

**LongOS Programming Guide**

|  |  |
| --- | --- |
| Author: | Gvin |
| Version: | 1.0 |

# 1 Executive Overview

This guide describes basics of programming for LongOS and using classes in Lua. It also contains several guidelines for those who would like to contribute his programs to the main LongOS repository (or just want to make their programming easier).

To use this document you should be familiar with such terms as “Object Oriented Programming” and “Classes”.

This guide was created for LongOS v1.0.

# 2 Table of Content

[1 Executive Overview 2](#_Toc366658181)

[2 Table of Content 3](#_Toc366658182)

[3 About This Guide 5](#_Toc366658183)

[1 OOP in Lua 6](#_Toc366658184)

[1.1 Basics of Class Creation 6](#_Toc366658185)

[1.2 Fields and Methods. Access Modifiers 7](#_Toc366658186)

[1.3 Inheritance 8](#_Toc366658187)

[1.4 Namespaces 9](#_Toc366658188)

[2 Introduction to Application 10](#_Toc366658189)

[2.1 Application 10](#_Toc366658190)

[2.2 Thread 10](#_Toc366658191)

[2.3 Window 11](#_Toc366658192)

[2.4 Components 12](#_Toc366658193)

[2.5 Menus 14](#_Toc366658194)

[3 Application Classes Specification 15](#_Toc366658195)

[3.1 Application Class 15](#_Toc366658196)

[3.2 Thread Class 17](#_Toc366658197)

[3.3 Window Class 18](#_Toc366658198)

[4 Components Classes Specification 22](#_Toc366658199)

[4.1 Components 22](#_Toc366658200)

[4.1.1 Component 22](#_Toc366658201)

[4.1.2 Label Component 24](#_Toc366658202)

[4.1.3 Button Component 25](#_Toc366658203)

[4.1.4 Edit Component 25](#_Toc366658204)

[4.1.5 VerticalScrollBar Component 26](#_Toc366658205)

[4.1.6 HorizontalScrollBar Component 27](#_Toc366658206)

[4.1.7 CheckBox Component 27](#_Toc366658207)

[4.1.8 ListBox Component 28](#_Toc366658208)

[4.2 PopupMenu 29](#_Toc366658209)

[5 System Classes Specification 31](#_Toc366658210)

[5.1 LongOS Class 31](#_Toc366658211)

[6 Guidelines 33](#_Toc366658212)

[6.1 Classes Parts 33](#_Toc366658213)

[6.2 Separation 33](#_Toc366658214)

[6.3 Comments 33](#_Toc366658215)

[6.4 Naming Rules 33](#_Toc366658216)

[6.5 Access Modifiers 34](#_Toc366658217)

[6.6 Errors Processing 34](#_Toc366658218)

[7 Conclusion 35](#_Toc366658219)

# 3 About This Guide

In this guide you can find information about basics of programming for LongOS. It consists of several sections:

* [Section 1: OOP in Lua](#_1_OOP_in)

The basics of OOP in Lua, creation and usage of classes.

* [Section 2: Introduction to Application](#_2_Introduction_to)

Some information about what does “application” means for LongOS.

* [Section 3: Application Classes Specification](#_3_Application_Classes)

Full specification of application classes with detailed information.

* [Section 4: Components Classes Specification](#_4_Components_Classes)

Full specification of components classes with detailed information.

* [Section 5: System Classes Specification](#_5_System_Classes)

Full specification of system classes with detailed information.

* [Section 6: Guidelines](#_6_Guidelines)

Several guidelines for those who would like to have clear and beautiful code.

# 1 OOP in Lua

OOP means Object Oriented Programming. Its basics will not be described here. If you would like to know more about it search the internet.

In this section is represented the OOP programming in Lua. This includes:

* [The basics of class creation in Lua](#_1.1_Basics_of)
* [Fields and methods. Access modifiers](#_1.2_Fields_and)
* [Inheritance realization](#_1.3_Inheritance)
* [Namespaces](#_1.4_Namespaces)

## 1.1 Basics of Class Creation

In Lua language as-it-is there are no classes at all. But there is an interesting data structure — table. With some coding it can be used to create classes and get access to almost all OOP principles.

To start creating classes in Lua you should first create function “Class” which you can find here: “LongOS/Classes/SystemClasses/ClassBase.lua”.

To create simple class you should create code like this:

local ClassExample = Class(Object, function(this)

Object.init(this, ‘ClassExample’);

end)

This class contains no fields, methods and didn’t receive any parameters in the constructor. To create an instance of in simply type:

Local instanceExample = ClassExample();

Function Class is responsible for creation of the class. First parameter of this function is parent class (for classes without parent it should be Object) and the other is special initialization function.

## 1.2 Fields and Methods. Access Modifiers

Fields and methods creation in classes are very similar to creation of variables and functions in standard function-style program. Access modifiers are realized through adding local prefix to variables and functions or inserting them into class table (because classes are tables).

To create private variable you should type:

local ClassWithVariables = Class(Object, function(this)

Object.init(this, ‘ClassWithVariables’);

local privateVariable = 3;

this.PublicVariable = ‘hello’;

end)

In the class above you can see the variables of two different access types: private and public. Private variables are only accessible from functions and other variables that come below their declaration in code and only inside their base class. Public variables can be accessed anywhere in code but to get access to them you must prefix their name with “**this.**”.

Methods creation is very similar:

local ClassWithMethods = Class(Object, function(this)

Object.init(this, ‘ClassWithMethods’);

local function privateMethod()

print(‘Private method.’);

end

function this:GlobalMethod()

print(‘Global method.’);

end

end)

In the class above you can see two methods: private and public. Private method, like private variable, can be used only in the code below its declaration and only inside its base class. Public methods can be used anywhere but to get access to them you must prefix their name with “**this:**”.

Protected access modifier didn’t exist in Lua.

## 1.3 Inheritance

To create class inherited from some parent class you must specify the parent’s class as the first parameter of the “Class” function.

local Animal = Class(Object, function(this, \_name)

Object.init(this, “Animal”);

local name;

function this:GetName()

return name;

end

local function constructor(\_name)

name = \_name;

end

constructor(\_name);

end)

This is the Animal class which can only store the animal’s name (receives it from the constructor) and returns it from the getter function.

local Cat = Class(Animal, function(this, \_type, \_voice)

Animal.init(this, “Cat”);

local type;

local voice;

function this:Voice()

return “Cat of type ”..type..” says “..voice;

end

local function constructor(\_type, \_voice)

type = \_type;

voice = \_voice;

end

constructor(\_type, \_voice);

end)

This is the Cat class which inherits parent class Animal. Then we can create an instance of Cat and call on it methods from Animal.

To override methods you simply must create new method with the same name as an overrided method’s name.

## 1.4 Namespaces

To prevent some system classes from accidental overriding by user’s classes in LongOS was implemented namespaces system. This means that system classes are contained in special hierarchical collection and can be accessed from it. This collection is “Classes” collection.

This means that to access “Button” class you should write:

Classes.Components.Button

You can also simplify this action to prevent yourself from writing so long declaration every time you use buttons. To do this just add this code above your window (or other) class declaration:

local Button = Classes.Components.Buttons;

You can find namespaces for each class in its description below.

# 2 Introduction to Application

Application in LongOS are much different from those in CraftOS. They are objects (as almost everything in the system) and the whole model of information processing consists of reaction on different events. The system catches events and your program just need to process them. But it is rarely needed because the standard events processing is already implemented and hidden from you.

## 2.1 Application

Applications is a complex object which contains sets of windows and threads. Also it calls event handlers on this sets according to special algorithms.

Application class declaration are located in “Application” namespace.

To create an application you simply must create a new application object from its constructor:

local appSample = Classes.Aplication.Application(“App sample”);

This is the easiest way to create an application. There are 3 constructors for Application class:

Application(string applicationName, boolean isUnique, boolean shutdownWhenNoWindows)

Application(string applicationName, boolean isUnique)

Application(string applicationName)

Any of this constructors can be used to create an application instance.

To start running the application you need to call the “Run” method. The only parameter for this method is an instance of Window class. But you can pass there “nil” value if shutdownWhenNoWindows was setted to true. About windows creation see [section 2.3](#_2.3_Window).

For more information about application see [section 3.1](#_3.1_Application_Class).

## 2.2 Thread

You can create a parallel thread in your application and specify function for it. This can be useful for applications where some long-time actions must be done. Threads are linked to your application and will be stopped when it is closed. So in you want to create background application which will be executing some runtime actions you need to specify shutdownWhenNoWindows option to “true” on application creation.

Thread class declaration are located in “Application” namespace.

To create a new thread you just need to create its object:

local newThread = Classes.Application.Thread(application, workingFunction);

And then you can start the thread by calling:

newThread:Start();

For full specification see [section 3.2](#_3.2_Thread_Class).

## 2.3 Window

Window is the main interface part of the LongOS. It contains components, reacts on user’s actions and shows information. It is able to create event handlers for different events in a window but for most purposes there is no need of this. You can handle most of the events by using components — special elements which reacts on user’s actions or displays some information (Button, Label, Edit, etc.).

Window class declaration are located in the “Application” namespace.

But you can’t just create a window object from Window class because it will be only an empty window. For customizing window’s components and behavior you need to inherit it.

The sample of custom window class:

local Window = Classes.Application.Window;

local MyWindow = Class(Window, function(this, \_application)

Window.init(\_application, “My window”, false);

this:SetTitle(“My window”);

this:SetWidth(15);

this:SetX(3);

end)

The new window will take only 1 parameter in the constructor and will be created with title “My window”, width 15 and x position 3. In this way you can create your own event processors or add components to the window (about components see [section 2.4](#_2.4_Components)).

To add event processor you just need to override some basic function of event processing. For example:

function this:ProcessCharEvent(\_char)

…

end

This function will receive char events, directed to the window and process them in the way you like. For more event processors see window class specification.

Also there are a lot of properties that can be setted on your window. See [section 3.3](#_3.3_Window_Class) for more information.

After the creation the window can be shown either as single non-modal or as modal (dialog) window. To do this you must call the “Show” (or “ShowModal”) method with the parent application as the only parameter. If you use window in “Run” function of your application you don’t need to call “Show” or “ShowModal” methods for it. They will be invoked on application starting.

## 2.4 Components

Components play significant role in windows creation. They can handle most of the events and are easy to use. There are a lot of components in the LongOS. The most basic are Button, Label and Edit.

All components classes are located in “Components” namespace.

To add component to your window you need to create it first with a constructor.

local buttonSample = Classes.Components.Button(“Sample”, nil, nil, 0, 0, “left-top”);

Then you need to add this component to the window from outside:

myWindow:AddComponent(buttonSample);

Or from the window constructor:

MyWindow = Class(Window, function(this, \_application)

Window.init(this, \_application, “My window”, false);

local buttonSample2;

local function constructor()

buttonSample2 = Button(“Sample 2”, nil, nil, 0, 0, “left-top”);

this:AddComponent(buttonSample2);

end

constructor();

end)

The button will be placed on the window and will automatically process clicks. But we also need to add some event handler on the button click. You can do this that way:

MyWindow2 = Class(Window, function(this, \_application)

Window.init(this, \_application, “My window”, false);

local buttonSample;

local function buttonSampleClick(\_sender, \_eventArgs)

-- Do something.

end

local function constructor()

buttonSample = Button(“Sample”, nil, nil, 0, 0, “left-top”);

buttonSample:AddOnClickEventHandler(buttonSampleClick);

this:AddComponent(buttonSample);

end

constructor();

end)

This code will create window class with a button “Sample”. And on clicking the button code of the “buttonSampleClick” function will be executed.

You can create other components in the same way. See [section 4.1](#_4.1_Components) for components specification.

## 2.5 Menus

You can also create menus for your window if you like. For now they are simple single menus with buttons. To create a menu you must create a menu object and add buttons to it. Then you need to add that menu to the window’s menus collection (using method “AddMenu(string name, Menu menu)”). To operate menu use window’s methods “OpenMenu(string name)”, “CloseMenu(string name)” and “OpenCLoseMenu(string name)”. For more information see [section 4.2](#_4.2_PopupMenu).

# 3 Application Classes Specification

This section contains full information about application classes. They include:

* [Application Class](#_3.1_Application_Class)
* [Thread Class](#_3.2_Thread_Class)
* [Window Class](#_3.3_Window_Class)

## 3.1 Application Class

Application is a basic class for user’s programs. It contains windows and threads and manages event processing for them.

Namespace:

Classes.Application.Application

Constructors:

public Application(string \_applicationName, bool \_isUnique, bool \_shutdownWhenNoWindows)

public Application(string \_applicationName, bool \_isUnique)

public Application(string \_applicationName)

“\_applicationName“ — the name of application, it will be displayed in TasksManager.

“\_isUnique” — specifies that if such application is already opened it will be selected as current and not opened another instance of it. The default value for the 3-rd constructor is “false”.

“\_shutdownWhenNoWindows” — specifies that application will be closed when no windows is left. The default value for the 2-nd and the 3-rd constructors is “true”.

Methods:

public string GetName() — gets the application name.

public bool GetIsUnique() — gets if the application is unique.

public bool GetEnabled() — gets if the application is enabled (is current active application).

public void SetEnabled(bool \_value) — sets the enabled property for the application. This method is used by system classes and must not be called in programs.

public string GetId() — gets the application’s id.

public void Initialize(string \_id) — initializes the application. This method is used by system classes and must not be called in programs.

public void AddWindow(Window \_window) — adds a new window to the application. This method is called in “window:Show()” and “window:ShowModal()” so you don’t need to call them manually.

public void DeleteWindow(string \_windowId) — deletes window from the application. This method is called in “window:Close()”so you don’t need to call them manually.

public void AddThread(Thread \_thread) — adds thread to the application. This method is called in “thread:Start()” so you don’t need to call them manually.

public void RemoveThread(string \_id) — removes thread from the application. This method is called in “thread:Stop()” so you don’t need to call them manually.

public void Clear() — clear all windows and threads in the application.

public bool Contains(int x, int y) — returns if any window of the application contains specified point.

public int GetWindowsCount() — returns windows count.

public void Run(Window \_window) — starts the application with the specified window as the starting window. “\_window” can be nil if “\_shutdownWhenNoWindows” is setted to “false”.

public void Draw(VideoBuffer \_videoBuffer) — draws application to the video buffer.

public void Update() — updates the application’s state.

public void Close() — closes the application.

public void ProcessKeyEvent(int \_key) — executes **key** event processing in all windows and threads.

public void ProcessCharEvent(char \_symbol) — executes **char** event processing in all windows and threads.

public void ProcessRednetEvent(int \_id, string \_message, float \_distance, string \_side, int \_channel) — executes **rednet** event processing in all windows and threads.

public void ProcessLeftClickEvent(int \_cursorX, int \_cursorY) — executes **left click** event processing in all windows and threads.

public void ProcessRightClickEvent(int \_cursorX, int \_cursorY) — executes **right click** event processing in all windows and threads.

public void ProcessDoubleClickEvent(int \_cursorX, int \_cursorY) — executes **double click** event processing in all windows and threads.

public void ProcessLeftMouseDragEvent(int \_newCursorX, int \_newCursorY) — executes **left mouse drag** event processing in all windows and threads.

public void ProcessRightMouseDragEvent(int \_newCursorX, int \_newCursorY) — executes **right mouse drag** event processing in all windows and threads.

public void ResetDragging() — resets the dragging state of all windows. This method is used by system classes and must not be called in programs.

public void ProcessTimerEvent(int \_timerId) — executes **timer** event processing in all windows and threads.

public void ProcessRedstoneEvent() — executes **redstone** event processing in all windows and threads.

public void ProcessMouseScrollEvent(int \_direction, int \_cursorX, int \_cursorY) — executes **mouse scroll** event processing in all windows and threads.

## 3.2 Thread Class

Thread class can be used to create multithreaded applications which maintain big data arrays processing or other “slow” stuff. All threads are created stopped. You need to start the thread to begin the work.

Namespace:

Classes.Application.Thread

Constructor:

public Thread(application \_application, function \_operation)

Methods:

public string GetId() — returns the id of the thread.

public void Initialize(string \_id) — initializes the thread. This method is used by system classes and must not be called in programs.

public void Start() — stars the thread attaching it to the application.

public void Stop() — forcibly stops the thread detaching it from the application. The thread will be stopped automatically upon exiting the thread function.

## 3.3 Window Class

Window class is the main interface part of the LongOS. It contains many methods which can be used in your programs to create various things.

Namespace:

Classes.Application.Window

Constructor:

public Window(Application \_application, string \_name, bool \_isUnique)

“\_application” — the parent application.

“\_name” — name of the window. Don’t mix up this with window’s title.

“\_isUnique” — currently not active part of code. Set this property to “false” in your windows.

Methods:

public Application GetApplication() — gets the parent application of the window.

public string GetName() — gets the name of the window.

public bool GetIsUnique() — gets if the window is unique.

public string GetId() — gets the window’s id.

public void SetId(string \_value) — sets the window’s id. This method is used by system classes and must not be called in programs.

public string GetTitle() — gets the window’s title.

public void SetTitle(string \_value) — sets the window’s title.

public int GetBackgroundColor() — gets the window’s background color.

public void SetBackgroundColor(int \_value) — sets the window’s background color.

public bool GetIsModal() — gets if the window is modal.

public bool GetAllowMove() — gets if the window is allowed to be moved.

public void SetAllowMove(bool \_value) — sets if the window is allowed to be moved.

public bool GetAllowResize() — gets if window is allowed to be resized.

public void SetAllowResize(bool \_value) — sets if the window is allowed to be resized.

public int GetMinimalWidth() — gets the window’s minimal width.

public void SetMinimalWidth(int \_value) — sets the window’s minimal width.

public int GetMinimalHeight() — gets the window’s minimal height.

public void SetMinimalHeight(int \_value) — sets the window’s minimal height.

public bool GetAllowMaximize() — gets if window is allowed to be maximized.

public void SetAllowMaximize(bool \_value) — sets if window is allowed to be maximized.

public int GetX() — gets the window’s X position.

public void SetX(int \_value) — sets the window’s X position.

public int GetY() — gets the window’s Y position.

public void SetY(int \_value) — sets the window’s Y position.

public int GetWidth() — gets the window’s width.

public void SetWidth(int \_value) — sets the window’s width.

public int GetHeight() — gets the window’s height.

public void SetHeight(int \_value) — sets the window’s height.

public bool GetVisible() — gets if the window is visible.

public void SetVisible(bool \_value) — sets if the window is vidible.

public bool GetMaximized() — gets if the window is maximized.

public void SetMaximized(bool \_value) — sets if the window is maximized.

public bool GetEnabled() — gets if the window is enabled.

public void SetEnabled(bool \_value) — sets if the window is enabled. This method is used by system classes and must not be called in programs.

public void AddOnCloseEventHandler(function \_value) — adds a new event handler on window close event.

public void AddOnShowEventHandler(function \_value) — adds a new event handler on window show event.

public void AddOnMoveEventHandler(function \_value) — add a new event handler on window move event.

public void AddOnResizeEventHandler(function \_value) — add new event handler on window resize event.

public void AddOnMaximizeEventHandler(function \_value) — add new event handler on window maximize event.

public void AddOnMinimizeEventHandler(function \_value) — add new event handler on window minimize event.

public void Close() — closes the window.

public void Show() — shows the window in normal state. This means that the window which called this will still be available and can be made active by clicking on it.

public void ShowModal() — shows the window in modal state. This means that the window becomes an active window and doesn’t allow switching to another window (except for new ones created by itself) until closed.

public void Maximize() — maximizes window if its allowed.

public void Minimize() — minimizes window if it was maximized.

public bool Contains(int \_x, int \_y) — returns if the window or it’s menus contains specified point.

public void AddComponent(Copmonent \_component) — add a new component to the window.

public void OpenCloseMenu(string \_menuName) — open or close menu with specified name. Opened menu will be closed and closed menu will be opened.

public PopupMenu GetMenu(string \_menuName) — gets the menu by its name.

public void AddMenu(string \_menuName, PopupMenu menu) — adds a new menu and links it to its name.

public void OpenMenu(string \_menuName) — opens a menu selected by the name.

public void CloseAllMenues() — closes all menus.

public abstract void Draw(Canvas \_canvas) — drawing method that can be overridden by child windows. Allows to draw inside the window.

public abstract void Update() — updating method that can be overridden by child windows.

public abstract void ProcessLeftClickEvent(int \_cursorX, int \_cursorY) — left click event processing function that can be overridden by child windows.

public abstract void ProcessRightClickEvent(int \_cursorX, int \_cursorY) — right click event processing function that can be overridden by child windows.

public abstract void ProcessDoubleClickEvent(int \_cursorX, int \_cursorY) — double click event processing function that can be overridden by child windows.

public abstract void ProcessLeftMouseDragEvent(int \_newCursorX, int \_newCursorY) — left mouse drag event processing function that can be overridden by child windows.

public abstract void ProcessRightMouseDragEvent(int \_newCursorX, int \_newCursorY) — right mouse drag event processing function that can be overridden by child windows.

public abstract void ProcessMouseScrollEvent(int \_direction, int \_cursorX, int \_cursorY) — mouse scroll event processing function that can be overridden by child windows.

public abstract void ProcessKeyEvent(int \_key) — key event processing function that can be overridden by child windows.

public abstract void ProcessCharEvent(char \_symbol) — char event processing function that can be overridden by child windows.

public abstract void ProcessRednetEvent(int \_id, string \_message, float \_distance, string \_side, int \_channel) — rednet event processing function that can be overridden by child windows.

public abstract void ProcessTimerEvent(int \_timerId) — timer event processing function that can be overridden by child windows.

# 4 Components Classes Specification

This section contains information about the components classes.

Popup menu is also included in the components list.

All components are located in “Components” namespace.

## 4.1 Components

The LongOS has a lot of build-in components which can be used in your applications. It is also possible to create your own components by inheriting the base “Component” class. The list of components is:

* [Component](#_4.1.1_Component)
* [Label Component](#_4.1.2_Label_Component)
* [Button Component](#_4.1.3_Button_Component)
* [Edit Component](#_4.1.4_Edit_Component)
* [VerticalScrollBar Component](#_4.1.5_VerticalScrollBar_Component)
* [HorizontalScrollBar Control](#_4.1.6_HorizontalScrollBar_Component)
* [CheckBox Component](#_4.1.7_CheckBox_Component)
* [ListBox component](#_4.1.8_ListBox_Component)

### 4.1.1 Component

Component is the basic abstract class of all components. It contains no drawing or events processing logic and just handles some basic actions.

Namespace:

Classes.Components.Component

Constructor:

public Component(int \_dX, int \_dY, string \_anchorType)

“\_dX” — the offset on X coordinate for the label from the selected in “\_anchorType” window border.

“\_dY” — the offset on Y coordinate for the label from the selected in “\_anchorType” window border.

“\_anchorType” — window borders to which the component will be attached to. Can be: “left-top”, “right-top”, “left-bottom” and “right-bottom”.

Methods:

public int GetdX() — gets the dX value.

public void SetdX(int \_value) — sets the dX value.

public int GetdY() — gets the dY value.

public void SetdY(int \_value) — sets the dY value.

public int GetWidth() — gets the width of the component.

public int GetHeight() — gets the height of the component.

public int GetX() — gets the X coordinate of the component on the screen.

public int GetY() — gets the Y coordinate of the component on the screen.

public bool GetVisible() — gets if the component is visible.

public void SetVisible(bool \_value) — sets the visibility of the component.

public bool GetEnabled() — gets if the component is enabled.

public void SetEnabled() — sets the enabled flag of the component.

public bool Contains(int \_x, int \_y) — gets if the component contains specified point.

public bool DrawBase(Canvas \_videoBuffer, int \_x, int \_y) — draws the component on the video buffer.

public bool ProcessLeftClickEventBase(int \_cursorX, int \_cursorY) — processes left click event. This method is used by system classes and must not be called in programs.

public abstract bool ProcessLeftClickEvent(int \_cursorX, int \_cursorY) — override this method in child classes to process left click events.

public bool ProcessKeyEventBase(int \_key) — processes key event. This method is used by system classes and must not be called in programs.

public abstract bool ProcessKeyEvent(int \_key) — override this method in child classes to process key events.

public bool ProcessCharEventBase(string \_char) — processes char event. This method is used by system classes and must not be called in programs.

public abstract bool ProcessCharEvent(string \_char) — override this method in child classes to process char events.

public bool ProcessDoubleClickEventBase(int \_cursorX, int \_cursorY) — processes double click event. This method is used by system classes and must not be called in programs.

public abstract bool ProcessDoubleClickEvent(int \_cursorX, int \_cursorY) — override this method in child classes to process double click events.

public bool ProcessMouseScrollEventBase(int \_direction, int \_cursorX, int \_cursorY) — processes mouse scroll event. This method is used by system classes and must not be called in programs.

public abstract bool ProcessMouseScrollEvent(int \_direction, int \_cursorX, int \_cursorY) — override this method in child classes to process mouse scroll events.

### 4.1.2 Label Component

Labels are simple text displaying components.

Namespace:

Classes.Components.Label

Constructor:

public Label(string \_text, int \_backgroundColor, int \_textColor, int \_dX, int \_dY, string \_anchorType)

“\_text” — text of the label.

“\_backgroundColor” — background color for the label’s text. Can be “nil” and will use standard color schema in this way.

“\_textColor” — text color for the label’s text. Can be “nil” and will use standard color schema in this way.

Methods:

public string GetAnchor() — gets the anchor value.

public void SetAnchor(string \_value) — sets the anchor value.

public string GetText() — gets the text of the label.

public void SetText(string \_value) — sets the text of the label.

public int GetBackgroundColor() — gets the background color of the label.

public void SetBackgroundColor(int \_value) — sets the background color of the label.

public int GetTextColor() — gets the text color of the label.

public void SetTextColor(int \_value) — sets the text color of the label.

### 4.1.3 Button Component

Button is inherited from the Label so it contains the same methods.

Namespace:

Classes.Component.Button

Constructor:

public Button(string \_text, int \_backgroundColor, int \_textColor, int \_dX, int \_dY, string \_anchorType)

Methods:

public void AddOnClickEventHandler(function \_value) — add new event handler on click event.

### 4.1.4 Edit Component

Edit component allows user to enter some text information.

Namespace:

Classes.Components.Edit

Constructor:

public Edit(int \_width, int \_backgroundColor, int \_textColor, int \_dX, int \_dY, string \_anchorType)

“\_width” — the width of the edit’s line.

Methods:

public string GetText() — gets the edit’s text.

public void SetText(string \_value) — sets the edit’s text.

public void SetWidth(int \_value) — sets the edit’s width.

public int GetBackgroundColor()

public void SetBackgroundColor(int \_value)

public int GetTextColor()

public void SetTextColor(int \_value)

public bool GetFocus() — gets if the edit is in focus. You can only type in edit if it is in focus. Focus is setted by simple clicking the edit with left mouse button.

public void SetFocus(bool \_value) — set the focus value of the edit.

public void AddOnFocusEventHadler(function \_value)

public void AddOnTextChangedEventHandler(function \_value)

public void Clear() — clears the edit from text.

### 4.1.5 VerticalScrollBar Component

Vertical scroll bar is designed to allow scrolling in programs.

Namespace:

Classes.Components.VerticalScrollBar

Constructor:

public VerticalScrollBar(int \_minValue, int \_maxValue, int \_height, int \_barColor, int \_rollerColor, int \_dX, int \_dY, string \_anchorType)

“\_minValue” — minimal value of the scroll bar.

“\_maxValue” — maximal value of the scroll bar.

“\_height” — height of the scrollbar.

“\_barColor” — the color of the bar.

“\_rollerColor” — the color of the roller.

Methods:

public int GetValue() — gets the value of the scroll bar.

public void SetValue(int \_value) — sets the value of the scroll bar.

public void SetMaxValue(int \_value) — sets the maximal value of the scroll bar.

public void SetMinValue(int \_value) — sets the minimal value of the scroll bar.

public void SetHeight(int \_value) — sets the height of the scroll bar.

public void AddOnValueChangedEventHandler(function \_value)

public void ScrollUp() — scroll the scroll bar up.

public void ScrollDown() — scroll the scroll bar down.

### 4.1.6 HorizontalScrollBar Component

Horizontal scroll bar is the horizontal copy of the vertical scroll bar and has the same functionality.

Namespace:

Classes.Components.HorizontalScrollBar

Constructor:

public HorizontalScrollBar(int \_minValue, int \_maxValue, int \_width, int \_barColor, int \_rollerColor, int \_dX, int \_dY, string \_anchorType)

Methods:

public int GetValue()

public void SetValue(int \_value)

public void SetMaxValue(int \_value)

public void SetMinValue(int \_value)

public void SetWidth(int \_value)

public void AddOnValueChangedEventHandler(function \_value)

public void ScrollLeft()

public void ScrollRight()

### 4.1.7 CheckBox Component

CheckBox component allows user to set boolean values through the controls.

Namespace:

Classes.Components.CheckBox

Constructor:

public CheckBox(int \_backgroundColor, int \_textColor, int \_dX, int \_dY, string \_anchorType)

Methods:

public string GetCheckedSymbol() — gets the symbol that will be used to marked checked CheckBox.

public void SetCheckedSymbol(string \_value) — sets the symbol that will be used to marked checked CheckBox.

public int GetBackgroundColor()

public void SetBackgroundColor(int \_value)

public int GetTextColor()

public void SetTextColor(int \_value)

public bool GetChecked() — gets if the checkbox is checked.

public void SetChecked(bool \_value) — sets the checked property of the checkbox.

public void AddOnCheckedChangedEventHandler(function \_value)

### 4.1.8 ListBox Component

ListBox component allows you to represent some data in the form of scrollable selectable list.

Namespace:

Classes.Components.ListBox

Constructor:

public void ListBox(int \_width, int \_height, int \_backgroundColor, int \_textColor, int \_dX, int \_dY, string \_anchorType)

Methods:

public int GetCount() — gets the count of elements in the listboxe’s collection.

public string GetSelectedItem() — gets currently selected item.

public int GetSelectedIndex() — gets the index of currently selected item.

public void SetSelectedIndex(int \_index) — selects the item with specified index.

public string GetItem(int \_index) — gets the item with specified index.

public void SetWidth(int \_value)

public void SetHeight(int \_value)

public int GetBackgroundColor()

public void SetBackgroundColor(int \_value)

public int GetSelectedBackgroundColor() — gets the background color of the selected item.

public void SetSelectedBackgroundColor(int \_value) — sets the background color of the selected item.

public int GetTextColor()

public void SetTextColor(int \_value)

public int GetSelectedTextColor() — gets the text color of the selected item.

public void SetSelectedTextColor(int \_value) — sets the text color of the selected item.

public void AddOnSelectedIndexChangedEventHandler(function \_value)

public bool Clear() — clears the list of items.

public void AddItem(string \_item) — adds new item to the list box.

public void RemoveItemAt(int \_index) — removes item at the specified index.

## 4.2 PopupMenu

Popup menus allows to organize main menu or context menu in your program. There is still no possibility to create nested menus but this feature will come shortly.

Namespace:

Classes.Components.PopupMenu

Constructor:

public PopupMenu(int x, int y, int width, int height, int backgroundColor, bool \_allowAutoHeight)

“x” — the X coordinate of the menu in screen coordinates.

“y” — the Y coordinate of the menu in screen coordinates.

“width” — width of the menu.

“height” — height of the menu.

“backgroundColor” — the background color of the menu. Can be “nil”.

“\_allowAutoHeight” — enable this option if you want your menu to automatically adjust its width to the maximal width of its components.

Methods:

public void AddComponent(Component component) — adds new component to the menu.

public void Clear() — clears the menu.

public bool GetIsOpened() — gets if the menu is opened.

public bool Contains(int x, int y) — gets if the menu contains specified point.

public void Open() — opens the menu.

public void Close() — closes the menu.

public void OpenClose() — opens or closes the menu depending on its current state.

To properly use menus just add them to the window by calling “window:AddMenu(menuName, menuInstance)”. Then call “window:OpenCloseMenu(menuName)” to open or close your menu. This will save you from conflicts with multiple menus showing in one time.

# 5 System Classes Specification

System classes are not designed to be created by user’s programs but their objects can be accessed through the system class. Because of that this section of the guide will describe the accessible system classes and their interfaces.

## 5.1 LongOS Class

The LongOS class is the main system class which all system’s methods and objects. There is only one instance of this class in LongOS — “System” object. From this object you can gain access to another system objects or perform some operations directly.

Methods:

public void LogRuntimeError(string errorText) — logs runtime error to the runtime log (/LongOS/Logs/runtime.log file). You can also use this command do debug values in your applications.

public void SetCurrentApplication(string id) — makes some application (specified by the id) active.

public string GetCurrentVersion() — gets the current version of the OS.

public void Shutdown() — shuts down the computer.

public void Reboot() — reboots the computer.

public void LogOff() — shuts down the system returning you back to the CraftOS interface.

public void ShowMessage(string title, string text, int textColor) — shows a system message with selected title, text and text color. Text color can be “nil”.

public void ShowError(string message) — shows error message with specified text.

public string GetCurrentTime() — gets current system time in the string form.

public string GenerateId() — generates a new unique id.

public table GetApplicationsList() — returns the applications list. The result is just a table with some metadata not the real applications objects.

public int GetApplicationsCount() — returns current applications count.

public void RunFile(string fileName) — runs a new file with error processing. This function is better than standard “shell.run”.

public bool Try(function func) — tries to execute some function without parameters. If an error occurs the error message will be logged and shown. This function should be used in program’s “exec” files.

## 5.2 Image Class

Image class helps to easy open, save and edit image files. It only works with “.image” files created by itself (from BiriPaint for example).

Namespace:

Classes.System.Graphics.Image

Constructors:

public Image(string fileName)

public Image(int width, int height)

public Image(Image image)

Methods:

public int GetWidth()

public int GetHeight()

public void SetWidth(int \_width)

public void SetHeight(int \_height)

public int GetPixel(int x, int y)

public void SetPixel(int x, int y, int color)

public void SetSize(int \_width, int \_height)

public void DrawLine(int \_color, int \_x1, int \_y1, int \_x2, int \_y2)

public void DrawRect(int \_color, int \_x, int \_y, int \_width, int \_height)

public void DrawEllipse(int \_color, int \_x, int \_y, int \_width, int \_height)

public void SaveToFile(string \_fileName)

public void LoadFromFile(string \_fileName)

# 6 Guidelines

There are several guidelines to help you keep your code beautiful and easy to understand. You also need to use them if you would like your program to be included in the LongOS.

## 6.1 Classes Parts

It is useful to organize methods and fields inside your class in a special way to make you code easier to read. It’s suggested to use such order:

* Fields
* Getters and setters for the fields
* Methods
* Initializer (constructor)

This order will also allow you to use class methods in the initialization area.

## 6.2 Separation

It is a good practice to separate the parts of your class by some comments-separators. You can use the Window class as an example. This will visually separate your code and make it more beautiful.

## 6.3 Comments

It’s a good practice to use comments inside your code. Especially in some methods with difficult for understanding. You can also add comments before methods but this is not necessary if your methods are named correctly.

## 6.4 Naming Rules

Private variables and methods should be named camel style with first letter in lover register. Public variables and methods should be named in camel style with first letter in upper register. Parameters in functions must be prefixed with the “\_” character. Constants must be named in upper register with “\_” character between the words.

## 6.5 Access Modifiers

All variables in your classes must be private (local). To access them you can create public methods to get value or set value to the variable.

## 6.6 Errors Processing

It is better to process errors in your program then allowing it to fall into error message generated by the system. For errors processing use function “pcall”.

# 7 Conclusion

This guide was created to answer some common questions of the advanced users. If you still have some questions you can ask them on the forum. If you found some bugs or have any good formulated suggestions you can add them to the issues on our [GiyHub repository](https://github.com/Gvin/LongOS/issues).