CC-Scripts

Tim Anhalt (BitTim)

Aug 05, 2023

GENERAL

CHAPTER

ONE

OVERVIEW

Danger: Work in Progress

CHAPTER

TWO

BANK

Danger: Work in progress

```
wget https://raw.githubusercontent.com/BitTim/CC-Scripts/master/src/apps/Bank/install.lua
install
```

CHAPTER

THREE

BUTTON

Danger: Work in progress

CHAPTER

FOUR

DNS

4.1 Utility

CHAPTER

FIVE

ESSENTIA

Note: This app is created for use with Thaumcraft 6

5.1 Overview

Essentia is a set of scripts to control the flow of essentia from an essentia storage room, similar to this one, through one pipe to a consuming machine, like the Thaumatorium.

The app is broken down into 3 scripts, which run on different computers:

- *Controller*
- *Server*
- *Display*

Warning: Essentia can only be moved in multiples of 5, due to a limitation in the redstone contraption for consistency reasons

5.2 Dependencies

5.2.1 Mods

- Thaumcraft 6
- Project Red - Core
- Project Red - Integration
- Project Red - Transmission
- NBT Peripheral

5.2.2 Structures

ToDo: Insert images here

Important: You also need to have your *DNS infrastructure* set up and running.

5.3 Setup

5.3.1 Prequisites

- Each jar in your storage room has to be labeled.
- Each jar in your storage room needs to be part of a flow control structure.
- For every **15 jars**, you need a **controller**
- For each **controller**, you need a computer with a **wireless modem**.
- For the **server**, you need a computer with an **ender modem**.
- For the **display**, you will need a computer with a **wireless modem** and a **5 x 2** monitor.

Important: It is required to use an ender modem for the server to bypass the distance limit for transmissions

Note: It is recommended to attach a disk drive and a monitor of any size to the server

5.3.2 Automated

Danger: The automated installer is not yet implemented. This section is prewritten for the future.

You need to run the automated installer on every computer placed in the previous step and select the correct install type. To run and execute the autmated installer, run these commands. The installer will accommodate configuring the computer accordingly:

```
wget https://raw.githubusercontent.com/BitTim/CC-Scripts/master/src/apps/Essentia/
↪install.lua
install
```

5.3.3 Manual

ToDo: Insert installation instructions

Note: The automated install is recommended.

5.4 Definitions

5.4.1 Controller

The program, that manages controlling the essentia valves and reading amount of aspects stored in jars

Contents:

Dependencies

- *ComLib*
- *LogLib*

Configurable Properties

Name	Type	Default
servedAspects	`table`	`{}`
nbtPeripheralTags	`table`	`{}`
outputSide	`string`	`"back"`
modemSide	`string`	`"top"`

servedAspects

A list containing the names of all aspects this controller serves. The order of aspects determines the local ID for each aspect.

```
local servedAspects = {}
```

- **Type:** `table`
- **Default:** `{}`

Warning: Please make sure, that the order of the aspects matches the order of the color channels used in the bundled cable. To see the order of the colors, consult the table at the bottom of this page.

---

nbtPeripheralTags

A list containing the peripheral names (e.g. `nbt_observer_0`) for the NBT observers. The order of tags corresponds to the local ID.

```
local nbtPeripheralTags = {}
```

- **Type:** `table`
- **Default:** `{}`

Warning: Please make sure, that the order of the peripheral names and the order of aspects in *servedAspects* match.

---

outputSide

The side the bundled cable is connected to the computer.

```
local outputSide = "back"
```

- **Type:** `string`
- **Default:** `"back"`

---

modemSide

The side the wireless modem is connected to the computer.

```
local modemSide = "top"
```

- **Type:** `string`
- **Default:** `"top"`

Requests

- *FLOW*
- *PROBE*

FLOW

Release 5 essentia from the specified aspect. Fails if aspect is not serverd by controller or amount of essentia of specified aspect is less than 5.

```
{head = "FLOW", contents = {aspect = ""}}
```

request Contents:

Name	Type	Default	Description
aspect	`string`	`nil`	Aspect of which 5 essentia should be released.

Response contents: `nil`

PROBE

Probe the amount of specified aspect in jar. Fails if aspect is not serverd by controller.

```
{head = "PROBE", contents = {aspect = ""}}
```

Request Contents:

Name	Type	Default	Description
aspect	`string`	`nil`	Aspect of which the amount should be probed.

Response contents:

Name	Type	Description
aspect	`string`	Aspect in probed jar.
amount	`number`	Amount of stored essentia in probed jar.

> **Warning:** If `aspect` in response contents doesn't match with `aspect` in request contents, then the order of *nbtPeripheralTags* is most likely faulty.

Constants

Name	Type	Value
title	`string`	`"Essentia Controller"`
version	`string`	`"v1.0"`

title

The title of this program.

```
local title = "Essentia Controller"
```

- **Type:** `string`
- **Default:** `"Essentia Controller"`

---

version

The version of this program.

```
local version = "v1.0"
```

- **Type:** `string`
- **Default:** `"v1.0"`

Internal Properties

Name	Type	Default
nbtPeripherals	`table`	`{}`
sModem	`sModem`	`nil`

nbtPeripherals

A list containing the wrapped nbt observer peripherals.

```
local nbtPeripherals = {}
```

- **Type:** `table`
- **Default:** `{}`

---

sModem

An instance of a secure modem object

```
local sModem = nil
```

- **Type:** `sModem`
- **Default:** `nil`

Local Functions

Note: Local functions are defined in a local scope and thus can only be used within this program. They mainly server as helper functions for the program itself.

---

getLocalID(aspect)

Converts aspect name to local ID using *servedAspects*.

```
local funtion getLocalID(aspect)
    ...
    return localID
end
```

Arguments:

Name	Type	Default	Description
aspect	`string`	`nil`	Aspect to convert to local ID.

Returns:

Type	Description
`number`	Local ID of **aspect** or 0 if **aspect** is not served.

Example:

```
local servedAspects = {"terra", "aqua", "aer", "ignis", "ordo"}
local localID = getLocalID("aer")
```

In this case, `localID` would equal to `3`, since `aer` is the third element in the table

---

Note: The table `servedAspects` would normally be set as a *configurable property*

---

sendPulse(id)

Sends a redstone pulse on the specified channel through the bundled wire at *outputSide*.

```
local funtion snedPulse(id)
    ...
end
```

Arguments:

Name	Type	Default	Description
id	`number`	`nil`	Local ID of aspect / Channel to send a pulse to.

Returns: `nil`

Example:

```
sendPulse(4)
```

This would send a redstone pulse on the *outputSide* on the color channel corresponding to the number `2 ^ (id - 1)`, in this case `8`, which corresponds to the color `lightBlue` as seen here. Thus this command would send a pulse on the lightBlue channel.

5.4.2 Server

The program, that manages requests to controllers and broadcasts them to every specified controller

Contents:

Dependencies

- *ComLib*
- *DNSLib*
- *LogLib*

Configurable Properties

Name	Type	Default
controllerDomains	`table`	`{}`
modemSide	`string`	`"top"`

controllerDomains

A list containing all the domain names of the controllers, which are used to look up the addresses of the controllers.

```
local controllerDomains = {}
```

- **Type:** `table`
- **Default:** `{}`

Important: The specified domain names have to be registered in the *DNS Server*.

---

modemSide

The side the wireless modem is connected to the computer.

```
local modemSide = "top"
```

- **Type:** `string`
- **Default:** `"top"`

Requests

- *FLOW*
- *PROBE*

FLOW

Release 5 essentia from the specified aspect. Fails if aspect is not serverd by controller or amount of essentia of specified aspect is less than 5.

Note: The server only broadcasts this request to the controllers specified with *controllerDomains*. The controllers actually handle the requests.

```
{head = "FLOW", contents = {aspect = ""}}
```

request Contents:

Name	Type	Default	Description
aspect	`string`	`nil`	Aspect of which 5 essentia should be released.

Response contents: `nil`

PROBE

Probe the amount of specified aspect in jar. Fails if aspect is not serverd by controller.

Note: The server only broadcasts this request to the controllers specified with *controllerDomains*. The controllers actually handle the requests.

```
{head = "PROBE", contents = {aspect = ""}}
```

Request Contents:

Name	Type	Default	Description
aspect	`string`	`nil`	Aspect of which the amount should be probed.

Response contents:

Name	Type	Description
aspect	`string`	Aspect in probed jar.
amount	`number`	Amount of stored essentia in probed jar.

Warning: If `aspect` in response contents doesn't match with `aspect` in request contents, then the order of *nbtPeripheralTags* is most likely faulty.

Constants

Name	Type	Value
title	`string`	`"Essentia Server"`
version	`string`	`"v1.0"`

title

The title of this program.

```
local title = "Essentia Server"
```

- **Type:** `string`
- **Default:** `"Essentia Server"`

---

version

The version of this program.

```
local version = "v1.0"
```

- **Type:** `string`
- **Default:** `"v1.0"`

Internal Properties

Name	Type	Default
controllerAddresses	`table`	`{}`
sModem	`sModem`	`nil`

controllerAddresses

A list containing the resolved addresses of the controllers specified in *controllerDomains*.

```
local controllerAddresses = {}
```

- **Type:** `table`
- **Default:** `{}`

---

sModem

An instance of a secure modem object

```
local nbtPeripherals = {}
```

- **Type:** `sModem`
- **Default:** `nil`

5.4.3 Display

Danger: Work in progress

Contents:

CHAPTER

SIX

KEYCARD

Danger: Work in progress

CHAPTER

SEVEN

LETTER STUDIO

Danger: Work in progress

CHAPTER

EIGHT

SERVER CLIENT FRAMEWORK

Danger: Work in progress

CHAPTER

NINE

TURTLE

Danger: Work in progress

CHAPTER

TEN

COMLIB

A library for communicating between two computers in a server and client relation

Contents:

10.1 Dependencies

- *ecnet (Third Party)*

10.2 Functions

10.2.1 open()

Opens secure connection on modem on specified side

```
function comlib.open(side)
    ...
    return sModem
end
```

Arguments:

Name	Type	Default	Description
side	`string`	`nil`	Side of the modem for the secure connection.

Returns:

Type	Description
`table`	Instance of secure modem object.

Example:

```
local comlib = require("comlib")
local sModem = comlib.open("front")
```

This would open a secure connection on the modem at the front of the computer, accessible with the `sModem` variable.

10.2.2 getAddress()

Returns address of current computer

```
function comlib.getAddress()
    ...
    return address
end
```

Arguments: `nil`

Returns:

Type	Description
`string`	Address of current computer.

Example:

```
local comlib = require("comlib")
print(comlib.getAddress())
```

This would print the address of the current computer, e.g. `b38a:a780:bd82:cd56:195f`

10.2.3 sendRequest()

Creates a request packet with the status `REQUEST`, sends it to the specified address and will wait for and return a response packet. This function will return `-1` if the receiving operation times out.

```
function sendRequest(address, header, contents, timeout)
    ...
    return response
end
```

Arguments:

Name	Type	Default	Description
address	`string`	`nil`	Address of the receiver.
header	`string`	`nil`	Header of the request packet.
contents	`table`	`nil`	Additional contents of the request packet.
timeout	`number`	`3`	Number of seconds before the timeout would get triggered.

Note: Additional contents depend on the type of request and what the receiver is expecting

Returns:

Type	Description
`table`	Received response packet.

Warning: This function returns `-1` instead of the above, if one of these conditions is met:

- Not being able to connect to the address.
- Sender of the response packet is `nil`.
- Deserialized response packet is `nil`.
- If response header does not match request header.

Example:

```
local comlib = require("comlib")
comlib.sendRequest("873b:a87c:fe93:846:c9d3", "GET", {key = "Hello"}, 3)
```

In this example, a request packet with the **header** `"GET"` and the **contents** `{key = "Hello"}` will be sent to `"873b:a87c:fe93:846:c9d3"`. For this example, we will assume that the receiver expects `key` in **contents**, which is why `key = "Hello"` is specified here. If no response is received within `3` seconds, the function would timeout and return `-1`.

Created packet: `{head = "GET", status = "REQUEST", contents = {key = "Hello"}}`

10.2.4 sendResponse()

Sends a response to the specified receiver with specified head, status and additional contents

```
function comlib.sendResponse(address, header, status, contents)
    ...
end
```

Arguments:

Name	Type	Default	Description
address	`string`	`nil`	Address of the receiver.
header	`string`	`nil`	Header of the response packet.
status	`string`	`nil`	Status of the response packet (e.g. "OK" or "FAIL").
contents	`table`	`nil`	Additional contents to add to the packet.

Note: Additional contents depend on the type of response and what the receiver is expecting

Returns: `nil`

Example:

```
local comlib = require("comlib")
comlib.sendResponse("b38a:a780:bd82:cd56:195f", "GET", "OK", {value = "Test"})
```

In this example, a response packet with the header `"GET"` and the status `"OK"` will be sent to `"b38a:a780:bd82:cd56:195f"`. For this example, we will assume that the receiver expects `value` in **contents**, which is why `value = "Test"` is specified here.

Created packet: `{head = "GET", status = "OK", contents = {value = "Test"}}`

10.2.5 broadcast()

Broadcasts a request to multiple receivers and collects all responses. If a response of a receiver fails (*sendRequest* returns `-1`), its response will fall back to this one: `{head = header, status = "FAIL", contents = {}}`.

Note: This function calls *sendRequest()* for each receiver.

```
function comlib.broadcast(addresses, header, contents, timeout)
  ...
  return responses
end
```

Arguments:

Name	Type	Default	Description
addresses	`table`	`nil`	Addresses of the receivers.
header	`string`	`nil`	Header of the response packet.
contents	`table`	`nil`	Additional contents to add to the packet.
timeout	`number`	`3`	Number of seconds before the timeout would get triggered.

Note: Additional contents depend on the type of response and what the receiver is expecting

Returns:

Type	Description
`table`	Received response packets.

Example:

```
local comlib = require("comlib")
local responses = {}
local receivers = {
  "01ae:4a1e:0195:6e6e:56af",
  "e990:4b07:961f:0b4b:c50a:",
  "7a57:2c08:9d08:7bac:91ff"
}

responses = comlib.broadcast(receivers, "TEST", {})
```

This would send the packet `{head = "TEST", status = "REQUEST", contents = {}}` to all three specified receivers and would store there responses in `responses`.

CHAPTER

ELEVEN

DNSLIB

A library for various DNS operations

Important: To use this library, a functioning *DNS Server* is required.

Contents:

11.1 Dependencies

- *ComLib*

11.2 Properties

Name	Type	Default
dnsAddress	`string`	`""`

11.2.1 dnsAddress

The address of the DNS Server.

```
dnslib.dnsAddress = ""
```

- **Type:** `string`
- **Default:** `""`

Note: This property is set using *init()*

11.3 Functions

11.3.1 init()

Initializes DNSLib and sets *dnsAddress* to an address read from the `/.dnsAddress` file. If the file does not exist the function will return `-1`.

```
function dnslib.init()
    ...
    return success
end
```

Arguments: `nil`

Returns:

Type	Description
`boolean`	Initialization success.

> **Warning:** This function returns `-1` instead of the above, if one of these conditions is met:
>
> - `/.dnsAddress` file does not exist.
> - Address read from `/.dnsAddress` is empty or `nil`.

Example:

```
local dnslib = require("dnslib")
dnslib.init()
```

This would initialize DNSLib and set *dnsAddress* to an addres found in `/.dnsAddress`.

11.3.2 lookup()

Looks up the specified domain and returns the address of the registered server.

```
function dnslib.lookup(domain)
    ...
    return address
end
```

Arguments:

Name	Type	Default	Description
domain	`string`	`nil`	Domain to look up.

Returns:

Type	Description
`string`	Address of the corresponding registered server.

> **Warning:** This function returns `-1` instead of the above, if one of these conditions is met:
>
> - *sendRequest()* has errored (Has returned `-1` as well).

Example:

```
local dnslib = require("dnslib")
local address = dnslib.lookup("test.com")
```

Here `address` would contain the address the *DNS Server* knows for `"test.com"`

11.3.3 lookupMultiple()

Looks up the multiple domains and returns the addresses of the registered servers.

> **Note:** This function calls *lookup()* for each domain.

```
function dnslib.lookupMultiple(domains)
    ...
    return addresses
end
```

Arguments:

Name	Type	Default	Description
domains	`table`	`nil`	Domains to look up.

Returns:

Type	Description
`table`	Addresses of the corresponding registered servers.

Warning: This function returns `-1` instead of the above, if one of these conditions is met:

- Any lookup operation failed (*lookup()* returned `-1`).

Example:

```
local dnslib = require("dnslib")
local domains = {
    "test1.com",
    "test2.com"
}

local addresses = dnslib.lookupMultiple(domains)
```

Here `addresses` would contain the addresses of both, `test1.com` and `test2.com`. If either one of those would cause an error in *lookup()*, `addresses` would contain `-1`.

CHAPTER

TWELVE

LOGLIB

A library for logging to the screen, which is either the built-in computer screen or a connected monitor of any size.

Contents:

12.1 Functions

12.1.1 init()

Initializes LogLib with setting the title in the computer and, if a monitor is present, setting the title in the monitor and redirecting output to the monitor.

```
function loglib.init(title, version)
    ...
end
```

Arguments:

Name	Type	Default	Description
title	`string`	`nil`	Title that will be displayed at the top of the screen.
version	`string`	`nil`	Version that will be displayed at the top of the screen.

Returns: `nil`

Example:

```
local loglib = require("loglib")
loglib.init("Test PC", "V1.0")
```

This would initialize LogLib and would display the text `Test PC V1.0` at the top of the screen.

Warning: If a monitor is present, **all** text ouput will be redirected onto the monitor.

Note: Screen refers to the active output. If no monitor is present, the active output is the computer screen. If a monitor is present, the active output is the monitor.

12.1.2 log()

Loggs a message with a tag and the current in-game time to the screen

```
function loglib.log(tag, msg)
    ...
end
```

Arguments:

Name	Type	Default	Description
tag	`string`	`nil`	Tag that will be displayed.
msg	`string`	`nil`	Message that will be displayed.

Example:

```
local loglib = require("loglib")
loglib.log("Test", "This is a test message")
```

This would display the text `<5.345> [Test] This is a test message`, if we assume the current in-game time is `5.345`.

Returns: `nil`

Important: LogLib has to be initialized when using this function.

12.2 Local Functions

Note: Local functions are defined in a local scope and thus can only be used within this program. They mainly server as helper functions for the program itself.

- *setTitle(title, version)*

12.2.1 setTitle(title, version)

Sets the title displayed at the top of the screen.

```
function loglib.setTitle(title, version)
    ...
end
```

Arguments:

Name	Type	Default	Description
title	`string`	`nil`	Title that will be displayed at the top of the screen.
version	`string`	`nil`	Version that will be displayed at the top of the screen.

Returns: `nil`

Example:

```
local loglib = require("loglib")
loglib.setTitle("Test PC", "V1.0")
```

This would display the text `Test PC V1.0` at the top of the screen.

CHAPTER

THIRTEEN

UILIB

A library for various UI actions and elements, like drawing and chacking for events.

Contents:

13.1 Classes

13.1.1 Style

This is a class, that contains and handles styles for UI elements for different states.

Contents:

States

The possible states for UI element styles. The flag **pressed** refers to the flag of the UI element, that is raised when it gets clicked. The flag **disabled** refers to the flag of the UI element, that can be raised to disable said element.

State	pressed	disabled
Normal / Default	`false`	`false`
Pressed	`true`	`false`
disabled	`true` or `false`	`true`

Properties

Name	Type	Default
normalFG	number	colors.white
normalBG	number	colors.gray
pressedFG	number	colors.white
pressedBG	number	colors.lime
disabledFG	number	colors.gray
disabledBG	number	colors.lightGray

normalFG

Foreground color in default state.

```
uilib.Style.normalFG = colors.white
```

- **Type:** number
- **Default:** colors.white

Note: It is recommended to set this property with the Colors(API). If you choose to set this property with numeric values directly, please consult the table at the bottom of this page for the correct values.

normalBG

Background color in default state.

```
uilib.Style.normalBG = colors.gray
```

- **Type:** number
- **Default:** colors.gray

Note: It is recommended to set this property with the Colors(API). If you choose to set this property with numeric values directly, please consult the table at the bottom of this page for the correct values.

pressedFG

Foreground color in pressed state.

```
uilib.Style.pressedFG = colors.white
```

- **Type:** `number`
- **Default:** `colors.white`

Note: It is recommended to set this property with the Colors(API). If you choose to set this property with numeric values directly, please consult the table at the bottom of this page for the correct values.

---

pressedBG

Background color in pressed state.

```
uilib.Style.pressedBG = colors.lime
```

- **Type:** `number`
- **Default:** `colors.lime`

Note: It is recommended to set this property with the Colors(API). If you choose to set this property with numeric values directly, please consult the table at the bottom of this page for the correct values.

---

disabledFG

Foreground color in disabled state.

```
uilib.Style.disabledFG = colors.gray
```

- **Type:** `number`
- **Default:** `colors.gray`

Note: It is recommended to set this property with the Colors(API). If you choose to set this property with numeric values directly, please consult the table at the bottom of this page for the correct values.

---

disabledBG

Background color in disabled state.

```
uilib.Style.disabledBG = colors.lightGray
```

- **Type:** `number`
- **Default:** `colors.lightGray`

Note: It is recommended to set this property with the Colors(API). If you choose to set this property with numeric values directly, please consult the table at the bottom of this page for the correct values.

Functions

- *new()*
- *getColors()*

new()

Creates a new instance of *Style* and returns it.

```
function uilib.Style:new(normalFG, normalBG, pressedFG, pressedBG, disabledFG,␣
↪disabledBG)
    ...
    return style
end
```

Arguments:

Name	Type	Default	Description
normalFG	`number`	`colors.white`	Foreground color for default state.
normalBG	`number`	`colors.gray`	Background color for default state.
pressedFG	`number`	`colors.white`	Foreground color for pressed state.
pressedBG	`number`	`colors.lime`	Background color for pressed state.
disabledFG	`number`	`colors.gray`	Foreground color for disabled state.
disabledBG	`number`	`colors.lightGray`	Background color for disabled state.

Returns:

Type	Description
Style	Instance of *Style* with specified properties.

Example:

```
local uilib = require("uilib")
local style = uilib.Style:new(colors.red. colors.lightGray)
```

This would create an instance of *Style* with `colors.red` as *normalFG* and `colors.lightGray` as *normalBG*. All other properties would be set to their respective default value.

---

getColors()

Returns the foreground and background color for the current state of the UI element.

```
function uilib.Style:getColors(pressed, disabled)
    ...
    return fg, bg
end
```

Arguments:

Name	Type	Default	Description
pressed	`boolean`	`false`	Flag if UI element has been clicked.
disabled	`boolean`	`false`	Flag if UI element has been disabled.

Returns:

Type	Description
`number`	Foreground color for current state of UI element.
`number`	Background color for current state of UI element.

Example:

```
local uilib = require("uilib")
local style = uilib.Style:new(colors.red. colors.lightGray)
local fg, bg = style:getColors(false, false)
```

This would create an instance of *Style* with `colors.red` as *normalFG* and `colors.lightGray` as *normalBG*. After that it would get the foregournd and background color for the default state, since `pressed` and `disabled` are both `false`. So `fg` and `bg` would contain `colors.red` and `colors.lightGray` respectively.

13.1.2 Group

A UI element, that holds multiple other UI elements and draws them together.

Contents:

- *Properties*
- *Functions*

Properties

Name	Type	Default
normalFG	number	nil
normalBG	number	nil
pressedFG	table	{}

x

X component of the position on the screen.

```
uilib.Group.x = nil
```

- **Type:** number
- **Default:** nil

---

y

Y component of the position on the screen.

```
uilib.Group.y = nil
```

- **Type:** number
- **Default:** nil

---

elements

List of all UI elements included within the group.

```
uilib.Group.elements = {}
```

- **Type:** table
- **Default:** {}

---

visible

Contains information about the group being visible or not.

```
uilib.Group.visible = true
```

- **Type:** boolean
- **Default:** true

Note: Please use *show()* to enable visibility and *hide()* to disable visibility of the group.

Functions

- *new()*
- *draw()*
- *add()*
- *remove()*
- *get()*
- *show()*
- *hide()*

new()

Function to create a new instance of *Group*.

```
function M.Group:new(x, y, elements)
  ...
  return group
end
```

Arguments:

Name	Type	Default	Description
x	`number`	`nil`	X component of position of the group.
y	`number`	`nil`	Y component of position of the group.
elements	`table`	`nil`	List of all contained UI elements.

Important: The `elements` list must contain the elements with a string key attached to them, e.g. `{label = elementVar}`.

Important: When you add a UI element to a group, the `x` and `y` parameters will become local to the group, which means that the actual position of the UI element would be: `(group.x + element.x - 1, group.y + element.y - 1)`.

Returns:

Type	Description
uilib.Group	Instance of *Group* with specified properties.

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})
```

This would create an instance of a *Group* and a *Label* within the created group. The Label would be drawn at position `(5, 6)`.

---

draw()

Function to draw the group.

```
function M.Group:draw()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:draw()
```

This would create an instance of a *Group* and a *Label* within the created group and draw it to the screen.

---

add()

Function to add a UI element to the group.

```
function M.Group:add(element, id)
  ...
end
```

Arguments:

Name	Type	Default	Description
element	`table`	`nil`	UI element to add.
id	`string`	`nil`	ID to refer to the UI element.

Important: When you add a UI element to a group, the `x` and `y` parameters will become local to the group, which means that the actual position of the UI element would be: `(group.x + element.x - 1, group.y + element.y - 1)`.

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {})

group:add(label, "label")
```

This would create an instance of a *Group* and a *Label*, which is being added to the group with the ID `"label2`.

---

remove()

Function to remove a UI element from the group.

```
function M.Group:remove(id)
  ...
end
```

Arguments:

Name	Type	Default	Description
id	`string`	`nil`	ID of the element.

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:remove("label")
```

This would create an instance of a *Group* and a *Label* within the created group. It would then remove the created label from the group.

---

get()

Function to get a specific UI element from the group.

```
function M.Group:get(id)
  ...
end
```

Arguments:

Name	Type	Default	Description
id	`string`	`nil`	ID of the UI element.

Returns:

Type	Description
`table`	UI element with the specified id in the group.

> **Warning:** This function returns `-1` instead of the above, if one of these conditions is met:
>
> - No UI element with the specified ID exists.

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

local labelAgain = group:get("label")
```

This would create an instance of a *Group* and a *Label* within the created group. After that it would store the label element with the ID `"label"` in `labelAgain`.

---

show()

Function to make the group visible.

```
function uilib.Group:show()
    ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:show()
```

This would create an instance of a *Group* and a *Label* within the created group and make it visible.

---

hide()

Function to make the group invisible.

```
function uilib.Group:hide()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:hide()
```

This would create an instance of a *Group* and a *Label* within the created group and make it invisible.

13.1.3 Page Handler

> **Danger:** Work in Progress
>
> The contents are copied from *Group* and have not been adapted yet.

A UI element, that holds multiple other UI elements and draws them together.

Contents:

- *Properties*
- *Functions*

Properties

Name	Type	Default
normalFG	`number`	`nil`
normalBG	`number`	`nil`
pressedFG	`table`	`{}`

x

X component of the position on the screen.

```
uilib.Group.x = nil
```

- **Type:** number
- **Default:** nil

---

y

Y component of the position on the screen.

```
uilib.Group.y = nil
```

- **Type:** number
- **Default:** nil

---

elements

List of all UI elements included within the group.

```
uilib.Group.elements = {}
```

- **Type:** table
- **Default:** {}

---

visible

Contains information about the group being visible or not.

```
uilib.Group.visible = true
```

- **Type:** boolean
- **Default:** true

Note: Please use *show()* to enable visibility and *hide()* to disable visibility of the group.

Functions

new()

Function to create a new instance of *Group*.

```
function M.Group:new(x, y, elements)
  ...
  return group
end
```

Arguments:

Name	Type	Default	Description
x	`number`	`nil`	X component of position of the group.
y	`number`	`nil`	Y component of position of the group.
elements	`table`	`nil`	List of all contained UI elements.

Important: The `elements` list must contain the elements with a string key attached to them, e.g. `{label = elementVar}`.

Important: When you add a UI element to a group, the `x` and `y` parameters will become local to the group, which means that the actual position of the UI element would be: `(group.x + element.x - 1, group.y + element.y - 1)`.

Returns:

Type	Description
uilib.Group	Instance of *Group* with specified properties.

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})
```

This would create an instance of a *Group* and a *Label* within the created group. The Label would be drawn at position `(5, 6)`.

---

draw()

Function to draw the group.

```
function M.Group:draw()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:draw()
```

This would create an instance of a *Group* and a *Label* within the created group and draw it to the screen.

---

add()

Function to add a UI element to the group.

```
function M.Group:add(element, id)
  ...
end
```

Arguments:

Name	Type	Default	Description
element	`table`	`nil`	UI element to add.
id	`string`	`nil`	ID to refer to the UI element.

Important: When you add a UI element to a group, the `x` and `y` parameters will become local to the group, which means that the actual position of the UI element would be: `(group.x + element.x - 1, group.y + element.y - 1)`.

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {})

group:add(label, "label")
```

This would create an instance of a *Group* and a *Label*, which is being added to the group with the ID "label2.

---

remove()

Function to remove a UI element from the group.

```
function M.Group:remove(id)
  ...
end
```

Arguments:

Name	Type	Default	Description
id	string	nil	ID of the element.

Returns: nil

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:remove("label")
```

This would create an instance of a *Group* and a *Label* within the created group. It would then remove the created label from the group.

---

get()

Function to get a specific UI element from the group.

```
function M.Group:get(id)
  ...
end
```

Arguments:

Name	Type	Default	Description
id	`string`	`nil`	ID of the UI element.

Returns:

Type	Description
`table`	UI element with the specified id in the group.

> **Warning:** This function returns `-1` instead of the above, if one of these conditions is met:
>
> - No UI element with the specified ID exists.

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

local labelAgain = group:get("label")
```

This would create an instance of a *Group* and a *Label* within the created group. After that it would store the label element with the ID `"label"` in `labelAgain`.

---

show()

Function to make the group visible.

```
function uilib.Group:show()
    ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:show()
```

This would create an instance of a *Group* and a *Label* within the created group and make it visible.

---

hide()

Function to make the group invisible.

```
function uilib.Group:hide()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪black))
local group = uilib.Group:new(2, 2, {label = label})

group:hide()
```

This would create an instance of a *Group* and a *Label* within the created group and make it invisible.

13.1.4 Label

A UI element, that displays plain text.

Fig. 1: Labels with two styles.

Contents:

- *Properties*
- *Functions*

Properties

Name	Type	Default
text	`string`	`nil`
x	`number`	`nil`
y	`number`	`nil`
style	*uilib.Style*	*Default Style*
visible	`boolean`	`true`

text

Text, which the label will display.

```
uilib.Label.text = nil
```

- **Type:** `string`
- **Default:** `nil`

---

x

X component of the position on the screen.

```
uilib.Label.x = nil
```

- **Type:** `number`
- **Default:** `nil`

---

y

Y component of the position on the screen.

```
uilib.Label.y = nil
```

- **Type:** `number`
- **Default:** `nil`

---

style

Style of the label.

```
uilib.Label.style = uilib.Style:new()
```

- **Type:** *uilib.Style*
- **Default:** *Default Style*

---

visible

Contains information about the label being visible or not.

```
uilib.Label.visible = true
```

- **Type:** `boolean`
- **Default:** `true`

Note: Please use *show()* to enable visibility and *hide()* to disable visibility of the button.

Functions

- *new()*
- *draw()*
- *show()*
- *hide()*

new()

Function to create a new instance of *Label*.

```
function M.Label:new(text, x, y, style)
  ...
  return label
end
```

Arguments:

Name	Type	Default	Description
text	`string`	`nil`	Text, which the label will display.
x	`number`	`nil`	X component of position of the label.
y	`number`	`nil`	Y component of position of the label.
style	*uilib.Style*	*Default Style*	Style of the label.

Note: Labels can only use the *default state* for styling.

Returns:

Type	Description
uilib.Label	Instance of *Label* with specified properties.

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪gray))
```

This would create an instance of a *Label* with the text `I am a Label!` with `colors.red` text on `colors.gray` background at position `(4, 5)`.

---

draw()

Function to draw the label.

```
function M.Label:draw()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪gray))
label:draw()
```

This would create an instance of *Label* and draw it to the screen.

---

show()

Function to make the label visible.

```
function uilib.Label:show()
    ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪gray))
label:show()
```

This would create an instance of *Label* and make it visible.

---

hide()

Function to make the label invisible.

```
function uilib.Label:hide()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local label = uilib.Label("I am a Label!", 4, 5, uilib.Style:new(colors.red, colors.
↪gray))
label:hide()
```

This would create an instance of *Label* and make it invisible.

13.1.5 Button

A UI element, that is clickable and executes a function when it is clicked.

Fig. 2: Button in three states: Default, Pressed and Disabled.

Contents:

- *Properties*
- *Functions*

Properties

Name	Type	Default
text	`string`	`nil`
x	`number`	`nil`
y	`number`	`nil`
w	`number`	`nil`
h	`number`	`nil`
style	*uilib.Style*	*Default Style*
action	`function`	`nil`
args	`table`	`nil`
toggle	`boolean`	`false`
visible	`boolean`	`true`
pressed	`boolean`	`false`
disabled	`boolean`	`false`

text

Text, which is displayed on the button.

```
uilib.Button.text = nil
```

- **Type:** `string`
- **Default:** `nil`

Note: If the text is longer than the *width* of the button with padding in mind, the text will just get cut off.

---

x

X component of the position on the screen.

```
uilib.Button.x = nil
```

- **Type:** `number`
- **Default:** `nil`

---

y

Y component of the position on the screen.

```
uilib.Button.y = nil
```

- **Type:** `number`
- **Default:** `nil`

---

w

Width of the button.

```
uilib.Button.w = nil
```

- **Type:** `number`
- **Default:** `nil`

---

h

Height of the button.

```
uilib.Button.h = nil
```

- **Type:** `number`
- **Default:** `nil`

---

style

Style of the button.

```
uilib.Button.style = uilib.Style:new()
```

- **Type:** *uilib.Style*
- **Default:** *Default Style*

---

action

Function, that should be run when the button is clicked.

```
uilib.Button.action = nil
```

- **Type:** `function`
- **Default:** `nil`

---

args

Arguments to the function specified with *action*.

```
uilib.Button.args = nil
```

- **Type:** `table`
- **Default:** `nil`

> **Warning:** The order of arguments within the table should be the same order as the function specified with *action* is expecting it.

---

toggle

Enables toggle mode for the button.

```
uilib.Button.toggle = false
```

- **Type:** boolean
- **Default:** false

visible

Contains information about the button being visible or not.

```
uilib.Button.visible = true
```

- **Type:** boolean
- **Default:** true

Note: Please use *show()* to enable visibility and *hide()* to disable visibility of the button.

pressed

Contains information about the button being clicked or not.

```
uilib.Button.pressed = false
```

- **Type:** boolean
- **Default:** false

Important: This property is not meant for being set directly and is usually only set by *clickEvent()*.

disabled

Contains information about the button being disabled or not.

```
uilib.Button.disabled = false
```

- **Type:** boolean
- **Default:** false

Note: Please use *disable()* to toggle if the button should be disabled of the button.

Functions

- *new()*
- *draw()*
- *clickEvent()*
- *disable()*
- *show()*
- *hide()*

new()

Creates a new instance of *Button* and returns it.

```
function uilib.Button:new(text, x, y, w, h, action, args, toggle, style)
    ...
    return button
end
```

Arguments:

Name	Type	Default	Description
text	`string`	`nil`	Text to be displayed on the button.
x	`number`	`nil`	X component of position of the button.
y	`number`	`nil`	Y component of position of the button.
w	`number`	`nil`	Width of the button.
h	`number`	`nil`	Height of the button.
action	`function`	`nil`	Function, that will be executed, when the button is clicked.
args	`table`	`nil`	Arguments for the function specified above.
toggle	`boolean`	`false`	Enables toggle mode for the button.
style	*uilib.Style*	*Default Style*	Style of the button for various states.

Returns:

Type	Description
uilib.Button	Instance of *Button* with specified properties.

Example:

```
local uilib = require("uilib")

function onClick(name)
    print("Hello " .. name)
end

local btn = uilib.Button:new("Test", 2, 2, 6, 3, onClick, {"User"}, false, uilib.
↪Style:new())
```

A *Button* with the text `Test` would be cerated at position `(2, 2)` with a size of `6 x 3` pixels. It would execute `onClick("User")` when it would be clicked. Toggle mode is disabled for this button, so this button is in one-shot mode. The button would have the *default Style*.

---

draw()

Function to draw the button on the screen.

```
function uilib.Button:draw()
    ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")

function onClick(name)
    print("Hello " .. name)
end

local btn = uilib.Button:new("Test", 2, 2, 6, 3, onClick, {"User"}, false, uilib.
↪Style:new())
btn:draw()
```

An instance of *Button* will be created with the *new()* method and the returned button will be drawn to the screen.

---

clickEvent()

Function that checks if a click event was on the button and executes *action* if it was. It will not execute the function if the button is either disabled or not visible.

```
function uilib.Button:clickEvent(ex, ey)
    ...
    return ret
end
```

Arguments:

Name	Type	Default	Description
ex	`number`	`false`	X component of the event position.
ey	`number`	`false`	Y component of the event position.

Returns:

Type	Description
`table`	Return values from the function specified in *action*

Example:

```
local uilib = require("uilib")

function onClick(name)
    print("Hello " .. name)
end

local btn = uilib.Button:new("Test", 2, 2, 6, 3, onClick, {"User"}, false, uilib.
↪Style:new())

while true do
  local e, btn, x, y = os.pullEvent("mouse_click")
  btn:clickEvent(x, y)
end
```

An instance of *Button* will be created with the *new()* method. After that it will wait for a `mouse_click` event. If this event happened inside the button, the *action* function will be executed.

---

disable()

Function to disable or enable the button.

```
function uilib.Button:disable(status)
    ...
end
```

Arguments:

Name	Type	Default	Description
status	`boolean`	Inverse of *disabled*	Specifies if it should be disabled or not.

Note: If **status** is omitted, the function toggles the *disabled* flag. If **status** is specified however, the function will set the *disabled* flag to **status**.

Returns: `nil`

Example:

```
local uilib = require("uilib")

function onClick(name)
    print("Hello " .. name)
end
```

(continues on next page)

(continued from previous page)

```
local btn = uilib.Button:new("Test", 2, 2, 6, 3, onClick, {"User"}, false, uilib.
↪Style:new())
btn:disable(true)
```

An instance of *Button* will be created with the *new()* method. Adter that the button will be disabled.

---

show()

Function to make the button visible.

```
function uilib.Button:show()
    ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")

function onClick(name)
    print("Hello " .. name)
end

local btn = uilib.Button:new("Test", 2, 2, 6, 3, onClick, {"User"}, false, uilib.
↪Style:new())
btn:show()
```

An instance of *Button* will be created with the *new()* method. Adter that the button will be made visible.

---

hide()

Function to make the button invisible.

```
function uilib.Button:hide()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")

function onClick(name)
    print("Hello " .. name)
end

local btn = uilib.Button:new("Test", 2, 2, 6, 3, onClick, {"User"}, false, uilib.
↪Style:new())
btn:hide()
```

An instance of *Button* will be created with the *new()* method. Adter that the button will be made invisible.

13.1.6 ProgressBar

A UI element that shows a bar, that can be filled to different amounts in multiple orientations.

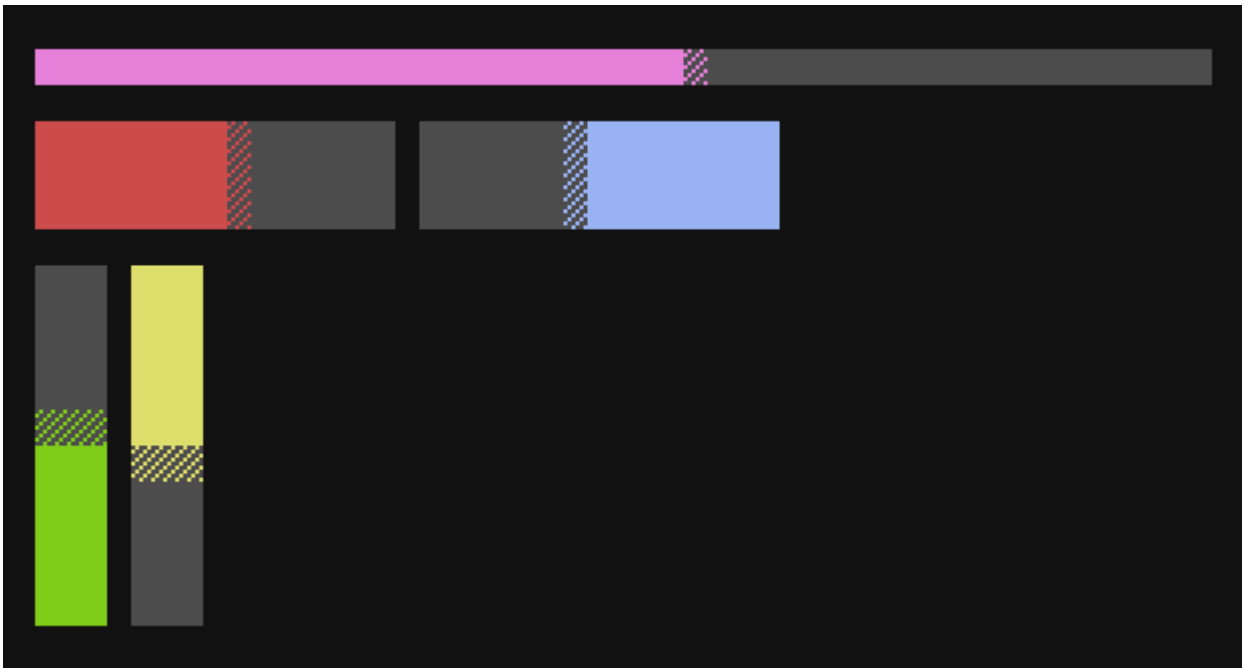

Fig. 3: Different kinds of porgress bar.

Contents:

- *Properties*
- *Functions*

Properties

Name	Type	Default
minVal	number	nil
maxVal	number	nil
val	number	nil
x	number	nil
y	number	nil
w	number	nil
h	number	nil
vertical	boolean	false
inverted	boolean	false
style	*uilib.Style*	*Default Style*
visible	boolean	true

minVal

Smallest value the progress bar can display.

```
uilib.ProgressBar.minVal = nil
```

- **Type:** number
- **Default:** nil

---

maxVal

Biggest value the progress bar can display.

```
uilib.ProgressBar.maxVal = nil
```

- **Type:** number
- **Default:** nil

---

val

Current value the progress bar should display.

```
uilib.ProgressBar.val = nil
```

- **Type:** number
- **Default:** nil

---

x

X component of the position on the screen.

```
uilib.ProgressBar.x = nil
```

- **Type:** number
- **Default:** nil

---

y

Y component of the position on the screen.

```
uilib.ProgressBar.y = nil
```

- **Type:** number
- **Default:** nil

---

w

Width of the progress bar.

```
uilib.ProgressBar.w = nil
```

- **Type:** number
- **Default:** nil

---

h

Height of the progress bar.

```
uilib.ProgressBar.h = nil
```

- **Type:** number
- **Default:** nil

---

vertical

Enables vertical mode for the progress bar.

```
uilib.ProgressBar.vertical = false
```

- **Type:** `boolean`
- **Default:** `false`

---

inverted

Enables inverted mode for the progress bar.

```
uilib.ProgressBar.inverted = false
```

- **Type:** `boolean`
- **Default:** `false`

---

style

Style of the progress bar.

```
uilib.ProgressBar.style = uilib.Style:new()
```

- **Type:** *uilib.Style*
- **Default:** *Default Style*

---

visible

Contains information about the progress bar being visible or not.

```
uilib.ProgressBar.visible = true
```

- **Type:** `boolean`
- **Default:** `true`

Note: Please use *show()* to enable visibility and *hide()* to disable visibility of the progress bar.

Functions

new()

Function to create a new instance of *ProgressBar*.

```
function M.ProgressBar:new(minVal, maxVal, val, x, y, w, h, vertical, inverted, style)
  ...
  return prog
end
```

Arguments:

Name	Type	Default	Description
minVal	`number`	`nil`	Smallest value the progress bar can display.
maxVal	`number`	`nil`	Biggest value the progress bar can display.
val	`number`	`nil`	Current value the progress bar should display.
x	`number`	`nil`	X component of position of the progress bar.
y	`number`	`nil`	Y component of position of the progress bar.
w	`number`	`nil`	Width of the progress bar.
h	`number`	`nil`	Height of the progress bar.
vertical	`boolean`	`false`	Enables vertical mode for the progres bar.
inverted	`boolean`	`false`	Enables inverted mode for the progres bar.
style	*uilib.Style*	*Default Style*	Style of the progress bar.

Note: Progress bars can only use the *default state* for styling.

Returns:

Type	Description
uilib.ProgressBar	Instance of *ProgressBar* with specified properties.

Example:

```
local uilib = require("uilib")
local prog = uilib.ProgressBar:new(0, 100, 35, 2, 2, 10, 1, false, false, uilib.
↪Style:new())
```

This would create an instance of *ProgressBar* with possible values between `0` and `100` and an initial value of `35`. The progress bar would be displayed at the position `(2, 2)` and would be `10 x 1` pixels in size. It would be in horizontal mode, since `vertical` is set to `false`. The style of the progress bar will be the default style.

draw()

Function to draw the progress bar.

```
function M.ProgressBar:draw()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local prog = uilib.ProgressBar:new(0, 100, 35, 2, 2, 10, 1, false, false, uilib.
↪Style:new())
prog:draw()
```

This would create an instance of *ProgressBar* and draw it to the screen.

---

show()

Function to make the progress bar visible.

```
function uilib.ProgressBar:show()
    ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local prog = uilib.ProgressBar:new(0, 100, 35, 2, 2, 10, 1, false, false, uilib.
↪Style:new())
prog:show()
```

This would create an instance of *ProgressBar* and make it visible.

---

hide()

Function to make the progress bar invisible.

```
function uilib.ProgressBar:hide()
  ...
end
```

Arguments: `nil`

Returns: `nil`

Example:

```
local uilib = require("uilib")
local prog = uilib.ProgressBar:new(0, 100, 35, 2, 2, 10, 1, false, false, uilib.
↪Style:new())
prog:hide()
```

This would create an instance of *ProgressBar* and make it invisible.

CHAPTER

FOURTEEN

ECNET

Danger: Work in progress

CHAPTER

FIFTEEN

SHA

Danger: Work in progress