XML-Gui Documentation

Content

Tags and Attributes

- root
- grid
- button
- icon
- label

`root`

In `root` are no attributes allowed, but the `root` is the hard required root tag.

`grid`

There two different kinds of grids.

- Top-Level `grid`:

Attribute	Value(s)
`id`	String
`x_dim`*	u32
`y_dim`*	u32
`x_offset`*	i32
`y_offset`*	i32
`width`*	u32
`height`*	u32
`vert_align`*	`top`, `middle`, `bottom`
`hori_align`*	`left`, `middle`, `right`
`padding`	u32
`margin`	u32
`background_image`	String
`background_color`	color
`menu_button`	String
`menu_button_selected`	String
`click_sound`	String
`hover_sound`	String

- Child `grid`:

Attribute	Value(s)
`id`	String
`x_slot`*	u32
`y_slot`*	u32
`x_dim`*	u32
`y_dim`*	u32
`padding`	u32
`margin`	u32
`background_image`	String
`background_color`	color
`menu_button`	String
`menu_button_selected`	String
`click_sound`	String
`hover_sound`	String

`button`

There are two different kinds of buttons.

- `button` with text inside

Attribute	Value(s)
`id`	String
`x_slot`*	u32
`y_slot`*	u32
`texture`	String
`select_texture`	String
`text_color`	color
`select`	`true`, `false`
`on_click`	String
`click_sound`	String
`hover_sound`	String

The content of this `button` is the text for it.

- `button` with icon inside

Attribute	Value(s)
`id`	String
`x_slot`*	u32
`y_slot`*	u32
`select_texture`	String
`select`	`true`, `false`
`on_click`	String
`click_sound`	String
`hover_sound`	String
`icon`*	String
`icon_x`*	i32
`icon_y`*	i32
`icon_width`*	u32
`icon_height`*	u32
`icon_vert_align`	`top`, `middle`, `bottom`
`icon_hori_align`	`left`, `middle`, `right`

`icon`

- just some simple icon parameters

Attribute	Value(s)
`id`	String
`x_slot`*	u32
`y_slot`*	u32
`icon`*	String
`icon_mode`	`squared`, `stretched`

`label`

- just some simple label parameters

Attribute	Value(s)
`id`	String
`x_slot`*	u32
`y_slot`*	u32
`text_color`	color
`background_image`	String
`background_color`	color

The content of this `label` is the text for it.

Types

`color`

- supported color names:
  - `black`
  - `blue`
  - `green`
  - `orange`
  - `white`
  - `red`
  - `yellow`
- CSS-Type color description:
  - e.g. #000000 black

Usage Inside Rust

The following chapter will explain the functions availiable for the `GuiBuilder`.

Loading the xml-File

Creaating the `GuiBuilder` object and therefore loading the xml-file.

(1) Definition:

```
pub fn new(
    path:                       &str,
) -> VerboseResult<Rc<GuiBuilder>>;
```

(2) Parameter:

- `path` - path to the xml-file

(3) Return Type

- Result - Either a valid reference counted GuiBuilder object or a error message

Setting Callbacks

(1) Definition

```
pub fn set_callbacks(
    &self,
    functions:              Vec<(&str, Box<dyn Fn() -> ()>)>
);
```

(2) Parameter

- `functions` - Is a tuple, where the first part is the string match for the `on_click` attribute from the xml file. The second part is the closure which is executed on click.

Accessing Elements

(1) Definition

```
pub fn element_by_id(
    &self,
    id:                     &str
) -> Option<&UiElement>;
```

(2) Parameter

- `id` - Is the string match for the `id` attribute of a gui element from the xml file.

(3) Return Type

- Option - If an element with the given `id` could be found, you get a refence to an `UiElement` type. Which is defined as following:

```
pub enum UiElement {
    Button(Weak<Button>),
    Grid(Weak<Grid>),
    Label(Weak<Label>),
    Textfield(Weak<Textfield>),
    Icon(Weak<Icon>),
}
```

Example

XML File

gui.xml

```
<root>
    <grid x_dim="5" y_dim="3" x_offset="50" y_offset="50"
          width="500" height="300" padding="30">
        <button x_slot="0" y_slot="0" on_click="hello_world()">
            first button
        </button>
        <button x_slot="1" y_slot="1">second button</button>
    </grid>
</root>
```

Rust side

```
use ui::prelude::*;

...

let context = ... // create context
let gui_handler = GuiHandler::new(/* create info */, &context)?;

// creating a GuiBuilder object
let gui = GuiBuilder::new(gui_handler.clone(), "gui.xml")?;

// create a closure
let hello = Box::new(move || {
    println!("hello world");
});

// connect this closure with the on_click event
gui.set_callbacks(vec![
    ("hello_world", hello),
]);
```

Appendix

*

- Attributes which have a * are mandatory!