor style when hovering over the UI element.
-   `.st.font()` : Set the font shorthand property of the UI element.
-   `.st.fontSize()` : Set the font size of the UI element.
-   `.st.fontFamily()`

-  `.st.translate(dx : number, dy : number, dz? : number, transitionTiming? : number)`
- `:`
-  `.st.translateX(dx : number, transitionTiming? : number)`
-  `.st.translateY(dy : number, transitionTiming? : number)`
-  `.st.translateZ(dz : number, transitionTiming? : number)`
-  `.st.rotateX(rx : number, transitionTiming? : number)`
-  `.st.rotateY(dy : number, transitionTiming? : number)`
-  `.st.rotateZ(dz : number, transitionTiming? : number)`
-  `.st.scale(sx : number, sy : number, transitionTiming? : number)`
-  `.st.scaleX(sx : number, transitionTiming? : number)`
-  `.st.scaleY(sy : number, transitionTiming? : number)`
-  `.st.perspective(distance : number, transitionTiming? : number)`
-  `.st.flipeX(transitionTiming? : number)`
-  `.st.flipeY(transitionTiming? : number)`
-  `.st.flipeZ(transitionTiming? : number)`
-  `.st.fadeIn(transitionTiming? : number)`
-  `.st.fadeOut(transitionTiming? : number)`
-  `.st.isBlock()`
-  `.st.isInline()`

## Usage :

- Methode 1 :

```
let txt = text("Hello World !");
txt.style({
  color : "darkblue",
  background : "gold"
})
```

- Methode 2 :







```
let txt = text("Hello World !");
txt.st.color("darkblue")
txt.st.background("darkblue")
```

- Methode 3 : Dynamique Update

```
let txt = text("Hello World !");
txt.add("default",{color : "darkblue", background : "gold"})
txt.add("reversed",{color : "gold", background : "darkblue"})
```

[!TIP] If you want to apply dynamic styles to multiple elements, you can use the `useStyle` hooks. This approach helps maintain consistency and manage styles efficiently across various elements.

- **Events :**

-  `.evt` : Retrieve the events registered on the UI element. Each event category (`.evt.click`, `.evt.ptr`, etc.) returns a specialized class that can handle additional functionalities, including methods and getters specific to that event type.
- **Custom Events :**
  -  `.emit(event,detail)` : Emit a custom event from the UI element, optionally providing additional details in the detail parameter. This allows for flexible communication between elements.
  -  `.on(event,...callbacks : function[])` : Register and listen to custom events. The specified callbacks will be executed whenever the event is emitted.
- **useClickEvent :**
  -  `.evt.click` : Returns an instance of `ZikoEventClick`, a class that handles click-related events and provides additional methods and getters.
  -  `.onClick(...callbacks : function[])` : Register and listen to single-click events on the element.
  -  `.onDbClick(...callbacks : function[])` : Register and listen to double-click events on the element.







**Usage :**


```
txt = text("Hello World").onClick(  
    e=>console.log(e.target.value)  
)
```

**Description :**

In this example, a text element with the content "Hello World" is created using the `text()` function. The `onClick` method is then used to register a click event listener on the element. When the element is clicked, the registered callback function is executed, logging the value of the event target (i.e., the text element) to the console. This demonstrates how `onClick` can be used to handle click events and interact with the element's properties.

- **usePointerEvent Events :**

-  `.evt.ptr` : Returns an instance of `ZikoPointerEvent`, a class that handles pointer-related events and provides additional methods and getters.
-  `.onPtrDown(...callbacks : function[])` : Register one or more callbacks to handle the pointer-down event when a pointer device makes contact with the element.
-  `.onPtrMove(...callbacks : function[])` : Register one or more callbacks to handle the pointer-move event, which occurs when the pointer moves within the element.
-  `.onPtrUp(...callbacks : function[])`
-  `.onPtrEnter(...callbacks : function[])`
-  `.onPtrLeave(...callbacks : function[])`









-  .onPtrOut(...callbacks : function[])






### Usage :

```
Scene = Canvas("500px", "500px")
.onPtrDown((e) => {
    e.target.ctx.beginPath();
    e.target.ctx.moveTo(
        map(e.dx, 0, e.target.width, e.target.Xmin, e.target.Xmax),
        map(e.dy, 0, e.target.height, e.target.Ymin, e.target.Ymax),
    );
})
.onPtrMove((e) => {
    if (e.isDown) {
        const x = map(e.mx, 0, e.target.width, e.target.Xmin, e.target.Xmax);
        const y = map(e.my, 0, e.target.height, e.target.Ymin, e.target.Ymax);
        e.target.append(
            canvasCircle(x, y, 1).color({ fill: "#5555AA" }).fill(),
        );
    }
})
.onPtrUp(() => {});
```




### Description :

This example demonstrates the creation of a simple paint application using a Canvas element. The canvas is 500x500 pixels, and the user can draw on it by interacting with pointer events.






- **onPtrDown:** Starts a new drawing path on the canvas, mapping the pointer's position to a custom coordinate system.
- **onPtrMove:** Draws a continuous line by placing circles along the pointer's path while it is pressed down.
- **onPtrUp:** This event is triggered when the pointer is released. In this context, it is used to change the value of isDown, ensuring that the drawing stops when the pointer is lifted. This prevents the drawing action from continuing when the user is no longer pressing down on the canvas.
- **useMouseEvent Events :**
  -  .evt.mouse : Returns an instance of `ZikoMouseEvent`, a class that handles mouse-related events and provides additional methods and getters.
  -  .onMouseDown(...callbacks : function[])
  -  .onMouseMove(...callbacks : function[])
  -  .onMouseUp(...callbacks : function[])
  -  .onMouseEnter(...callbacks : function[])
  -  .onMouseLeave(...callbacks : function[])
  -  .onMouseOut(...callbacks : function[])
  -  .onWheel(...callbacks : function[])
- **Keyboard Events :**

-  `.evt.key` : Returns an instance of `ZikoKeyEvent` , a class that handles key-related events and provides additional methods and getters.
-  `.onKeyDown(...callbacks : function[])`
-  `.onKeyPress(...callbacks : function[])`
-  `.onKeyUp(...callbacks : function[])`
-  `.onKeysDown(...callbacks : function[])`






#### ◦ **Focus Events :**

-  `.evt.focus` : Returns an instance of `ZikoFocusEvent` , a class that handles focus-related events and provides additional methods and getters.
-  `.onFocus(...callbacks : function[])` : Register and listen to the focus event, which occurs when the element gains focus.
-  `.onBlur(...callbacks : function[])` : Register and listen to the blur event, which occurs when the element loses focus.


#### ◦ **Drag Events :**





-  `.evt.drag` : Returns an instance of `ZikoDragrEvent` , a class that handles drag-related events and provides additional methods and getters.
-  `.onDragStart(...callbacks : function[])` : Register and listen to the drag-start event, which occurs when the user starts dragging an element.
-  `.onDrag(...callbacks : function[])` : Register and listen to the drag event, which occurs as the element is being dragged.
-  `.onDragEnd(...callbacks : function[])` : Register and listen to the drag-end event, which occurs when the drag operation is finished.
-  `.onDrop(...callbacks : function[])` : Register and listen to the drop event, which occurs when the dragged element is dropped.

#### **-Clipboard Events :**


-  `.evt.clipboard` : Returns an instance of `ZikoClipboardEvent` , a class that handles clipboard-related events and provides additional methods and getters.
-  `.onSelect(...callbacks : function[])` : Register and listen to the select event, which occurs when the user selects text.
-  `.onCopy(...callbacks : function[])` : Register and listen to the copy event, which occurs when the user copies content to the clipboard.
-  `.onCut(...callbacks : function[])` : Register and listen to the cut event, which occurs when the user cuts content to the clipboard.
-  `.onPaste(...callbacks : function[])` : Register and listen to the paste event, which occurs when the user pastes content from the clipboard.

#### • **Watchers :**

-  `.observer` : Returns an instance of [ZikoObserver](#), a class that handles observing various aspects of the UI element and provides additional methods and utilities.



-  `.watchSize(callback : function)` : Observe changes in the size of the UI element and trigger the callback whenever a size change occurs.
-  `.watchIntersection(callback)` : Observe changes in the intersection of the UI element with other elements or the viewport, triggering the callback on intersection updates.
-  `.watchAttr(callback)` : Observe changes in the attributes of the UI element and execute the callback when any attribute changes.
-  `.watchChildren(callback)` : Observe changes in the child elements (e.g., addition, removal) of the UI element and invoke the callback when such changes occur.

- **Misc :**











-  `.toPdf()` :

**ZikoUIContainerElement** is a specialized subclass of ***ZikoUIElement*** designed to manage and contain other UI elements. It inherits all methods from ***ZikoUIElement*** and introduces additional methods and getters for handling child elements:

- **Acces :**

-  `.at(index)` : Retrieves the UI element at the specified index within the ***ZikoUIElement***
-  `[index]` : Alternative syntax for accessing UI elements by index

- **Dom :**

-  `.append(...items)` : Append child elements to the UI element
-  `.remove(...items)` : Remove child elements from the UI element
-  `.insertAt(index,...items)` : Insert child elements at a specified index within the UI element.
-  `.forEach(callback)` : Iterate over child elements and applies a callback function.
-  `.map(callback)` : Map over child elements and applies a callback function.
-  `.find(condition)` : Find All child elements that match a specified condition.
-  `.filter(condition,if_callback,else_callback)` : Filter child elements based on a condition, with optional callbacks for filtered and non-filtered elements.
-  `.filterByTextContent(text,exactMatch)` : Filter child elements based on text content, with an option for exact matching.
-  `.filterByClass(value)` : Filter child elements based on class name.
-  `.sortByTextContent(value,displays)` : Sort child elements by their text content.

