CS699 Project: Linux Tutorial Platform

Generated by Doxygen 1.8.15

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 System Struct Reference

Structure that contains all the state of the engine.

Classes

- struct System_Audio

  Structure that contains the state of the audio subsystem.
- struct System_Controls

  Structure that contains the state of the control subsystem.
- struct System_Time

  Structure that contains the state of the timing subsystem.
- struct System_Window

  Structure that contains the state of the windowing subsystem.

Public Attributes

- struct System::System_Window **window**
- struct System::System_Audio **audio**
- struct System::System_Time **time**
- struct System::System_Controls **controls**

3.1.1 Detailed Description

Structure that contains all the state of the engine.

This structure is used to maintain and communicate the state of the engine to its various subsystems. This struct, as a whole, can be thought to incorporate the global state of the program at any given moment.

The documentation for this struct was generated from the following file:

- src/main.c

3.2 System::System_Audio Struct Reference

Structure that contains the state of the audio subsystem.

Public Attributes

- SDL_AudioDeviceID audio_device
- SDL_AudioSpec audio_spec
- S16 ∗ audio_buffer
- U32 bytes_per_sample
- U32 target_queue_bytes
- U32 volume

3.2.1 Detailed Description

Structure that contains the state of the audio subsystem.

This contains all the handles for the audio device, etc. as well as the parameters of audio such as sampling rate that are used for the playing of audio.

3.2.2 Member Data Documentation

3.2.2.1 audio_buffer

```
S16* System::System_Audio::audio_buffer
```

Memory buffer to store sound sample data

3.2.2.2 audio_device

```
SDL_AudioDeviceID System::System_Audio::audio_device
```

Handle to audio device

3.2.2.3 audio_spec

```
SDL_AudioSpec System::System_Audio::audio_spec
```

Specifications of the audio device

3.2.2.4 bytes_per_sample

```
U32 System::System_Audio::bytes_per_sample
```

Size of each sample in bytes

3.2.2.5 target_queue_bytes

```
U32 System::System_Audio::target_queue_bytes
```

Latency of audio stream in bytes

3.2.2.6 volume

```
U32 System::System_Audio::volume
```

Sound's loudness level

The documentation for this struct was generated from the following file:

- src/main.c

3.3 System::System_Controls Struct Reference

Structure that contains the state of the control subsystem.

Public Attributes

- U8 keyboard [SDL_NUM_SCANCODES]
- struct {
  S32 x
  S32 y
  B32 left
  B32 right
  B32 middle
  } mouse

3.3.1 Detailed Description

Structure that contains the state of the control subsystem.

This contains all the data regarding the input devices through which the users interact with the program.

3.3.2 Member Data Documentation

3.3.2.1 keyboard

```
U8 System::System_Controls::keyboard[SDL_NUM_SCANCODES]
```

State of each ley of keyboard

3.3.2.2 left

```
B32 System::System_Controls::left
```

State of left mouse button

3.3.2.3 middle

```
B32 System::System_Controls::middle
```

State of middle mouse button

3.3.2.4 mouse

```
struct { ...  } System::System_Controls::mouse
```

State of mouse input device

3.3.2.5 right

```
B32 System::System_Controls::right
```

State of right mouse button

3.3.2.6 x

```
S32 System::System_Controls::x
```

Horizontal osition of the mouse cursor

3.3.2.7 y

```
S32 System::System_Controls::y
```

Vertical osition of the mouse cursor

The documentation for this struct was generated from the following file:

- src/main.c

3.4 System::System_Time Struct Reference

Structure that contains the state of the timing subsystem.

Public Attributes

- U64 last_counter

3.4.1 Detailed Description

Structure that contains the state of the timing subsystem.

This contains the last computed value of time, to be used to calculate the elapse of time.

3.4.2 Member Data Documentation

3.4.2.1 last_counter

```
U64 System::System_Time::last_counter
```

Last computed value of time

The documentation for this struct was generated from the following file:

- src/main.c

3.5 System::System_Window Struct Reference

Structure that contains the state of the windowing subsystem.

Public Attributes

- SDL_Window ∗ window
- SDL_GLContext gl_context
- GLuint quad_vao
- GLuint luabuffer
- GLuint luabuffer_texture
- GLuint xbloombuffer
- GLuint xbloombuffer_texture
- GLuint xbloom_shader
- GLuint crt_shader
- S32 width
- S32 height

3.5.1 Detailed Description

Structure that contains the state of the windowing subsystem.

This contains all the handles for the window, framebuffers, etc. that are used for the display of the game.

3.5.2 Member Data Documentation

3.5.2.1 crt_shader

```
GLuint System::System_Window::crt_shader
```

Shader used to finally render to the screen

3.5.2.2 gl_context

```
SDL_GLContext System::System_Window::gl_context
```

Handle to OpenGL context

3.5.2.3 height

```
S32 System::System_Window::height
```

Height of the window

3.5.2.4 luabuffer

```
GLuint System::System_Window::luabuffer
```

Framebuffer into which the Lua code renders

3.5.2.5 luabuffer_texture

```
GLuint System::System_Window::luabuffer_texture
```

Texture associated with the luabuffer

3.5.2.6 quad_vao

```
GLuint System::System_Window::quad_vao
```

Handle to the vertex array object associated with a quad

3.5.2.7 width

```
S32 System::System_Window::width
```

Width of the window

3.5.2.8 window

```
SDL_Window* System::System_Window::window
```

Handle to the window

3.5.2.9 xbloom_shader

```
GLuint System::System_Window::xbloom_shader
```

Shader associated with the xbloombuffer

3.5.2.10 xbloombuffer

```
GLuint System::System_Window::xbloombuffer
```

Intermediate framebuffer with horizontal bloom effect

3.5.2.11 xbloombuffer_texture

```
GLuint System::System_Window::xbloombuffer_texture
```

Texture associated with the xbloombuffer

The documentation for this struct was generated from the following file:

- src/main.c

Chapter 4

File Documentation

4.1 src/assets.c File Reference

Functions for assets loading.

Functions

- internal_function B32 assetLoadShader (char ∗vertex_path, char ∗fragment_path, GLuint ∗program)

  This function load the GLSL shaders assets for visual effetcs.
- internal_function B32 assetLoadScript (lua_State ∗game, char ∗script_path)

  This function load the Lua script for gameplay code.
- internal_function B32 assetLoadTrueTypeFont (Char ∗font_path, U32 font_size, Char ∗vert_path, Char ∗frag_path, U32 bitmap_width, U32 bitmap_height, U32 char_first, U32 char_num, GLuint vao, GLuint vbo, stbtt_bakedchar ∗∗baked_char, GLuint ∗program, GLuint ∗texture)

  This function load a TTF font and bakes it into a bitmap.

4.1.1 Detailed Description

Functions for assets loading.

This file contains the functions used for loading various files containing data used in the running of this game. These files are often called assets in the the parlance of game development.

Author

Team Octal

4.1.2 Function Documentation

4.1.2.1 assetLoadScript()

```
internal_function B32 assetLoadScript (
              lua_State * game,
              char * script_path )
```

This function load the Lua script for gameplay code.

It reads the Lua script file at `script_path` and compiles it into a Lua context `game`. If any errors occur during compilation, it also takes care of them.

Parameters

game	The Lua context in which the scripts will be run
script_path	Path of Lua script source file

Returns

Execution status

4.1.2.2 assetLoadShader()

```
internal_function B32 assetLoadShader (
            char * vertex_path,
            char * fragment_path,
            GLuint * program )
```

This function load the GLSL shaders assets for visual effetcs.

It reads the vertex shader stored at `vertex_path` and fragment shader stored at `fragement_path` and compiles the into a single shader program. Before exiting it frees up the vertex and fragment source.

Parameters

vertex_path	Path of vertex shader
fragment_path	Path of fragment shader
program	Used to return the GLSL program handle

Returns

success/failure

4.1.2.3 assetLoadTrueTypeFont()

```
internal_function B32 assetLoadTrueTypeFont (
            Char * font_path,
            U32 font_size,
            Char * vert_path,
            Char * frag_path,
            U32 bitmap_width,
            U32 bitmap_height,
            U32 char_first,
            U32 char_num,
            GLuint vao,
            GLuint vbo,
            stbtt_bakedchar ** baked_char,
            GLuint * program,
            GLuint * texture )
```

This function load a TTF font and bakes it into a bitmap.

It load the font file and using stb_truetype library, bakes the vector font into a bitmap that can be rendered using OpenGL's native capabilities to render textured quads.

Parameters

font_path	Path of the font file
font_size	Point size of rendered font
vert_path	Path of vertex shader used for text rendering
frag_path	Path of fragment shader used for text rendering
bitmap_width	Width opf baked bitmap
bitmap_height	Heifght of baked bitmap
char_first	First renderable character
char_num	Total number of renderable characters
vao	Vertex Array Object
vbo	Vertex Buffer Object
baked_char	Baked bitmap
program	Returns handle for compiled shader
texture	Returns handle for texture stored GPU side

Returns

Execution status

4.2 src/assets_script.c File Reference

Lua functions for asset loading.

Functions

- internal_function Sint scriptAssetLoadScript (lua_State ∗l)

  Lua injected function which calls assetLoadScript.
- internal_function int scriptAssetLoadTrueTypeFont (lua_State ∗l)

  Lua injected function which calls assetLoadTrueTypeFont.

4.2.1 Detailed Description

Lua functions for asset loading.

These functions are called from Lua and are used to hook into the asset loader that is implemented in the engine.

Author

Team Octal

4.2.2 Function Documentation

4.2.2.1 scriptAssetLoadScript()

```
internal_function Sint scriptAssetLoadScript (
            lua_State * l )
```

Lua injected function which calls assetLoadScript.

This function is called from Lua and is used to call into assetLoadScript using proper parameters.

Parameters

l	Lua context

Returns

Execution status

4.2.2.2 scriptAssetLoadTrueTypeFont()

```
internal_function int scriptAssetLoadTrueTypeFont (
            lua_State * l )
```

Lua injected function which calls assetLoadTrueTypeFont.

This function is called from Lua and is used to call into assetLoadTrueTypeFont using proper parameters.

Parameters

l	Lua context

Returns

Execution status

4.3 src/debug.c File Reference

Functions for debugging.

```
#include <execinfo.h>
#include <stdio.h>
#include <stdlib.h>
```

Macros

- #define **MAX_STACK_FRAMES** 64

Functions

- internal_function int **debug_AddressToLine** (void ∗addr)
- internal_function void debugPrintCallStackTrace ()

  Function to print stack trace for debugging.

4.3.1 Detailed Description

Functions for debugging.

These functions are used in debugging any runtime errors that maybe otherwise be hard to fix. Usually, these functions are used in tandem with logging system.

Author

Team Octal

4.3.2 Function Documentation

4.3.2.1 debugPrintCallStackTrace()

```
internal_function void debugPrintCallStackTrace ( )
```

Function to print stack trace for debugging.

This function print the stack trace for the current execution state of program. It skip the first couple of stack frames and also skip the last frame as it usually contains junk.

4.4 src/event.c File Reference

Functions for event handling.

Functions

- internal_function void eventKeyboard (lua_State ∗l, char ∗key, char ∗state)

  Function to send the key being pressed to Lua.
- internal_function void eventText (lua_State ∗l, const char ∗const text)

  Function to send the text to Lua after it has been typed.
- internal_function void eventTextControl (lua_State ∗l, const char ∗const control)

  Function to swend the control characters beeing pressed to Lua.

4.4.1 Detailed Description

Functions for event handling.

These functions are used in handling operating system events generated due to user interactionss with the program. Since the actual event processing happens in Lua code, we just transfer these events to a Lua context.

Author

Team Octal

4.4.2 Function Documentation

4.4.2.1 eventKeyboard()

```
internal_function void eventKeyboard (
             lua_State * l,
             char * key,
             char * state )
```

Function to send the key being pressed to Lua.

This fucntion sends the "Key Pressed" events to Lua game code using neccessary lua function to perform corresponding action.

Parameters

l	Lua context
key	Name of pressed key
state	The state of key (up/down/held)

4.4.2.2 eventText()

```
internal_function void eventText (
             lua_State * l,
             const char *const text )
```

Function to send the text to Lua after it has been typed.

This function gets the text which was typed and call neccessary lua function to send it to the Lua context..

Parameters

l	Lua context
text	Typed text

4.4.2.3 eventTextControl()

```
internal_function void eventTextControl (
            lua_State * l,
            const char *const control )
```

Function to swend the control characters beeing pressed to Lua.

This function sends control characters (e.g. ENTER, BACKSPACE, etc.) to Lua which are used to perform special actions in game.

Parameters

l	Lua context
control	Name of special character

4.5 src/file.c File Reference

Functions for file read/write operations.

Functions

- internal_function Byte ∗ fileRead (char ∗file_path, Size ∗size)

  Reads a file at the given path.

4.5.1 Detailed Description

Functions for file read/write operations.

These functions are used in readig and writing files. These are the low-level functions on which other systems like asset loader rely.

Author

Team Octal

4.5.2 Function Documentation

4.5.2.1 fileRead()

```
internal_function Byte* fileRead (
            char * file_path,
            Size * size )
```

Reads a file at the given path.

This function is used for loading and reading files for the game which contain various pieces of data needed for the proper operations of the game.

Parameters

file_path	Path to the file
size	Returns the size of read file

Returns

Pointer to the file data

4.6 src/log.c File Reference

Functions for logging.

Enumerations

- enum Log_Level {
  LOG_LEVEL_VERBOSE, **LOG_LEVEL_DEBUG**, **LOG_LEVEL_INFO**, **LOG_LEVEL_WARN**,
  LOG_LEVEL_ERROR, **LOG_LEVEL_CRITICAL**, **LOG_LEVEL_COUNT** }

  Enumeration of all priority levels of logging.
- enum Log_Channel {
  LOG_CHANNEL_UNKNOWN = SDL_LOG_CATEGORY_APPLICATION, **LOG_CHANNEL_OPENGL**, **L**←↩
  OG_CHANNEL_ASSETS, **LOG_CHANNEL_LOG**,
  LOG_CHANNEL_FILE, **LOG_CHANNEL_SCRIPT**, **LOG_CHANNEL_TIME**, **LOG_CHANNEL_AUDIO**,
  LOG_CHANNEL_RENDER, **LOG_CHANNEL_INIT**, **LOG_CHANNEL_ARG**, **LOG_CHANNEL_LOOP**,
  LOG_CHANNEL_COUNT }

  Enumeration of all sources of logging messages.

Functions

- internal_function B32 logConsole (enum Log_Level level, enum Log_Channel channel, const char ∗text,...)

  Function to log console data/information.
- internal_function void logGLDebugCallback (U32 source, U32 type, U32 id, U32 severity, S32 length, const Char ∗message, const void ∗user_param)

  Function to log OpenGL diagnostics.

4.6.1 Detailed Description

Functions for logging.

These functions are used for logging any data that could be used for debugging, diagnostics or analysis in the future.

Author

Team Octal

4.6.2 Enumeration Type Documentation

4.6.2.1 Log_Channel

```
enum Log_Channel
```

Enumeration of all sources of logging messages.

This enum provides all the channels (or sources) from which a log diagnostic may arrive.

4.6.2.2 Log_Level

```
enum Log_Level
```

Enumeration of all priority levels of logging.

This enum provides various priority (or severity) levels that we can assign to any log diagnostic.

4.6.3 Function Documentation

4.6.3.1 logConsole()

```
internal_function B32 logConsole (
             enum Log_Level level,
             enum Log_Channel channel,
             const char * text,
              ... )
```

Function to log console data/information.

This function is used for logging console information and other errors and warning messages produced during the game's executions.

4.6.3.2 logGLDebugCallback()

```
internal_function void logGLDebugCallback (
             U32 source,
             U32 type,
             U32 id,
             U32 severity,
             S32 length,
             const Char * message,
             const void * user_param )
```

Function to log OpenGL diagnostics.

This function is used for logging 3D rendering debug data and other error and warning messages related to 3D OpenGL graphics.

Parameters

source	Information on who sent the diagnostic
type	Kind of diagnostic
id	Unique ID of the diagnostic
severity	Denotes how severe it is
length	Length of the diagnostic message
message	Diagnostic message
user_param	Any user parameters, unused

4.7 src/log_script.c File Reference

Lua functions for logging.

Functions

- internal_function int scriptLog (lua_State *l)

  Lua injected function which calls logConsole.

4.7.1 Detailed Description

Lua functions for logging.

These functions are called from Lua and are used to hook into the logger that is implemented in the engine.

Author

Team Octal

4.7.2 Function Documentation

4.7.2.1 scriptLog()

```
internal_function int scriptLog (
            lua_State * l )
```

Lua injected function which calls logConsole.

This function is called from Lua and is used to call into logConsole using proper parameters.

Parameters

l	Lua context

Returns

Execution status

4.8 src/main.c File Reference

Main engine source.

```
#include "nlib/nlib.h"
#include "nlib/linear_algebra.h"
#include "external/glad/glad.h"
#include "external/glad/glad.c"
#include "external/SDL2/SDL.h"
#include "stb/stb_truetype.h"
#include "external/lua/lua.h"
#include "external/lua/lauxlib.h"
#include "external/lua/lualib.h"
#include "debug.c"
#include "log.c"
#include "time.c"
#include "opengl.c"
#include "render.c"
#include "file.c"
#include "assets.c"
#include "event.c"
#include "log_script.c"
#include "assets_script.c"
#include "render_script.c"
```

Classes

- struct System

  Structure that contains all the state of the engine.
- struct System::System_Window

  Structure that contains the state of the windowing subsystem.
- struct System::System_Audio

  Structure that contains the state of the audio subsystem.
- struct System::System_Time

  Structure that contains the state of the timing subsystem.
- struct System::System_Controls

  Structure that contains the state of the control subsystem.

Macros

- #define **STB_TRUETYPE_IMPLEMENTATION**
- #define **STBTT_STATIC**
- #define **SCRIPT_FUNCTION_SYSTEM_UPVALUE**(FUNC)
- #define **SCRIPT_FUNCTION_GAME_UPVALUE**(FUNC)
- #define **SCRIPT_FUNCTION_SYSTEM_GAME_UPVALUE**(FUNC)
- #define **SCRIPT_FUNCTION_NO_UPVALUE**(FUNC)

Typedefs

- typedef struct System_Window **System_Window**
- typedef struct System_Audio **System_Audio**
- typedef struct System_Time **System_Time**
- typedef struct System_Controls **System_Controls**
- typedef struct System System

  Structure that contains all the state of the engine.

Functions

- Sint main (Sint argc, Char ∗argv[ ])

  Entry point of the program.

Variables

- global_variable char ∗ **global_program_name**
- global_variable B32 **global_game_is_running**

4.8.1 Detailed Description

Main engine source.

This file implements the core engine and uses all the various other subsystems in an orderly manner.

Author

Team Octal

4.8.2 Macro Definition Documentation

4.8.2.1 SCRIPT_FUNCTION_GAME_UPVALUE

```
#define SCRIPT_FUNCTION_GAME_UPVALUE(
                FUNC )
```

Value:

```
do {                \
                char *str = #FUNC;                               \
                lua_pushlightuserdata(game_code, game_code);     \
                lua_pushcclosure(game_code, FUNC, 1);            \
                lua_setfield(game_code, 1, &(str[6]));           \
            } while (0)
```

4.8.2.2 SCRIPT_FUNCTION_NO_UPVALUE

```
#define SCRIPT_FUNCTION_NO_UPVALUE(
                FUNC )
```

Value:

```
do {   \
        char *str = #FUNC;                      \
        lua_pushcclosure(game_code, FUNC, 0);   \
        lua_setfield(game_code, 1, &(str[6]));  \
    } while (0)
```

4.8.2.3 SCRIPT_FUNCTION_SYSTEM_GAME_UPVALUE

```
#define SCRIPT_FUNCTION_SYSTEM_GAME_UPVALUE(
                FUNC )
```

Value:

```
do {          \
                char *str = #FUNC;                              \
                lua_pushlightuserdata(game_code, &system);      \
                lua_pushlightuserdata(game_code, game_code);    \
                lua_pushcclosure(game_code, FUNC, 2);           \
                lua_setfield(game_code, 1, &(str[6]));          \
            } while (0)
```

4.8.2.4 SCRIPT_FUNCTION_SYSTEM_UPVALUE

```
#define SCRIPT_FUNCTION_SYSTEM_UPVALUE(
                FUNC )
```

Value:

```
do {               \
                char *str = #FUNC;                              \
                lua_pushlightuserdata(game_code, &system);      \
                lua_pushcclosure(game_code, FUNC, 1);           \
                lua_setfield(game_code, 1, &(str[6]));          \
            } while (0)
```

4.8.3 Typedef Documentation

4.8.3.1 System

```
typedef struct System System
```

Structure that contains all the state of the engine.

This structure is used to maintain and communicate the state of the engine to its various subsystems. This struct, as a whole, can be thought to incorporate the global state of the program at any given moment.

4.8.4 Function Documentation

4.8.4.1 main()

```
Sint main (
             Sint argc,
             Char * argv[] )
```

Entry point of the program.

This function is the entry point into the program and brings together all the parts of the engine to make a coherent whole. It implements program initialization, main loop and destruction.

Parameters

argc	Number of command line parameters
argv	Array of command line parameters

Returns

Return value of the program

4.9 src/opengl.c File Reference

Functions for OpenGL API.

Functions

- internal_function GLint openglShaderCreate (const char ∗const vert_src, const char ∗const frag_src)

  Function to compile OpenGL shaders.

4.9.1 Detailed Description

Functions for OpenGL API.

These functions are used to wrap around the OpenGL API in order to provide a higher level abstraction for often used operations.

Author

Team Octal

4.9.2 Function Documentation

4.9.2.1 openglShaderCreate()

```
internal_function GLint openglShaderCreate (
            const char *const vert_src,
            const char *const frag_src )
```

Function to compile OpenGL shaders.

This function takes the source of a vertex shader and a fragment shader, and compiles them into a the OpenGL program for graphics rendering.

Parameters

vert_src	Vertex shader source
frag_src	Fragment shader source

Returns

Handle to the compiled program

4.10 src/render.c File Reference

Functions for rendering.

Functions

- internal_function B32 renderText (GLuint vao, GLuint vbo, GLuint texture, GLuint program, char char_first, char char_num, U32 bitmap_width, U32 bitmap_height, stbtt_bakedchar *baked_char, const char *text, Vec3 screen_pos, Vec3 color, F32 scale_factor, F32 x_scaling, F32 *x_ret, F32 *y_min_ret, F32 *y_max_ret)

  Function to render text using OpenGL.

4.10.1 Detailed Description

Functions for rendering.

These functions are used for performing the rendering using the OpenGL API. In the current application, they are only used for 2D rendering of text, as well as applying various postprocessing effects using shaders.

Author

Team Octal

4.10.2 Function Documentation

4.10.2.1 renderText()

```
internal_function B32 renderText (
            GLuint vao,
            GLuint vbo,
            GLuint texture,
            GLuint program,
            char char_first,
            char char_num,
            U32 bitmap_width,
            U32 bitmap_height,
            stbtt_bakedchar * baked_char,
            const char * text,
            Vec3 screen_pos,
            Vec3 color,
            F32 scale_factor,
            F32 x_scaling,
            F32 * x_ret,
            F32 * y_min_ret,
            F32 * y_max_ret )
```

Function to render text using OpenGL.

This function is used to render text through OpenGL API. It uses shaders to render text in different font and color in the game.

Parameters

vao	Vertex Array Object
vbo	Vertex Buffer Object
baked_char	Baked bitmap font
program	Handle to shader program
texture	Handle to font texture
bitmap_width	Width of font bitmap
bitmap_height	Height of font bitmap
char_first	First renderable character
char_num	Total number of renderable characters
text	Text which needs to be rendered
screen_pos	Screen space position of rendered text
color	Color of rendered text
scale_factor	Scaling factor, to be applied on quads
x_scaling	Horizontal scaling constant used in font baking process
x_ret	Returns the width of rendered text in screen space
y_min_ret	Returns height of rendered text above baseline in screen space
y_max_ret	Returns depth of rendered text below baseline in screen space

Returns

Execution status

4.11 src/render_script.c File Reference

Lua functions for rendering.

Functions

- internal_function int scriptRenderGetTextDimensions (lua_State ∗l)

    Lua injected function which computes the size of text in screen space.
- internal_function int scriptRenderText (lua_State ∗l)

    Lua injected function which calls renderText.

4.11.1 Detailed Description

Lua functions for rendering.

These functions are called from Lua and are used to hook into the renderer that is implemented in the engine.

Author

Team Octal

4.11.2 Function Documentation

4.11.2.1 scriptRenderGetTextDimensions()

```
internal_function int scriptRenderGetTextDimensions (
            lua_State ∗ l )
```

Lua injected function which computes the size of text in screen space.

This function is called from Lua and is used to pre-compute the size some text would take in screen space if rendered as is.

Parameters

l	Lua context

Returns

Execution status

4.11.2.2 scriptRenderText()

```
internal_function int scriptRenderText (
            lua_State ∗ l )
```

Lua injected function which calls renderText.

This function is called from Lua and is used to call into renderText using proper parameters.

Parameters

l	Lua context

Returns

Execution status

4.12 src/time.c File Reference

Functions for timing.

Functions

- internal_function F64 timeMicrosecondsElapsed (U64 ∗last_counter)

  Function to compute elapsed time.

4.12.1 Detailed Description

Functions for timing.

These functions are used for various timing related operations, the output of which is then used in various synchronized activities such as animation. In the current program, the only place we make use of timing is during the blinking of cursor.

Author

Team Octal

4.12.2 Function Documentation

4.12.2.1 timeMicrosecondsElapsed()

```
internal_function F64 timeMicrosecondsElapsed (
            U64 * last_counter )
```

Function to compute elapsed time.

Given start time, this function count the total elapsed time from the start time.

Parameters

last_counter	From when to start counting

Returns

Time passed since `last_counter`

Index