📘 DOCUMENTATION: BUTTON MANAGER MODULE

Author: Nethra

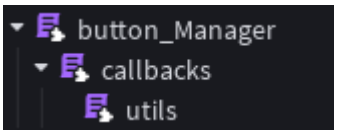
GitHub: n3thr4-lang

Purpose: A centralized system for managing Roblox UI Button states, lifecycle, and complex interactions.

---

1. Initialization

script setup(copy and paste script and place them like the image below):



To begin managing buttons, you must first initialize a new Button List instance.

```lua
local ButtonManager = require(path.to.ButtonManager)
local myManager = ButtonManager.new_button_list()
```

Adding a Button

Registers a button into the manager's memory.

```lua
myManager:add_button(GuiButton, "KeyName")
```

- **GuiButton**: The TextButton or ImageButton instance.
- **KeyName** (Optional): A unique string identifier. Defaults to button.Name.

---

2. The Core Method: Activate_button

This is the primary function used to define how a button behaves.

Lua
myManager:Activate_button(key, operator, config)

📥 Parameters:

Parameter	Type	Description

key	string / Instance	The unique key or the Button Instance itself.
operator	table	A dictionary containing callback functions (see Section 3).
config	table	A dictionary for behavior settings (see Section 4).

---

3. Button Types & Operators

The button_type defined in the config determines which callbacks the operator table requires.

A. single_press

Standard click detection.

- **Operator Hook:** fire(button, state_snapshot)

B. hold

Detects when a user starts and stops pressing.

- **Operator Hooks:** * hold(isHolding, button, x, y, state)
    - release(isHolding, button, x, y, state)

C. toggle

Maintains an internal boolean state (On/Off).

- **Operator Hooks:**
    - toggle(true, button, state)
    - untoggle(false, button, state)

D. long_press

Triggers only after holding for a specific duration.

- **Operator Hooks:**
    - start_pressing(button, x, y, state)
    - finished(button, x, y, state) — *Triggered after duration.*
    - stopped_pressing(button, x, y, state)
    - cancelling(button, x, y, state) — *If released before duration ends.*

---

4. Configuration Table (config)

Settings used to customize the button's logic:

Key	Type	Description
button_type	string	**Required**. (single_press, hold, toggle, long_press)
visible	boolean	Sets Visible property after activation.
reset_state	boolean	If true, clears timestamps and toggle status.
state	boolean	(Toggle only) Sets the initial toggle state.
time_take	number	(Long Press only) Seconds required to trigger (Default: 0.5s).

---

5. Interaction Hooks

You can add these hooks to **any** operator table to handle secondary mouse/touch events:

- enter: Triggered on MouseEnter.
- leave: Triggered on MouseLeave.
- down: Triggered on MouseButton1Down.
- up: Triggered on MouseButton1Up.
- on_toggle: Runs after binded :Activate_button.

---

6. The State Snapshot

Callbacks receive a state table. This is a read-only "snapshot" of the button's current data:

- .enabled: Boolean (Matches Interactable).
- .toggled: Boolean (Current toggle state).
- .holding: Boolean (True if currently pressed).
- .last_activation: Timestamp of the last successful click/long press.
- .last_hold / .last_release: Timestamps of raw mouse input.

---

7. API Reference

Method	Description

disable_button(key)	Disconnects all events and cancels active threads for a specific button.
toggle_visibility(bool, key)	Changes visibility for one or all buttons.
toggle_interactable(key, bool)	Enables/Disables button interaction.
reset_state(key)	Wipes the state data (toggle, timers) for a button.
remove_button(key)	Completely removes the button from the manager.
TERMINATE_EVERYTHING()	Cleans up the entire manager and destroys all connections.

💡 Code Example

```lua
myManager:Activate_button("SettingsBtn", {
    toggle = function(isTrue, button, state)
        print("Settings Open")
    end,
    untoggle = function(isFalse, button, state)
        print("Settings Closed")
    end,
    enter = function(button)
        button.BackgroundColor3 = Color3.new(1, 1, 1)
    end
}, {
    button_type = "toggle",
    state = false
})
```