Lua App Creation Tutorial

[SDK 2.0]

Year 2011
Version 1.40

Samsung Smart TV

Preface

Purpose of Document

This document is written to help Game content developers to use and enjoy game development using the Samsung's Software Development Kit (SDK). This provides an overview of the Samsung's Lua Script based Game framework and illustrative examples to game content developers to use SDK effectively.

Using this document, Game developers can develop and manage Script based games for Samsung Consumer devices more easily and effectively.

Target Readers

This document is aimed at Game Content developer companies and software programmers who intend to use Lua with SDL & SDL extended libs as basic engine for Game development. This document aims at audience who are interested to understand the guidelines for 2D game development for Samsung Consumer devices using Samsung SDK.

1. Introduction to Samsung Game Framework

This section describes what Samsung Game Framework (SGF) is, how it is configured and how it works.

1.1. What is Samsung Game Framework?

Samsung Game Framework is a Lua script based software framework which is used for development of 2D games for Samsung Consumer devices. It is a cross platform software architecture that would enable Game content developers to develop a game without building from the ground up. It ensures portability of the developed Game content by providing a ready set of cross platform APIs.

Samsung Game Framework is based on Simple DirectMedia Layer (SDL), SDL extension libraries and Lua Scripting Language. Game Content developers create Games in Lua using the Lua SDL interfaces. More details of Samsung Game Framework are described in the subsequent sections.

Samsung has also created various sample applications and Tutorials to extensively demonstrate the capabilities of Game Framework. This guide not only provides an introduction to Samsung Game Framework features and capability, but it also publishes extensive step-wise tutorials to help the Game content creators to use the power of Game Framework effectively.

1.1.1. What is Samsung SDK?

Software development for embedded systems is usually cross development and time consuming. Running the development environment in the target system is often infeasible and therefore the development environment runs in a separate host system using the target Emulator. Even though emulators usually cannot completely replace real target hardware, they often provide various advantages in software development. One obvious reason for using emulators is the target hardware dependencies are ruled out.

Samsung follows a similar philosophy and it has prepared a Software development kit (SDK) which

will be used by Game developers to create Games for Samsung Consumer electronics devices. Samsung Game Framework is packaged as part of Samsung SDK so that game developers get an easy environment to create the game contents very quickly and to effectively utilize the capabilities of Samsung Game Framework. Samsung SDK provides integrated Game framework in which developers develop games using API set on Windows (X86) on PC without the need of actual target system (such as DTV with Samsung Smart TV) and the same seamlessly runs on the latter.

One more benefit of using Samsung SDK is to achieve cross platform independence. Consumer Electronic devices may use different cores like ARM, MIPS, etc. If game content developers use Samsung SDK, the game content would be working fine for 2011 and later Samsung Consumer devices which are released. So game developers are free from worrying about the underlying hardware on which their games would be running on.

1.1.2. Supported Consumer Device Platform

At present, Samsung SDK can be used to create Game contents for Samsung 2011 Digital TV that support Samsung Smart TV feature.

1.1.3. What makes Samsung Game Framework powerful?

Samsung Game Framework is a Script based game framework comprising of SDL (and extension libraries) and Lua. It exposes its constituent's functionalities for the development of Game through Lua SDL interfaces. There are various features which makes Samsung Game Framework and SDK very powerful.

'Script' based Framework: One useful tool for rapid iteration during development is the use of interpreted scripts, if the Game Designer can write in a script language and immediately execute it on the platform (as opposed to a long recompile or download time), the iteration cycle can become quite fast, enabling many experiments and much more fine-tuning of Game play ideas.

Lua & SDL Based: Lua is a powerful, fast, lightweight, embeddable scripting language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. As per internet survey, Lua is being used as the favored scripting language for Game Content developers. SDL is most preferable library used by Game content developers for Graphics and other operations.

Samsung Game Framework is powered by SDL GLUE Layer which is a binding of SDL (and extension libraries) with Lua. Almost all the SDL (and extension libraries) APIs are supported by this framework and are available in Lua form to the Game content developers. [Refer Appendix A for List of supported APIs]. Game Framework also supports Lua SDL callback functions.

Note:
Samsung's Game Framework does NOT fully cover all SDL APIs in SDL Lua GLUE. Only the APIs which are supported for game developing on Samsung Consumer devices have been supported. [Refer Appendix A for List of supported APIs].

Cross platform Architecture: Various sub-components of Samsung Game Framework are based on Libraries SDL, SDL_Mixer, SDL_Image, SDL_net, SDL_ttf, Lua Interpreter and the Lua Abstraction which are available across platform. Samsung Game Framework ensures that a Game content developed on the above libraries will be in sync with all the libraries present on the target DTV system.

Target Independence: By using the game framework over SDK, rapid development of Games on Windows is possible and there is no need for cross compilation or cross checking of Game on Target. This saves lot of time. Developers can easily create and test their program which would run on target system without any modification (unless specified in this guide).

Open Source Approach: Samsung Game Framework is easier for common developers to use since open source based approach is followed in it. Game Developers will find it easier to get more information on the web about the technologies and concepts used in Samsung Game Framework. Both SDL (and extension libraries) and Lua both are extremely popular in the game development.

Tools & Support: Samsung Game framework comes with a set of tools (such as Lua script interpreter, optimized libraries, etc) which help to verify the Game content easily and effectively. Moreover, Game framework users can get technical support from Samsung to create the Game contents if they face any technical issue while using SDK.

The above features of the Samsung Game Framework make it powerful and easy to use for the game content developers.

1.1.4. Game Framework Architecture Overview

The Architecture of Samsung Game Framework consists of following components:

1. Game Engine – The game engine written in native Language having various components like
   - SDL
   - SDL_image
   - SDL_ttf
   - SDL_net
   - SDL_mixer
2. Script Game Widget – It will launch the Script based games.
3. Lua interpreter – It is developed to understand the Script based Games.
4. Lua Abstraction – It is the abstraction layer in Lua on SDL & SDL Extension libraries.

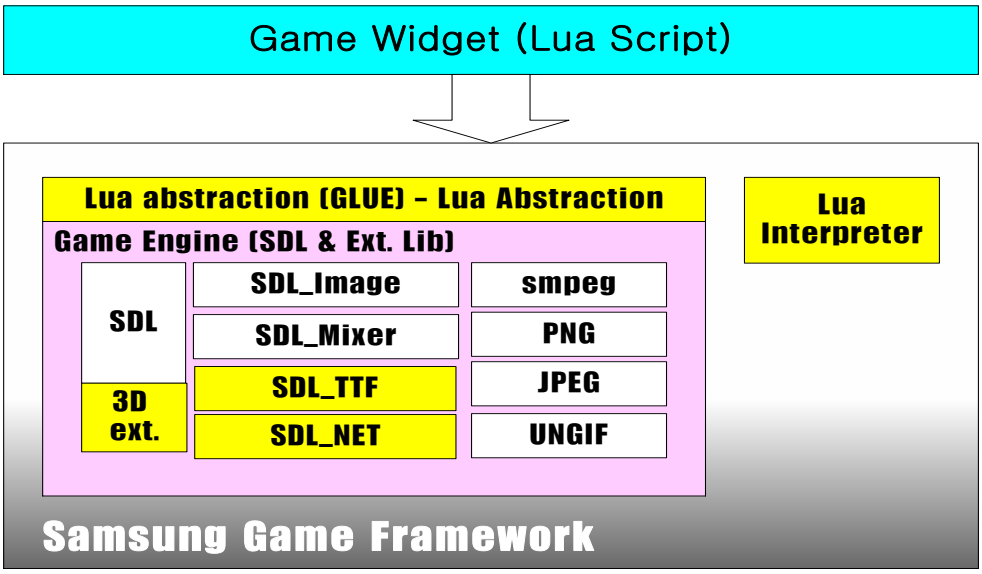


Figure: Samsung Game Framework Architecture

1.1.5. What is SDL?

Simple DirectMedia Layer (SDL) and its extension libraries (SDL_image, SDL_mixer, SDL_ttf, SDL_net) provides the necessary multimedia infrastructure for 2D games which includes

- Windowing support and event handling
- audio/ video
- image
- fonts
- networking

Besides, SDL is cross platform. It is used by MPEG playback software, emulators, and many popula r award winning games. For more detailed information, please visit http://www.libsdl.org/

1.1.6. What is SDL_image

SDL_image is an image loading library that is used with the SDL library, and almost as portable. It allows a programmer to use multiple image formats without having to code all the loading and conversion algorithms themselves.

For more detailed information on SDL_image, please visit is available at www.libsdl.org or more specifically The SDL_image homepage

1.1.7. What is SDL _mixer

SDL_mixer is a sound mixing library that is used with the SDL library, and almost as portable. It allows a programmer to use multiple samples along with music without having to code a mixing

algorithm themselves. It also simplifies the handling of loading and playing samples and music from all sorts of file formats.

SDL_mixer is available at www.libsdl.org or more specifically The SDL_mixer homepage

1.1.8. What is SDL_ttf

SDL_ttf is a TrueType font rendering library that is used with the SDL library, and almost as portable. It depends on freetype2 to handle the TrueType font data. It allows a programmer to use multiple TrueType fonts without having to code a font rendering routine themselves. With the power of outline fonts and ant aliasing, high quality text output can be obtained without much effort.

SDL_ttf is available at www.libsdl.org or more specifically The SDL_ttf homepage

1.1.9. What is SDL_net

SDL_net is a network library that is used with the SDL library, and almost as portable. It allows a programmer to use network functionality without having to code different things for different platforms. It also simplifies the handling of network connections and data transfer.

SDL_net is available at www.libsdl.org or more specifically The SDL_net homepage

1.1.10. What is Lua

And why choose Lua?

Well, Lua is a powerful, fast, lightweight, embeddable scripting language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

And Samsung Game Framework is powered with Lua!

Lua is a proven, robust language

Lua is currently the leading scripting language in games. Lua has been used in many industrial applications with an emphasis on embedded systems (e.g., the Ginga middleware for digital TV in Brazil) and games (e.g., World of Warcraft). Lua has a solid reference manual and there are several books about it. Several versions of Lua have been released and used in real applications since its creation in 1993.

Lua is fast

Lua has a deserved reputation for performance. It is an aspiration of other scripting languages to claim to be "as fast as Lua". Several benchmarks show Lua as the fastest language in the realm of interpreted scripting languages. Lua is fast not only in fine-tuned benchmark programs, but in real life too. A substantial fraction of large applications have been written in Lua.

Lua is portable

This enables the game developers to develop games in Lua and SDl and ext libs on Windows X86 based PCs and the same games will run seamlessly on Samsung’s Samsung Smart TV product for post-2011 targets.

Lua is embeddable

Lua is a fast language engine with small footprint that you can embed easily into your application. Lua has a simple and well documented API that allows strong integration with code written in other languages.

Lua is powerful (and simple)

Lua provides meta-mechanisms for implementing classes and inheritance. Lua's meta-mechanisms bring an economy of concepts and keep the language small, while allowing the semantics to be extended in unconventional ways.

Lua is small and free

Adding Lua to an application does not bloat it. The Samsung Game Framework provides the Lua interpreter (version 5.1.4) packed and ready for use by game content developers. Lua is free software, distributed under a very liberal license (the well-known MIT license). For more details, please visit http://www.lua.org/.

Please Note:

For security reason and protection of our product, some Lua functions may not be supported on Samsung's product.

1.1.11. What is LuaSDL

LuaSDL is a glue/abstraction layer of Lua on SDL & Ext Libs (SDL_Mixer, SDL_Image, SDL_net, and SDL_ttf). This architecture allows Game content developers to use SDL interfaces in Lua scripts.

Below figure depicts the S/W data flow in Lua Abstraction Framework. There are various components which executes the control flow from native SDL to Lua Abstraction layer as follows:

1. **Script Game Launcher**: It is responsible to execute the Lua Script Game.
2. **Script game contents**: It includes all the game resources. Game script calls the APIs of Lua Abstraction layer & so provides similar functionality to end user.
3. **Lua Abstraction Layer**: This layer is basically wrapper over the corresponding native Library. Like for SDL & SDL ext Libs, it has Lua based data types & APIs support which has interaction with respective native data types & APIs.
4. **Native SDL & SDL ext Libs**: Framework consists of both Native & Lua Abstraction layer to provide integrated solution for product development.

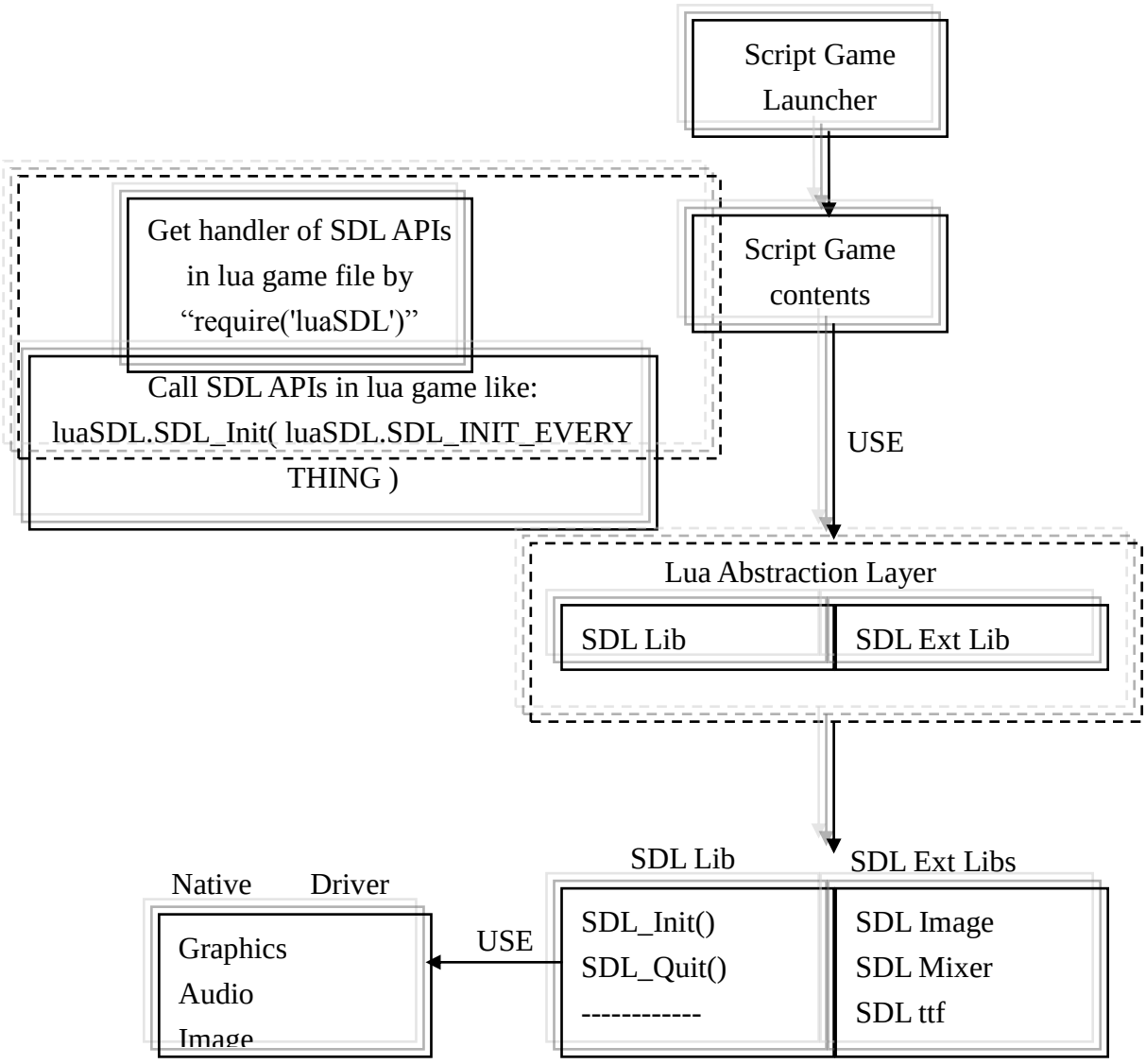


Figure: S/W data flow in Lua Abstraction Framework

1.2. Pre-requisites

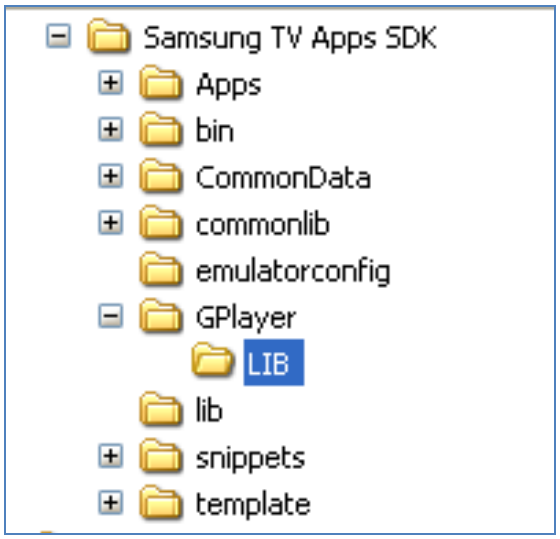
This document is aimed at programmers who are familiar with usage of Lua scripting language, and SDL & ext libs (as graphics library). This tutorial aims at providing the guidelines for 2D game development using Game Framework.

This document would also be very useful for engineers at Samsung Developer Forum (SDF) who are supporting Game content developers for technical issues encountered with Samsung SDK.

1.3. Directory structure and internals of the Game Framework

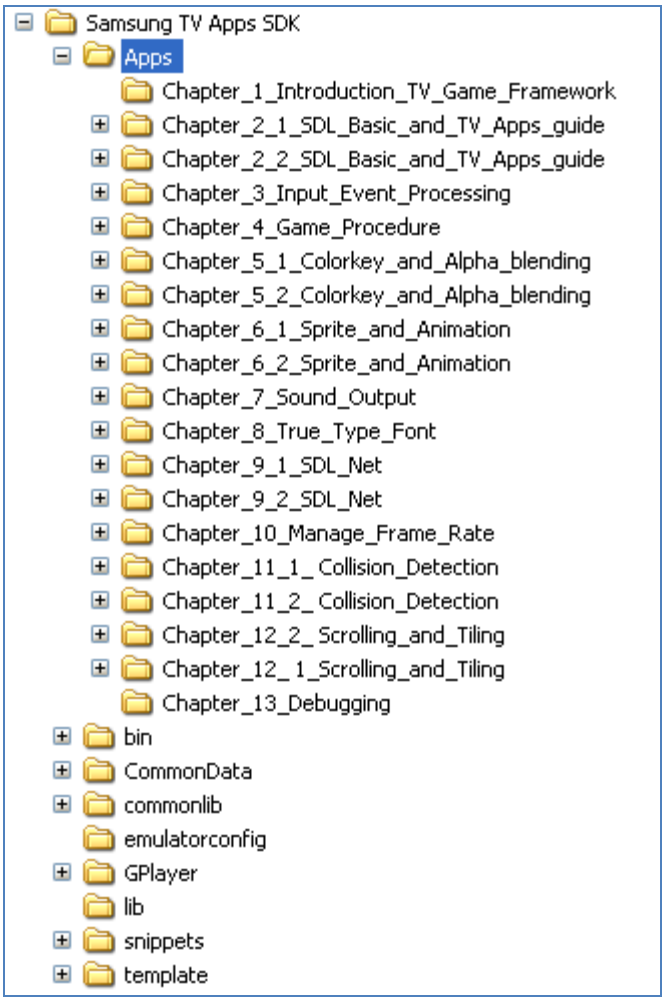
1.3.1. Directory structure of the Engine

LIB contains GLUE layer dlls and .lua files for SDL:



1.3.2. Application Distribution

Applications are distributed chapter-wise as follows:



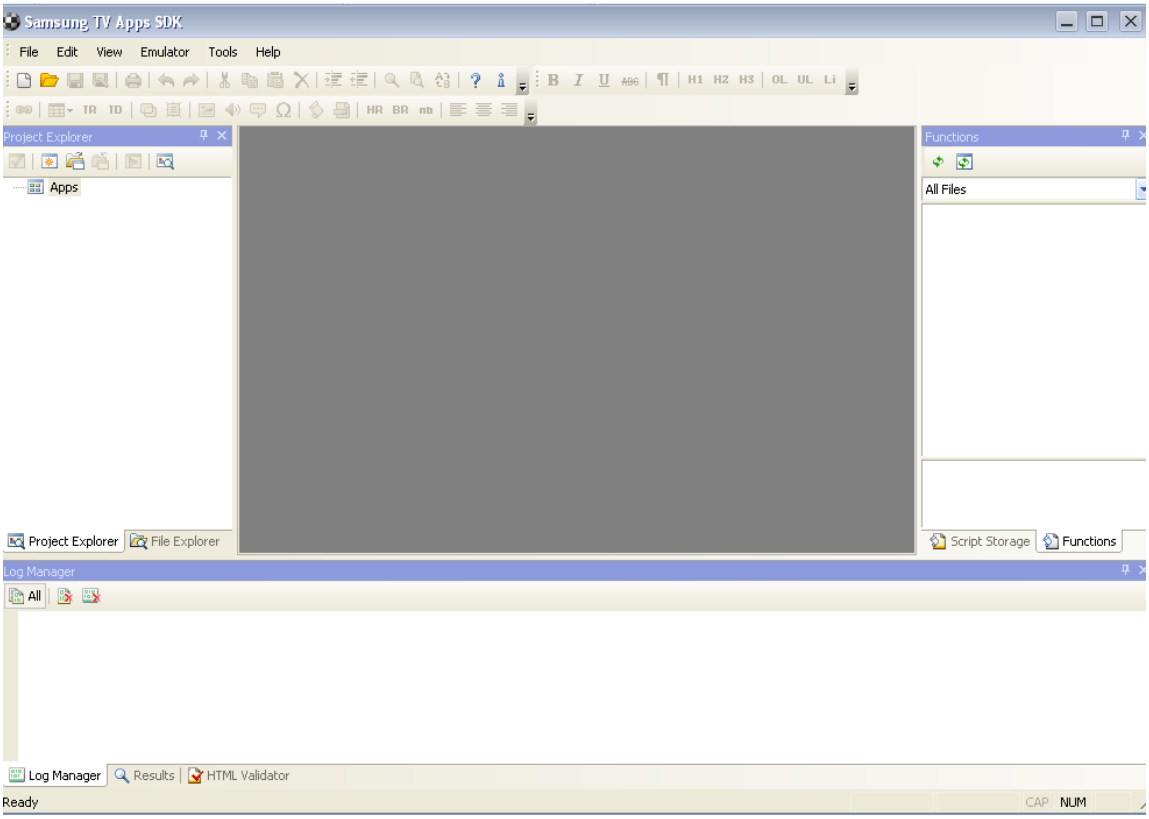
1.4. Running Tutorial Applications

Following steps need to be followed to run the sample applications or any other developed application using the Samsung Smart TV SDK:

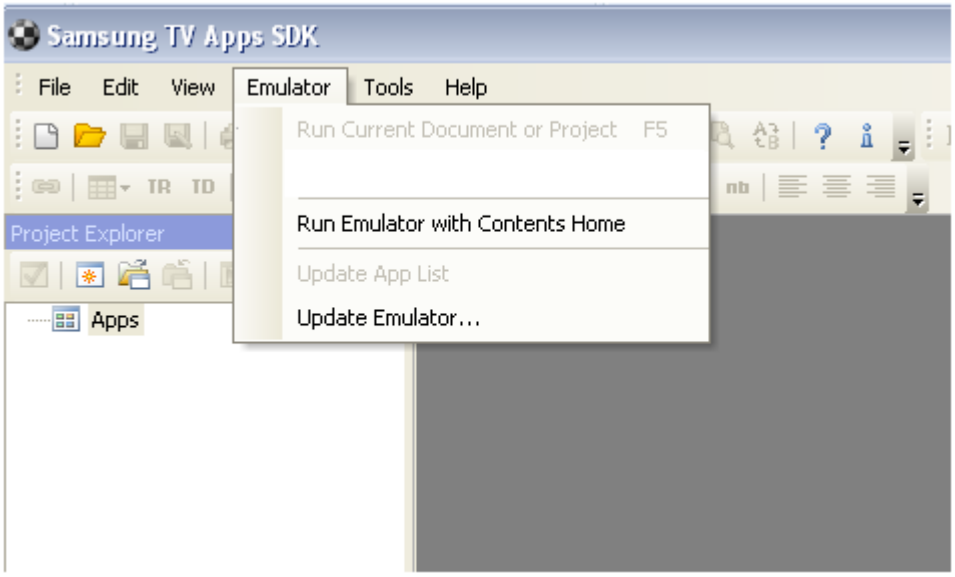
- Application should have one ‘Main.lua’ file which contains Game_Main(resPath) as main execution function. Application may contain several files besides Main.lua file.

- Copy the folder containing application to the ‘Apps’ directory in the installation root of Samsung Smart TV SDK (<***Installation Directory***>\Samsung Smart TV SDK\Apps).

- Run Samsung Smart TV SDK.



- "Run emulator with Contents Home" from Emulator menu of Samsung Smart TV SDK.



- All the applications copied into Apps directory will be loaded and corresponding application widgets will be displayed in the Samsung Smart TV SDK Emulator.

- Select the corresponding application widget by traversing using UP / DOWN / LEFT / RIGHT keys of remocon provided on the right side of the Emulator.

- Execute the selected application using ENTER key of the remocon provided on the Emulator.

- Applications can exit using RETURN / EXIT / BLUE keys of remocon provided on the Emulator.

- Some sample applications may exit after a specified time as per the design of the applications. Other exit methods (RETURN / EXIT / BLUE keys) may not be supported in these applications to retain the simplicity of these sample applications.

2. SDL Basic and TV Apps guide

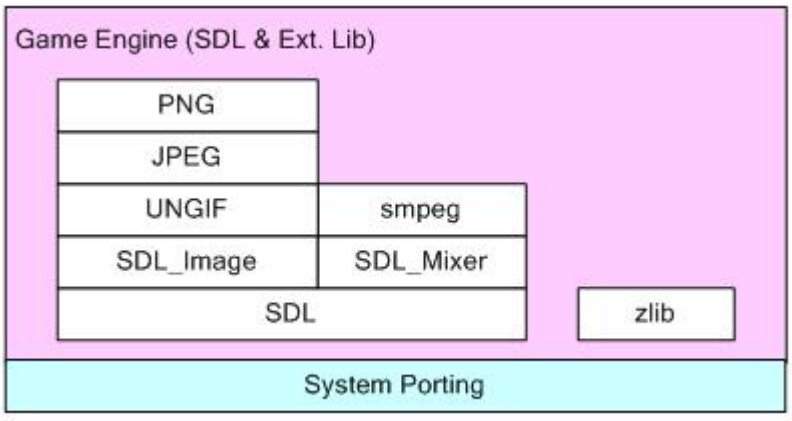
This chapter starts with basic functionality of SDL in Samsung Game Framework and a walkthrough about usage of TV Apps guide.

The foremost application using SDL is to display an image. So in the next two subsections we will display images using our Game Framework.

In this section, we will make you familiar with TV Apps guide.

Internet @ TV Scope : As you will notice Internet @ TV is rich collection of user applications. It consists of GPlayer, for game launching. As GPlayer is part of the DTV platform, Game Engine interacts with it to execute a game.

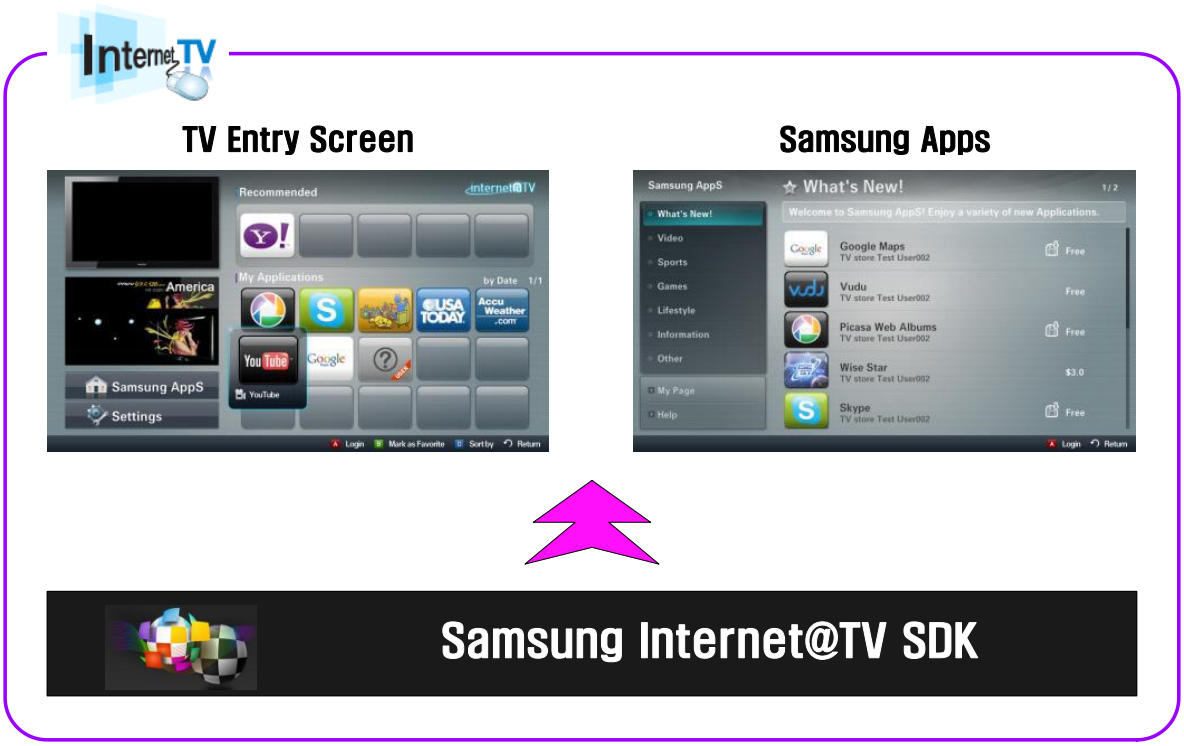
Game engine : The game engine supports applications using SDL and extension libraries - SDL_image, SDL_ttf, SDL_net and SDL_mixer. Below diagram lists the support of SDL_Image and SDL_Mixer extention libraries.



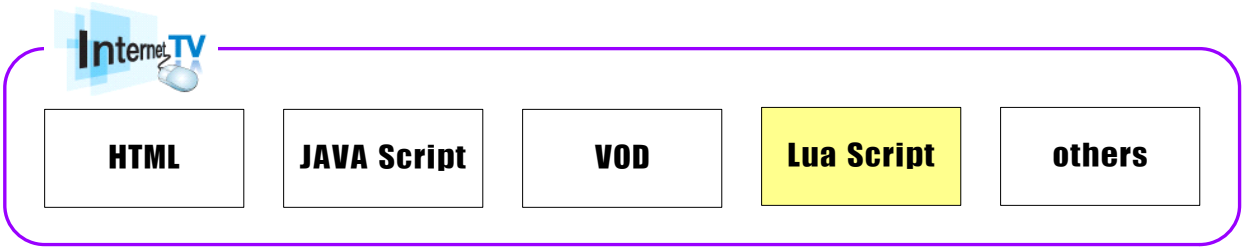
SDL provides support for event handling, audio, and video.

Samsung Smart TV SDK:

Interactive@TV graphical user interface has very user friendly look and feel.



The Samsung Smart TV supports a number of features such as HTML, JAVA Script, Lua Script etc. So a user has wide variety of choices to develop an application.



Samsung Smart TV Support Feature:

Here is complete TV's LuaSDL support feature:

Graphics

A. Screen size : 960*540
B. Pixel per bits : 32bit (ARGB8888)
C. SDL_VideoFlags : SDL_HWSURFACE

2. **Audio** (SDL_AudioSpec)
   A. Freq (samples per second : Hz) : 16000, 22050, 24000, 32000, 44100, 48000
   B. Format (Audio Data Format) : AUDIO_S16
   C. Channels (number of channels : 1 mono, 2 stereo) : 2

Programming rule

Now we will provide a brief of SDL events mapping with Remocon keys, which will help in programming a game.

1. Exit game
   A. SDLK_ESCAPE : Remote controller BLUE or RETURN key
      i. Showing POP-up Quit yes or no (optional)
      ii. Select yes -> quite the game
   B. SDLK_POWER : Remote coltroller POWER, CONTENTS, INTERNET, CH Up/Down
      i. Quit the game as soon as possible with out pop-up message box.

2. Global variable
   A. g*[GameName]_Variable* : basically using namespace

3. TV Remote control key guide line

Device	Key Value	SDL Event		Action guideline
		type	key.keysym.sym	
TV Remote Controller	Enter	SDL_KEYDOWN	SDLK_z	Basic Select
		SDL_KEYUP		
	Up	SDL_KEYDOWN	SDLK_UP	Up
		SDL_KEYUP		
	Down	SDL_KEYDOWN	SDLK_DOWN	Down
		SDL_KEYUP		
	Left	SDL_KEYDOWN	SDLK_LEFT	Left
		SDL_KEYUP		
	Right	SDL_KEYDOWN	SDLK_RIGHT	Right
		SDL_KEYUP		
	Red Key	SDL_KEYDOWN	SDLK_HOME	Home
		SDL_KEYUP		
	Green Key	SDL_KEYDOWN	SDLK_F1	Option1
		SDL_KEYUP		
	Yellow Key	SDL_KEYDOWN	SDLK_F2	Option2
		SDL_KEYUP		
	Blue Key	SDL_KEYDOWN	SDLK_ESCAPE	Exit
		SDL_KEYUP		
	1	SDL_KEYDOWN	SDLK_1	Number 1
		SDL_KEYUP		
	2	SDL_KEYDOWN	SDLK_2	Number 2
		SDL_KEYUP		
	3	SDL_KEYDOWN	SDLK_3	Number 3
		SDL_KEYUP		
	4	SDL_KEYDOWN	SDLK_4	Number 4
		SDL_KEYUP		

	5	SDL_KEYDOWN SDL_KEYUP	SDLK_5	Number 5
	6	SDL_KEYDOWN SDL_KEYUP	SDLK_6	Number 6
	7	SDL_KEYDOWN SDL_KEYUP	SDLK_7	Number 7
	8	SDL_KEYDOWN SDL_KEYUP	SDLK_8	Number 8
	9	SDL_KEYDOWN SDL_KEYUP	SDLK_9	Number 9
	0	SDL_KEYDOWN SDL_KEYUP	SDLK_0	Number 0

Samsung Smart TV metadata guide line (config.xml)

Any application which has be excuted using Samsung Smart TV, needs to have a config.xml file. This file will make the integration of the application with Samsung Smart TV seamlessly.

A brief guidance is provided here about how to use the basic tags in the metadata file.

Basic tag information:

1. <cpname> : Development company name
2. <thumbIcon> : Interent@TV home icon(106x87)
3. <BigThumbIcon> : Interent@TV home big icon(115x95)
4. <ListIcon> : Interent@TV home list icon(85x70)
5. <BigListIcon> : Interent@TV home big list icon(95x78)
6. <contents> : Lua contents entry file name
   A. Eg. [lua file name]
7. <ver> : Samsung Apps displayed widget version
8. <widgetname> : widget's name

Config.xml example

Metadata Example

```
<?xml version="1.0" encoding="UTF-8"?>
<widget>
   <icon></icon>
    <preview></preview>
    <previewimg></previewimg>
    <previewjs></previewjs>

    <cpname>Com2Us</cpname>
    <cpauthjs>Com2Us</cpauthjs>
    <movie>y</movie>
    <ThumbIcon>icon/MiniGamePack_icon_kor_trial_106x87.png</ThumbIcon>
    <BigThumbIcon>icon/MiniGamePack_icon_kor_trial_115x95.png</BigThumbIcon>
    <ListIcon>icon/MiniGamePack_icon_kor_trial_85x70.png</ListIcon>
    <BigListIcon>icon/MiniGamePack_icon_kor_trial_95x78.png</BigListIcon>

    <category>game</category>
    <apptype>14</apptype>
    <contents>Main.lua </contents>

   <srcctl>n</srcctl>
   <ver>1.002</ver>
   <ticker>n</ticker>
   <audiomute>n</audiomute>
   <videomute>n</videomute>
   <login>n</login>
   <childlock>n</childlock>
```

```
   <dcont>y</dcont>
   <widgetname>MiniGamePack Trial KOR</widgetname>
```

```
        <cplogo></cplogo>
        <preIcon></preIcon>
        <description>
         MiniGamePack Trial KOR
        </description>
        <width>960</width>
        <height>540</height>
        <author>
                <name>Samsung Electronics Co. Ltd.</name>
           <email></email>
                <link>http://www.sec.com</link>
                <organization>Samsung Electronics Co. Ltd.</organization>
        </author>
    </widget>
```

The config .xml file is read by the engine and the applications are listed under My Applications. Game developers have the choice to add an icon and name for their application/widget which gives it an identity. User can browse through the list and upon selecting one, the application in the contents head of the config .xml is launched. The Remote control functions as user input device. Go through the TV remote control guidelines to program the application for inputs.

2.1. Image (BMP) Display

The default image format support is BMP for Samsung Game Framework. This chapter explains how to create a simple application that loads and displays a BMP image on the screen.

2.1.1. Introduction

SDL only provides support for display of BMP images. To display other image formats, we will use SDL_image extension library (Described in next subsection 2.2).

The image of BMP format is loaded onto the SDL_Surface and painted onto the screen.

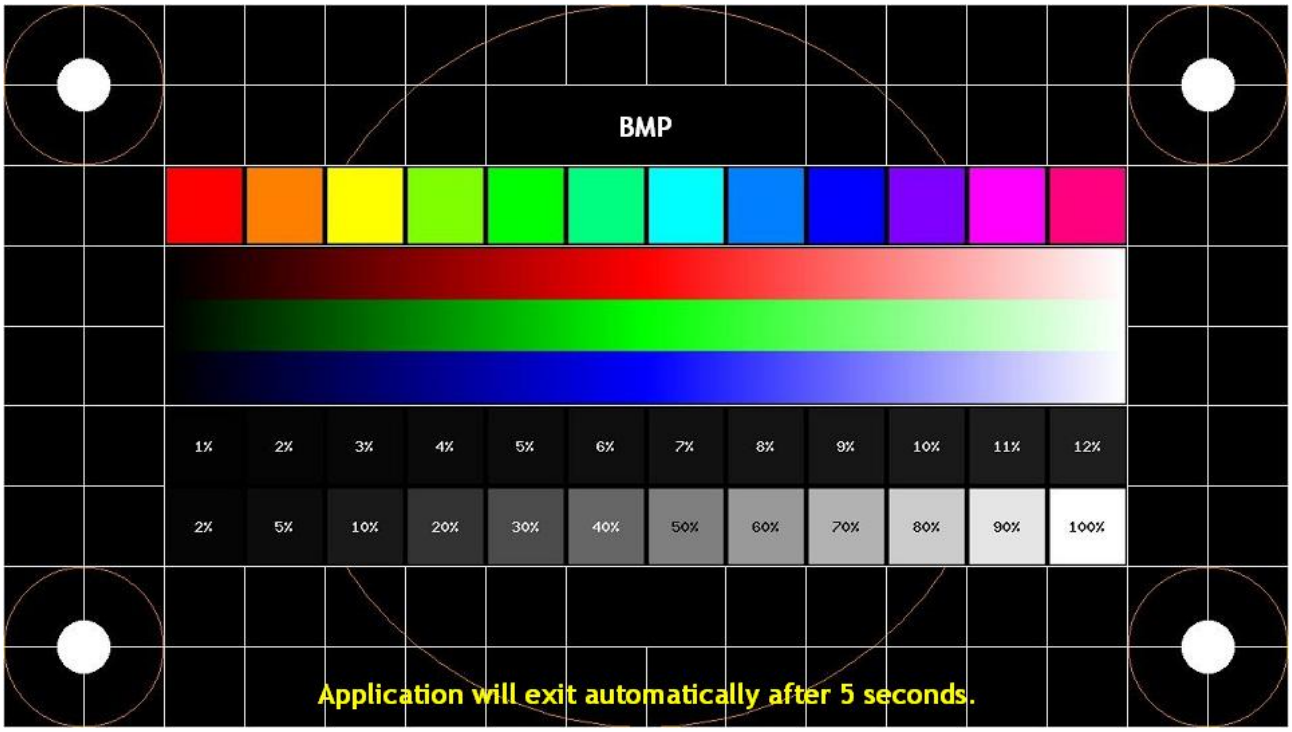


Figure: BMP Image

Let's begin with how to display bitmap images using Samsung Game Framework.

2.1.2. Steps

1. We begin with including appropriate SDL headers and libraries at the start of the program. Only after that one can use SDL APIs to build the game application.

   We start by including the desired includes, Dlls, shared libs, Lua files.
   Here, 'SDL' is the Lua wrapper on SDL file.
   Then we define all global variables/data that are used in the program.

   Every game would have audio, video, images etc resources. These resources must be placed at a common location specified by chResPath. The path chResPath should be defined relative to ***resPath*** argument to the ***Game_Main*** function.

2. Programmers must use **SDL_UpperBlit** () to use **SDL_BlitSurface** () API.

```
require('SDL')

SDL_BlitSurface = SDL_UpperBlit

SCREEN_WIDTH    = 960      -- Screen Width
SCREEN_HEIGHT = 540      -- Screen Height

chResPath = nil
```

   Please Note:
   Game_Main(resPath) is the main execution function which this guide uses in all the chapters. This function takes one input ***resPath*** which defines the location of directory containing the application ***xml file***. All the resources used in the game can be defined relative to this path.

3. We now define the ***Game_Main*** execution function that encapsulates the program logic. ***LoadGameContents*** will set the required package path for the resource files.

```
function Game_Main(resPath)
        LoadGameContents(resPath)
        chResPath = resPath.."/Res/"
        ….
end

function LoadGameContents(dirPath)
        package.path = package.path..";"..dirPath.."/?.lua;"
end
```

4. Initialize SDL sub-system and set the video mode with specified width, height and bits per pixel.

5. It is good to ensure correct initialization through API return codes.

```
--Initialize SDL
if(-1 == SDL_Init(SDL_INIT_VIDEO)) then
        print("Couldn't Initialize SDL: "..SDL_GetError())
end

--Set up the screen
screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32,
                                SDL_HWSURFACE)
if(nil == screen) then
        print("Couldn't Set Video Mode: "..SDL_GetError())
end
```

6. Load the image that is to be displayed on screen. The BMP image can be loaded using ***SDL_LoadBMP()*** API.

```
local tempImage = SDL_LoadBMP( chResPath.."sampleimagebmp.bmp" )
if( nil == tempImage ) then
        print( "Couldn't Load Image: "..SDL_GetError() )
end

-- Set Display Format
image = SDL_DisplayFormat(tempImage)
if ( nil == image ) then
        print( "Couldn't Set Display Format:" ..SDL_GetError() )
        CleanUp()
end

-- Map RGB components
SDL_FillRect( screen, screen.clip_rect, SDL_MapRGB( screen.format, 0xFF, 0xFF,
                0xFF ) )
```

7. Blit the loaded image with screen using ***SDL_BlitSurface***() API. ***SDL_BlitSurface***() performs a fast blitting from the source surface to the destination surface. To display the image, the source surface is image and the destination surface is screen.

   .

```
--Apply image to screen
if(-1 == SDL_BlitSurface(image, nil, screen, nil)) then
        print("Couldn't BLIT surfaces: "..SDL_GetError())
end
```

8. To render the image on screen, we call ***SDL_Flip***() API. For the displayed image to be perceived by viewer, it must stay on the screen for some time. ***SDL_Delay***() API pauses the program execution so that the image is visible for this time period.

```
--Update Screen
if(-1 == SDL_Flip(screen)) then
        print("Couldn't Update Screen: "..SDL_GetError())
end
```

9. Before the program exits, it is important to free all allocated surfaces and quit the SDL sub-system by calling ***SDL_Quit***() API.

```
--Free the optimized image
SDL_FreeSurface(image)

--Quit SDL
SDL_Quit()
```

10. The application will automatically exit after 5 seconds.

2.1.3. Remarks

Game Framework supports more image display formats like JPG, PNG and GIF. For these formats please follow next subsection 2.2.

2.2. Image Display (PNG, JPG, and GIF)

2.2.1. Introduction

This chapter explains how to create a simple application that loads and displays an image (JPG) on the screen.
SDL provides an extension library called SDL_image which supports JPEG, PNG and simple GIF image formats. The images of these formats are loaded onto the ***SDL_Surface*** and painted onto the screen.

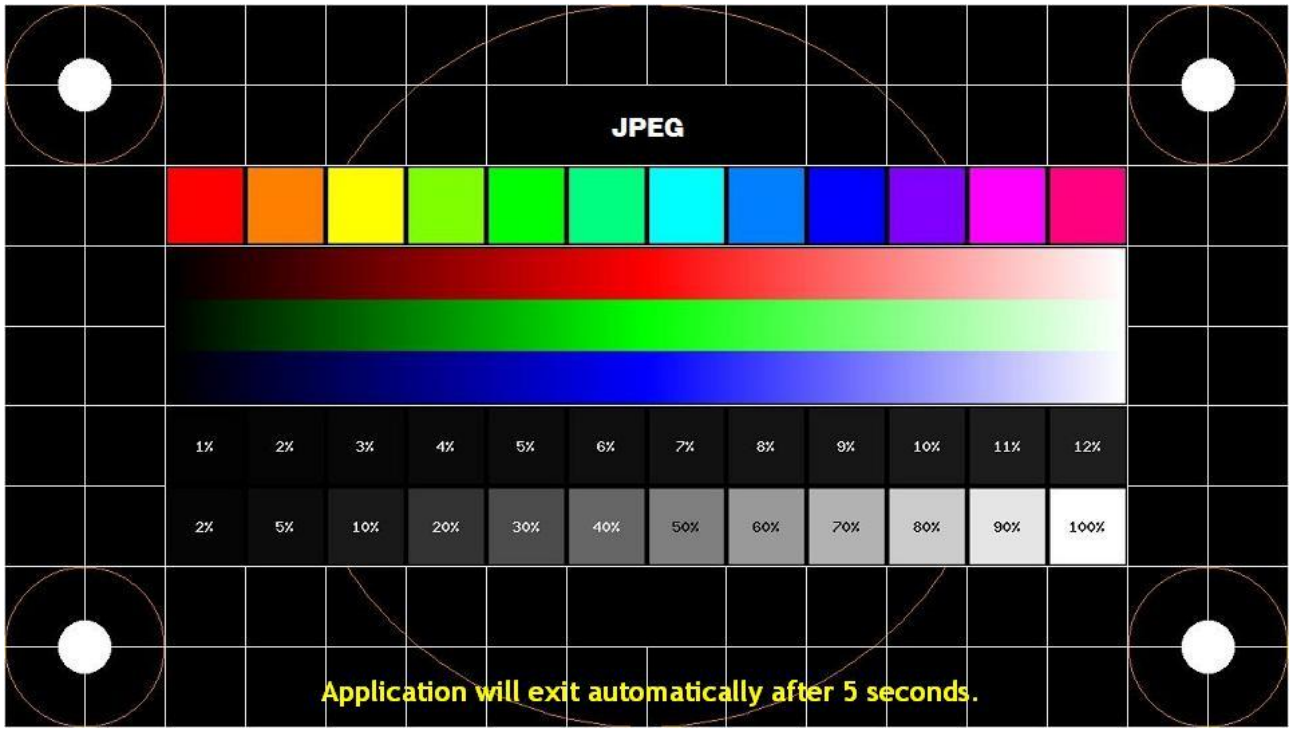


Figure: JPG Image

2.2.2. Steps

1. Include appropriate SDL headers and libraries at the start of the program. Only after that one can use SDL APIs to build the game application.

Here SDL include file will provide the basic framework for SDL functionality and SDL_image is the include file which provides the functionality for loading images.

```
require('SDL')
require('SDL_image')
```

2. Using below assignment, programmers can use SDL_BlitSurface() API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. Define all global variables/data that are used in the program.

```
SCREEN_WIDTH = 960      -- Screen Width
SCREEN_HEIGHT = 540     -- Screen Height
chResPath = nil
```

4. Define the execution function that encapsulates the program logic. Every game would have audio, video, images etc resources. These resources must be placed at a common location which will be specified by ***chResPath***. ***chResPath*** should be defined relative to the input path to the ***Game_Main*** function. ***LoadGameContents*** will update the package path.

```
function Game_Main(resPath)
        LoadGameContents(resPath)
        chResPath = resPath.."/Res/"
        ….
end

function LoadGameContents(dirPath)
        package.path = package.path..";"..dirPath.."/?.lua;"
end
```

5. Initialize SDL sub-system and set the video mode with specified width, height and bits per pixel. It is good to ensure correct initialization through API return codes.

```
--Initialize SDL
if(-1 == SDL_Init(SDL_INIT_VIDEO)) then
        print("Couldn't Initialize SDL: "..SDL_GetError())
end

--Set up the screen
screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32,
                                SDL_HWSURFACE)
if(nil == screen) then
        print("Couldn't Set Video Mode: "..SDL_GetError())
end
```

6. For loading and displaying images of PNG, JPEG and GIF image formats, SDL_image library will be used. ***IMG_Load***() API loads the image into memory.

```
-- Load image
local tempImage = IMG_Load( chResPath.."sampleimagejpg.JPG" )
if( nil == tempImage ) then
        print( "Couldn't Load Image: "..SDL_GetError() )
end

-- Set Display Format
image = SDL_DisplayFormat(tempImage)
if ( nil == image ) then
        print( "Couldn't Set Display Format:" ..SDL_GetError() )
end
```

7. Next, blit the loaded image with the screen, update the screen and pause the execution of the program so that the image can be perceived. It is exactly how we displayed the image in default (.bmp) format.

```
--Apply image to screen
if(-1 == SDL_BlitSurface(image, nil, screen, nil)) then
        print("Couldn't BLIT surfaces: "..SDL_GetError())
end

--Update Screen
if(-1 == SDL_Flip(screen)) then
        print("Couldn't Update Screen: "..SDL_GetError())
end

--Pause
SDL_Delay(5000)
```

8. Before the program exits, it is important to free all allocated surfaces and quit the SDL sub-system by calling ***SDL_Quit***() API.

9. The application will automatically exit after 5 seconds.

2.2.3. Remarks

None

3. Input Event Processing

This sample application describes about the input event functionality on DTV. The application focus on polling various types of events and its event state. There can be various types of events like Key press, Key release etc.

For demonstration, the application includes Navigation keys, Number keys and other miscellaneous keys. It will also make the user familiar with key mapping methodology to handle Samsung Remocon color keys like Red, Green, and Yellow. The application will keep on polling for events and can display the key value based on mapping done.

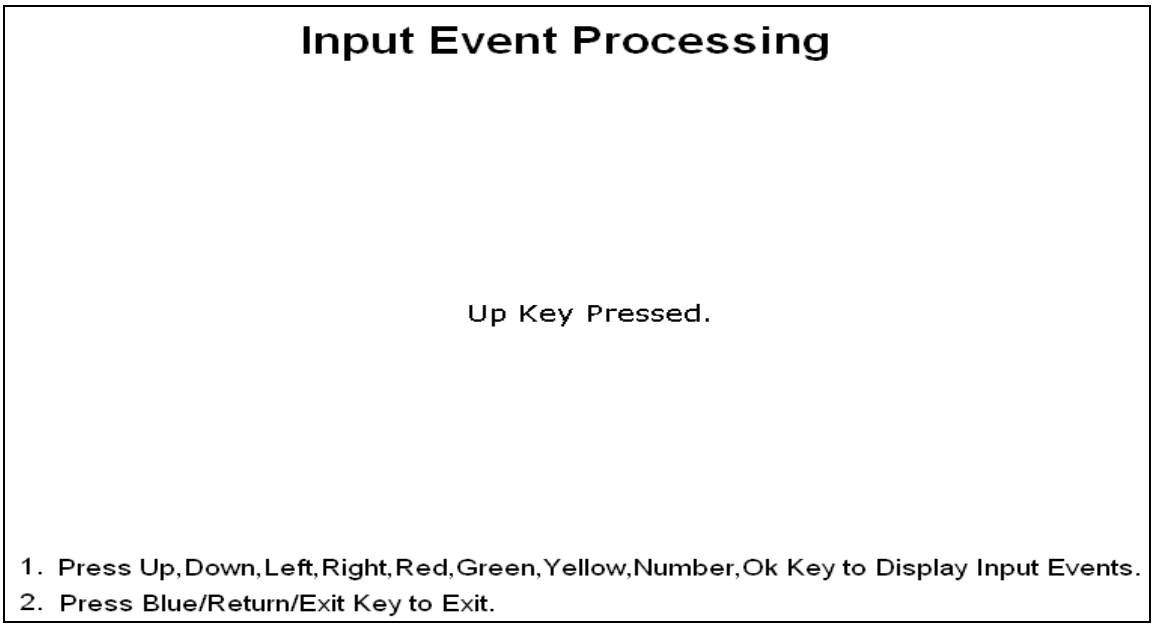
3.1. Introduction

This LUA key event application highlights the steps involved to demonstrate ***SDL_PollEvent***(), Input event type & key values. User can see the prints on screen as and when any Remocon key is pressed.

Event handling allows application to receive input from the user. Event handling is initialized (along with video) with a call to ***SDL_Init***(SDL_INIT_VIDEO).

Internally, SDL stores all the events waiting to be handled in an event queue. Using functions like ***SDL_PollEvent*** user can observe and handle waiting input events.

The key to event handling in SDL is the SDL_Event union. The event queue itself is composed of a series of SDL_Event unions, one for each waiting event. SDL_Event unions are read from the queue with the ***SDL_PollEvent*** API and it is then up to the application to process the information stored with them.



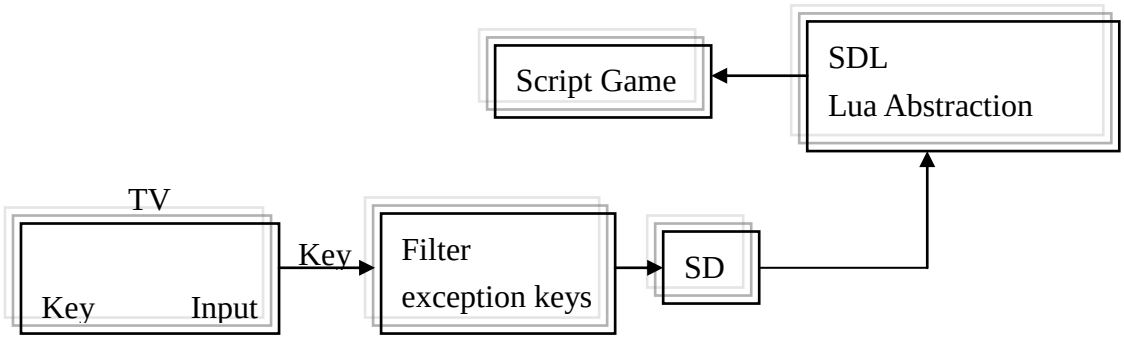


Figure: Key Input control to Lua SDL Framework

Key control can be well described from above figure. It consists of Key Input module. The generated key events are passed to SDL layer. SDL has its specific Key events and collect from its event queue by polling it.

As Lua Abstraction layer has access to native layer, key gets transferred to abstraction layer successfully. Script game calls the Abstraction layer APIs and so get the same key event via its abstraction layer.

Remote Control Key Action:

On TV some SDL events are modified to support Remocon.

void DUMMY_InitOSKeymap(_THIS)

This function takes the Remote key input and translates to SDK virtual key.

Following table illustrates the mapping between Remocon keys and corresponding SDL key events.

Remocon Key	**SDL Key Event**	**Description**
dKEY_EXIT	SDLK_POWER	Direct Exit
dKEY_RETURN	SDL_QUIT	Exit Message Box
dKEY_0	SDLK_0	Number Key
dKEY_1	SDLK_1	Number Key
dKEY_2	SDLK_2	Number Key
dKEY_3	SDLK_3	Number Key
dKEY_4	SDLK_4	Number Key
dKEY_5	SDLK_5	Number Key
dKEY_6	SDLK_6	Number Key
dKEY_7	SDLK_7	Number Key
dKEY_8	SDLK_8	Number Key
dKEY_9	SDLK_9	Number Key
dKEY_JOYSTICK_OK	SDLK_z	Action
dKEY_JOYSTICK_UP	SDLK_UP	Up
dKEY_JOYSTICK_DOWN	SDLK_DOWN	Down
dKEY_JOYSTICK_LEFT	SDLK_LEFT	Left
dKEY_JOYSTICK_RIGHT	SDLK_RIGHT	Right
dKEY_RED	SDLK_HOME	Home
dKEY_GREEN	SDLK_F1	Option1
dKEY_YELLOW	SDLK_F2	Option2
dKEY_BLUE	SDLK_ESCAPE	Exit

3.2. Steps

1. To use key event functionality of SDL, include SDL headers and libraries with the current program as below.
   Here, to display the print message on screen corresponding to the any input key, we will use true type fonts. True Type Fonts require SDL_ttf extension library of SDL so it is included in this application. True Type Font functionality of SDL will be described in Chapter 8 of this guide with details.

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
```

2. Initialize the SDL subsystem Instance and set video mode surface with specified width, height and bits per pixel. Also verify the Initialization of the SDL with proper error message. Alongwith this, we will need to initialize the ttf module as well to display the true type font messages on screen.

```
-- Initialize SDL
if (0 ~= SDL_Init(SDL_INIT_AUDIO or SDL_INIT_VIDEO) ) then
        print("Error:" .."Unable to initialize SDL: " ..SDL_GetError())
         return 1
else
        print("Sucess:".."Initialization of SDL is done: ")
end

videoflags = SDL_HWSURFACE
--Set video mode
Screen  =SDL_SetVideoMode(SCREEN_WIDTH,SCREEN_HEIGHT,VIDEO_BPP,
          videoflags)
if (nil == screen) then
        print("Error:".."Couldn't initialize SDL_SetVideoMode:
        " ..SCREEN_WIDTH ..SCREEN_HEIGHT..SDL_GetError() )
end
```

3. Event Loop will handle the events like SDL_KEYDOWN, SDL_KEYUP, SDLK_LEFT, SDLK_RIGHT, SDLK_UP, SDLK_DOWN, SDLK_ESCAPE, SDL_QUIT and several other keys. ***TTF_RenderText_Solid*** is used to render a specified message on screen.

```
function loop()
    quit = 1
    while ( 1==quit and SDL_PollEvent(event)) do
        if (event.type == SDL_QUIT) then
            print("Quitting")
            quit = 0
          elseif (event.type == SDL_KEYDOWN) then
                if (event.key.keysym.sym == SKEY_RETURN) then
                    fontSurface = TTF_RenderText_Solid( font, "Enter Key
                                  Pressed.", color )
                    Draw()
                    print("Enter Key ")
                    break
                elseif (event.key.keysym.sym == SDLK_LEFT) then
                    fontSurface = TTF_RenderText_Solid( font, "Left Key
                                  Pressed.", color )
                    Draw()
                    print("Left Key ")
                    break
                elseif (event.key.keysym.sym == SDLK_RIGHT) then
                    fontSurface = TTF_RenderText_Solid( font, "Right Key
                                  Pressed.", color )
                    Draw()
                    print("Right Key ")
                    break
                elseif (event.key.keysym.sym == SDLK_UP) then
                    fontSurface = TTF_RenderText_Solid( font, "Up Key
                    Pressed.", color )
                    Draw()
```

```
                print("Up Key ")
                break
        elseif (event.key.keysym.sym == SDLK_DOWN) then
                fontSurface = TTF_RenderText_Solid( font, "Down Key
                        Pressed.", color )
                Draw()
                print("Down Key ")
                break
        elseif event.key.keysym.sym == SDLK_HOME then
                fontSurface = TTF_RenderText_Solid( font, "Red Key
                        Pressed.", color )
                Draw()
                print("Red Key ")
                break
        elseif event.key.keysym.sym == SDLK_F1 then
                fontSurface = TTF_RenderText_Solid( font, "Green Key
                        Pressed.", color )
                Draw()
                print("Green Key ")
                break
        elseif event.key.keysym.sym == SDLK_F2 then
                fontSurface = TTF_RenderText_Solid( font, "Yellow
                        Key Pressed.", color )
                Draw()
                print("Yellow Key ")
                break
        elseif (event.key.keysym.sym == SDLK_ESCAPE) then
                fontSurface = TTF_RenderText_Solid( font, "Quitting
                        Application.", color )
                Draw()
                print("Quitting")
                quit = 0
                break
```

```
elseif ( event.key.keysym.sym == SDLK_0 ) then
        fontSurface = TTF_RenderText_Solid( font, "0 Number
        Key Pressed.", color )
        Draw()
        print("0 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_1 ) then
        fontSurface = TTF_RenderText_Solid( font, "1 Number
        Key Pressed.", color )
        Draw()
        print("1 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_2 ) then
        fontSurface = TTF_RenderText_Solid( font, "2 Number
        Key Pressed.", color )
        Draw()
        print("2 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_3 ) then
        fontSurface = TTF_RenderText_Solid( font, "3 Number
        Key Pressed.", color )
        Draw()
        print("3 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_4 ) then
        fontSurface = TTF_RenderText_Solid( font, "4 Number
        Key Pressed.", color )
        Draw()
        print("4 Number Key Pressed")
        break
```

```
elseif ( event.key.keysym.sym == SDLK_5 ) then
        fontSurface = TTF_RenderText_Solid( font, "5 Number
        Key Pressed.", color )
        Draw()
        print("5 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_6 ) then
        fontSurface = TTF_RenderText_Solid( font, "6 Number
        Key Pressed.", color )
        Draw()
        print("6 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_7 ) then
        fontSurface = TTF_RenderText_Solid( font, "7 Number
        Key Pressed.", color )
        Draw()
        print("7 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_8 ) then
        fontSurface = TTF_RenderText_Solid( font, "8 Number
        Key Pressed.", color )
        Draw()
        print("8 Number Key Pressed")
        break
elseif ( event.key.keysym.sym == SDLK_9 ) then
        fontSurface = TTF_RenderText_Solid( font, "9 Number
        Key Pressed.", color )
        Draw()
        print("9 Number Key Pressed")
        break
```

```
                elseif ( event.key.keysym.sym == SDLK_z ) then
                        fontSurface = TTF_RenderText_Solid( font, "Action
                        Key Pressed.", color )
                        Draw()
                        print("Action Key Pressed")
                else
                        --nothing
                end
        end
    end
    return quit
end
```

4. Draw function will render to the screen.

```
function Draw()
    -- Apply the image to screen
    SDL_BlitSurface( image, nil, screen, nil )

    -- Apply the font to the screen
    posHeadline = SDL_Rect_new()
    posHeadline.x = ( SCREEN_WIDTH - fontSurface.w )/2
    posHeadline.y = ( SCREEN_HEIGHT - fontSurface.h )/2

    -- Apply the font to the screen, flip the screen and pause for a while
    SDL_BlitSurface( fontSurface, nil, screen, posHeadline )
    SDL_Flip( screen )
    SDL_Delay( 200 )
end
```

5. Font and Image will be loaded to demonstrate different event effects by displaying corresponding texts and images on screen and call the loop for input.

```
-- Keyboard repeat rate
SDL_EnableKeyRepeat(300, 80)

-- Load the image to be displayed
local tempImage = IMG_Load( chResPath.."background.bmp" )

if( nil == tempImage ) then
        print( "Couldn't Load Image: "..SDL_GetError() )
end

-- Set Display Format
image = SDL_DisplayFormat(tempImage)
if ( nil == image ) then
        print( "Couldn't Set Display Format:" ..SDL_GetError() )
end

-- Initialize TTF
if( -1 == TTF_Init() )then
        print( "Couldn't Initialize TTF: "..SDL_GetError() )
end

-- Load the font to be displayed
font = LoadFont( chResPath.."verdana.TTF" )

-- Color for the font
color = SDL_Color_new()
color.r = 0
color.g = 0
color.b = 0
```

```
-- Apply the image to screen
SDL_BlitSurface( image, nil, screen, nil )

-- Update screen
SDL_Flip( screen )

event = SDL_Event_new()

-- Loop for input
while(Loop() == 1) do
end

-- delete the event
SDL_Event_delete(event)
```

6. It is important to quit the SDL sub-system before exiting from the application.

```
--Quit SDL
SDL_Quit()
```

3.3. Remarks

Always create the new SDL Event type handler using **SDL_Event_new**() before starting the Polling.

4. Game Procedure

This sample application describes about the game procedure, key input to control the game and status of game execution.

4.1. Introduction

Game loop is a mechanism through which game continues to run until an event to exit the game is provided to the game. The game exit condition may be time out situation, press of escape key or the logic of the game which triggers the exit of game.

Once a game is started, it will continue to be in a running status till a game exit condition is provided to the game.

This key input and image display application highlights the steps involved to demonstrate game procedure (key input and image display) functionality of SDL LUA scripts.

4.2. Steps

1. As illustrated in previous chapters, we need to include SDL, SDL_image, and SDL_ttf header and library with the current program:

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
```

2. Using below assignment, programmers can use **SDL_BlitSurface** API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. As in the previous examples, declare all global variables/data that is used in the program.

```
SCREEN_WIDTH    = 960      -- Screen Width
SCREEN_HEIGHT = 540        -- Screen Height
BPP = 32

chResPath = nil
color = SDL_Color_new() -- Font color
color.r = 255
color.g = 0
color.b = 0
```

4. Initialize the SDL subsystem Instance and set video mode surface with specified width, height and bits per pixel. Also verify the Initialization of the SDL with proper error message. TTF submodule also needs to be initialized to use the fonts for displaying help statements.

```
--Initialize the SDL
local InitRes = SDL_Init(SDL_INIT_VIDEO)
if InitRes == 0 then
        local Screen
        Screen, ErrStr = MakeScreen()
        ……
else
        ErrStr = debug.traceback(SDL_GetError())
end
--Initialize TTF
if ( -1 == TTF_Init() ) then
        print( "Couldn't Initialize TTF: " ..SDL_GetError() )
end
```

5. MakeScreen () API is used to Returns a “screen” (a canvas within a window upon which the sprites will be drawn).

```
local function MakeScreen()
        local ErrStr
        local Screen = SDL_SetVideoMode(SCREEN_WIDTH,
                        SCREEN_HEIGHT, BPP, SDL_HWSURFACE)
        if not Screen then
                ErrStr = debug.traceback(SDL_GetError())
        end
        return Screen, ErrStr
end
```

6. Create a sprite to be controlled by the user and another to be controlled by the program. Make different Sprite with different Image. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by chResPath. Game Framework passes the path of the application directory. chResPath can be defined relative to it.

```
local function Game_Main(resPath)

        LoadGameContents(resPath)
        chResPath = resPath.."/Res/"
        ……
        local UserSprite
        UserSprite, ErrStr = MakeSprite(Screen, chResPath.."jerry.bmp")
        if UserSprite then
                AutoSprite, ErrStr = MakeSprite(Screen, chResPath.."tom.bmp")
                if AutoSprite then
                 AutoSprite1, ErrStr = MakeSprite(Screen, chResPath.."jerry2.bmp")
                        ……
                end
        end
end

function LoadGameContents(dirPath)
        package.path = package.path..";"..dirPath.."/?.lua;"
end
```

7. Load the the font which needs to be used to display and render the text to surface.

```
local function Game_Main()
        ……..
        ……..
        -- Load the font to be displayed
        font = LoadFont(chResPath.."verdana.TTF", 30)
        if ( nil == font ) then
                print( "Unable to load font:" ..SDL_GetError() )
        end

        fontBottom = LoadFont(chResPath.."verdana.TTF", 20)
        if ( nil == fontBottom ) then
                print( "Unable to load font:" ..SDL_GetError() )
        end

        -- Render the font text to surface
        fontSurface1 = TTF_RenderText_Solid( font, "Sample Application - Game
                        Procedure", color )

        fontSurface2 = TTF_RenderText_Solid( fontBottom, " Application will exit
                        if Jerry is caught or one of the BLUE/RETURN/EXIT
                        buttons is pressed.", color )

        fontSurface3 = TTF_RenderText_Solid( fontBottom, "Move Jerry using
                        Navigation Keys.", color )

        ……..
        ……..
end
```

8. Apply the font to the screen for the headline and bottomline

```
local function MainLoop(Screen, UserSprite, AutoSprite,AutoSprite1)
        ……….
        -- Apply the font to the screen for the headline
        posHeadline1 = SDL_Rect_new()
        posHeadline1.x = ( SCREEN_WIDTH - fontSurface1.w )/2
        posHeadline1.y = SCREEN_HEIGHT/10
        SDL_BlitSurface( fontSurface1, nil, Screen, posHeadline1 )

        -- Apply the font to the screen for the bottomline
        posBottomline2 = SDL_Rect_new()
        posBottomline2.x = ( SCREEN_WIDTH - fontSurface2.w )/2
        posBottomline2.y = SCREEN_HEIGHT * 9/10
        SDL_BlitSurface( fontSurface2, nil, Screen, posBottomline2 )

        -- Apply the font to the screen for the bottomline
        posBottomline3 = SDL_Rect_new()
        posBottomline3.x = ( SCREEN_WIDTH - fontSurface3.w )/2
        posBottomline3.y = SCREEN_HEIGHT * 8.5/10
        SDL_BlitSurface( fontSurface3, nil, Screen, posBottomline3 )
```

9. Load the image that is to be displayed on screen. To preserve readability of code, the operations to load the image are moved into a separate function, MakeSprite. Input parameter to MakeSprite function is the image file name which needs to be loaded and the screen, it returns the sprite object.

```
-- Returns a sprite object; ImgName is the filename of a .bmp file:
local function MakeSprite(Screen, ImgName)

    local tempImage = IMG_Load(ImgName)
    local Img = SDL_DisplayFormat(tempImage)
    SDL_FillRect( Screen, Screen.clip_rect,
                  SDL_MapRGB( Screen.format, 0xFF, 0xFF, 0xFF ) )
  if Img then
            local Background = SDL_MapRGB(Screen.format, 255, 255, 255)
            -- Current X and Y positions:
            local CurX, CurY = 0, 0

            -- Current velocities along the X and Y axes, in pixels per tick
            (minimum -1, maximum 1):
            local VelX, VelY = 0, 0
            ……
            -- Tables to be (re)used as rectangle arguments to SDL_FillRect and
            SDL_BlitSurface:

            FillRect = SDL_Rect_new();
            ……
            BlitRect =SDL_Rect_new();
```

```
-- The sprite object:
Sprite = {}
function Sprite:Move(X, Y)
        local Succ, ErrStr
        X, Y = math.floor(X or CurX), math.floor(Y or CurY)

        -- Erase the sprite at its current position:
        FillRect.x, FillRect.y = CurX, CurY
        if SDL_FillRect(Screen, FillRect, Background) == 0 then
                -- Write it to its new position:
                BlitRect.x, BlitRect.y = X, Y
                if SDL_BlitSurface(Img, nil, Screen, BlitRect) ==
                0 then


……….
-- Call this once for every tick:
function Sprite:Tick(Ticks)
        ……….

        NewX = CurX + Sign(VelX)
        -- Make sure it doesn't go off the edges:
        NewX,Fix= ComputeBounce(NewX, 0,self:ScreenWidth() -
                self:Width())
        if Fix then
                VelX = -VelX
        end
        ----------
        -- Make sure it doesn't go off the edges:  local Fix
        NewY, Fix = ComputeBounce(NewY, 0,self:ScreenHeight()
        - self:Height())
        ----------
end
```

```
        -- Accelerates the sprite along the X axis (negative values of
        -- Accel accelerate to the left):
        function Sprite:AccelX1(Accel)
                if Accel ~= 0 then
                        VelX = Between(VelX + Accel * VelInc*32, -1, 1)
                        ……………
                end
        end
        function Sprite:AccelX(Accel)
                if Accel ~= 0 then
                        VelX = Between(VelX + Accel * VelInc, -1, 1)
                end
        end

        -- Accelerates the sprite along the Y axis (negative values of
        -- Accel accelerate upward):
        function Sprite:AccelY(Accel)
                if Accel ~= 0 then
                        ……..
                end
        end
        function Sprite:AccelY1(Accel)
                ……..
        end
        ……
        -- Give the sprite its initial position:
        local Succ
        Succ, ErrStr = Sprite:Move()
        ……
end
```

10. Returns N, or (if N is lower than Min), a number as much above Min as N is below it, or (if N is higher than Max), a number as much below Max as N is above it; also returns a second value telling whether had to return a number other than N: ComputeBounce() API.

```
local function ComputeBounce(N, Min, Max)
	local Fix = false
	if N > Max then
		N = Max - (N - Max)
		Fix = true
	elseif N < Min then
		N = Min + (Min - N)
		Fix = true
	end
	return N, Fix
end
```

11. Apply the image to screen (appropriate surface). This performs a fast blit from the source surface to the destination surface.

```
if SDL_BlitSurface(Img, nil, Screen, BlitRect) == 0 then
	CurX, CurY = X, Y
	Succ = true
else
	Succ, ErrStr = false, debug.traceback(SDL_GetError())
end
```

12. Now for moving the sprite on the screen, give them their initial positions and velocities.

```
local function Game_Main()
        ……
        -- Give them their initial positions and velocities:
        local ScreenWidth, ScreenHeight = Screen.w, Screen.h
        Succ, ErrStr = UserSprite:Move( ScreenWidth / 3 - UserSprite:Width() / 2,
                        ScreenHeight / 2 - UserSprite:Height() / 2)
        if Succ then
                -- Go in one of the four diagonal directions depending
                -- on what time it is:
                local Time = math.mod(os.time(), 4) + 1
                local A = 5 -- AutoSprite initial acceleration.
                local AccelX = ({A, A, -A, -A})[Time]
        end
end
```

13. While loop is for verifying the event type for drawing an Image.

```
Succ,ErrStr=MainLoop(Screen,UserSprite,AutoSprite,AutoSprite1)
```

14. Inside the MainLoop API the Event Instance is initiated for verifying the event type. In the MainLoop verifying the condition of quit from application if Sprite1 and Sprite2 overlapping.

```
local function Overlap(Sprite1, Sprite2)
        -- This views the two sprites as rectangle-shaped, even if they
        -- look like other shapes.
        local DiffX = math.abs(Sprite1:X() - Sprite2:X())
        local Width = (Sprite1:Width() + Sprite2:Width()) / 2
        ……
        return Ret
end
```

15. Move the user sprite and automatic sprite the appropriate distance for the given amount of ticks.

```
local function Step(UserSprite, AutoSprite,AutoSprite1, Ticks)
	local Succ, ErrStr = true

	-- Move the sprites:
	Succ, ErrStr = UserSprite:Tick(Ticks)
	if Succ then
		Succ, ErrStr = AutoSprite:Tick(Ticks)
		if Succ then
			Succ, ErrStr = AutoSprite1:Tick(Ticks)
		end
	end
	return Succ, ErrStr
end
```

16. Make sure the given area is updated on the given screen; it makes sure any changes to the given area of the screen are made visible. The rectangle must be confined within the screen boundaries because there's no clipping.

```
-- Update the entire screen:
SDL_UpdateRect(Screen, 0, 0, 0, 0)
```

17. It is important to de-allocate the surfaces and quit the SDL sub-system before exiting the application.

```
--Delete Event
SDL_Event_delete(event)

-- Delete the SDL Rect
SDL_Rect_delete(FillRect)
SDL_Rect_delete(BlitRect)

--Free the loaded image
SDL_FreeSurface(image)

--Free the loaded surfaces
SDL_FreeSurface(fontSurface1)
SDL_FreeSurface(fontSurface2)
SDL_FreeSurface(fontSurface3)

--Close the TTF
TTF_Quit()

--Close the SDL
SDL_Quit()
```

4.3. Remarks

None

5. Colorkey and Alpha Blending

In this chapter, we will learn how to use colorkey and alpha blending to simulate transparent objects.

5.1. Colorkey

5.1.1. Introduction

This subsection of the chapter explains about the sample application for color key functionality with SDL on DTV.

Color key is a functionality provided by the SDL by which it sets the transparent pixels in a blittable surface.

The color key application highlights the steps involved to demonstrate color key functionality of SDL on screen. In the below output color key will remove the red, green and blue colors of the surface blitted on the screen. This results in the transparent pixels corresponding to the specified color.

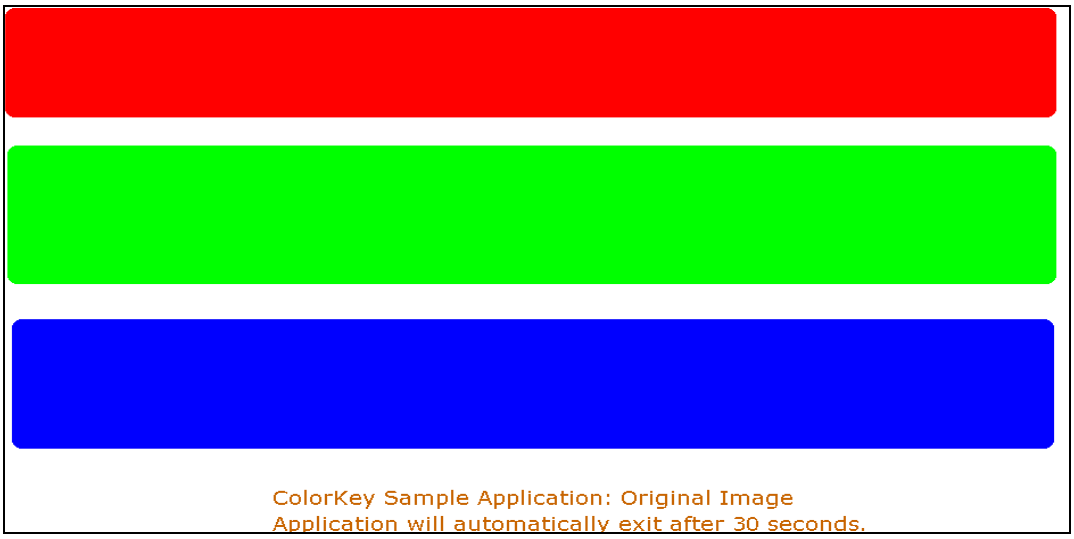


Figure: Original Image

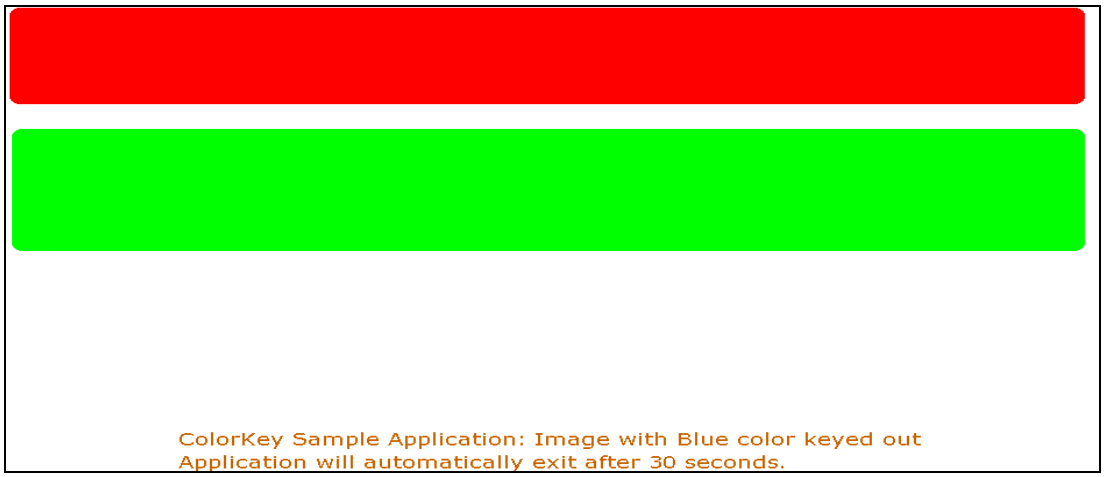


Figure: Color Keyed Image – Blue Color

As you can see, the blue background of the image has colorkeyed to background color.

5.1.2. Steps

1. To use color key functionality of SDL, include SDL_image header and library with the current program as below. SDL_ttf is included to use true type fonts functionality which we will learn in separate chapter on true type fonts.

```
require('SDL')
require('SDL_ttf')
require('SDL_image')
```

2. Using below assignment, programmers can use SDL_BlitSurface API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. As in the above examples, declare all global variables/data that is used in the program.

```
SCREEN_WIDTH    = 960      -- Screen Width
SCREEN_HEIGHT = 540      -- Screen Height
chResPath = nil
```

4. Initialize the SDL subsystem Instance and set video mode surface with specified width, height and bits per pixel. Also verify the Initialization of the SDL is completed with the return value. TTF submodule also needs to be initialized to use the fonts for displaying help statements.

```
--Initialize the SDL
if(SDL_Init(SDL_INIT_VIDEO)==-1) then
        print("Couldn't SDL: "..SDL_GetError())
end

--Initialize TTF
if ( -1 == TTF_Init() ) then
        print( "Couldn't Initialize TTF: " ..SDL_GetError() )
end

--Set up screen
screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32,
                                    SDL_HWSURFACE)
if(screen==nil) then
        print("Couldn't Set Video Mode: "..SDL_GetError())
end
```

5. Load the background image and the color key image that is to be displayed on screen.

```
--Load the background
background = IMG_Load( chResPath.."background.bmp" )
…
-- Load the image to be displayed
local image = IMG_Load( chResPath.."colorkey.bmp" )
…
```

6. Load the the font which needs to be used to display the hints.

```
-- Load the font to be displayed
font = LoadFont( chResPath.."verdana.TTF" )
if ( nil == font ) then
        print( "Unable to load font:" ..SDL_GetError() )
        CleanUp()
end

function LoadFont(fontname)
        -- Open the font
        loaded_font = TTF_OpenFont( fontname, 20 )
        if ( nil == loaded_font ) then
                print( "Error:" ..fontname .."font could not be
                        opened:" ..SDL_GetError() )
        end
        return loaded_font
end
```

7. Set the color which needs to made transparent.

```
-- Colour which needs to be transparent
COLORKEYr = 255
COLORKEYg = 0
COLORKEYb = 0
```

8. For Map of RGB color value to a pixel format the SDL_MapRGB()

```
--Map the color key
color_key = SDL_MapRGB(image.format, COLORKEYr, COLORKEYg,
                                COLORKEYb)
```

SDL_MapRGB API:-

Maps the RGB color value to the specified pixel format and returns the pixel value as a 32-bit int. If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned. If the specified pixel format has an alpha component it will be returned as all 1 bits (fully opaque).

9. Once the color is mapped successfully, Set the color key to make specified color transparent so, that it can be rendered on the output screen

```
--Set the color key to make specified color transparent
if(SDL_SetColorKey(image, SDL_SRCCOLORKEY, color_key)==-1) then
        print("Warning: colorkey will not be used, reason: "..SDL_GetError())
end
```

SDL_SetColorKey sets the color key (transparent pixel) in a blittable surface and RLE acceleration, RLE acceleration can substantially speed up blitting of images with large horizontal runs of transparent pixels (i.e., pixels that match the key value). The key must be of the same pixel format as the surface, SDL_MapRGB is often useful for obtaining an acceptable value. The Return value is ‘0’ on success and ‘ -1’ on any error.

10. After setting the colorkey convert the surface to the display format before drawing the image. The new surface does formed has pixel format and colors of the video frame buffer, also an additional alpha channel for faster blitting onto the display surface

```
--Set the Display surface to display format
temp_image = SDL_DisplayFormatAlpha(image)
```

11. For rendering the images on the screen

```
--Draw the image on the screen with appropriate surface
Draw(screen, temp_image, str, strExit)

color = SDL_Color_new() -- Font color
color.r = 200
color.g = 100
color.b = 0

fontSurface = TTF_RenderText_Solid( font, str, color )

posHeadline = SDL_Rect_new()
posHeadline.x = ( SCREEN_WIDTH - fontSurface.w )/2
posHeadline.y = ( SCREEN_HEIGHT - fontSurface.h )
```

12. Apply the image to screen (appropriate surface), this performs a fast blit from the source surface to the destination surface.

```
--Apply the image to screen
SDL_BlitSurface(background, nil, screen, nil)

-- Apply the image to screen
SDL_BlitSurface( image, nil, screen, nil )

-- Apply the text to screen
SDL_BlitSurface( fontSurface, nil, screen, posHeadline )
```

13. Update screen or flush the screen for (SDL_Flip API swaps the display surfaces to render the image on output screen). On hardware that supports double-buffering, this function sets up a flip and returns. The hardware will wait for vertical retrace, and then swap video buffers before the next video surface blit or lock will return. On hardware that doesn't

support double-buffering or if SDL_SWSURFACE was set, this is equivalent to calling SDL_UpdateRect.

```
--Update Screen
SDL_Flip( screen )
```

14. For the image to be perceived / analysis by viewer, it must stay on the screen for some time. In this example, a pause the execution is introduced so that the image is visible for this time period

```
--Pause
SDL_Delay( 5000 )
```

15. It is important to de-allocated the surfaces and quit the SDL sub-system before exitting from the color key application.

```
--Free the loaded image
SDL_FreeSurface( image )
SDL_FreeSurface( background )

--Quit SDL
SDL_Quit()
```

16. Game_Main is the main entry function of the application to be called by the Game Framework. The input argument resPath defines the path of directory contanining the application config.xml file. The resource path chResPath should be defined relative to this path.

```
Game_Main(resPath)
        LoadGameContents(resPath)
        chResPath = resPath.."/Res/"
…
end

function LoadGameContents(dirPath)

        package.path = package.path..";"..dirPath.."/?.lua;"
end
```

5.1.3. Remarks

None

5.2. Alpha Blending

5.2.1. Introduction

Alpha blending is the process of combining a translucent foreground image with a background image, thereby producing a new blended image, as you can see below.

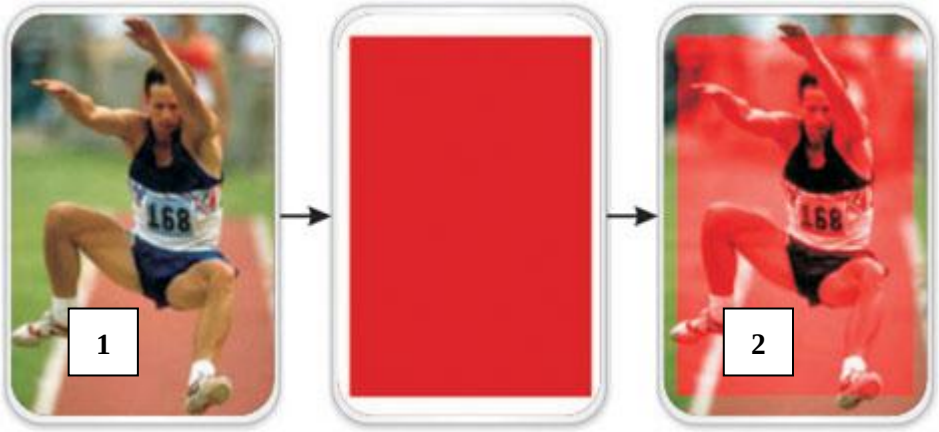


Figure: Alpha blending demonstration

The degree of the foreground image's translucency may range from completely transparent to completely opaque. If the foreground image is completely transparent, the blended image will be the background image. Conversely, if it is completely opaque, the blended image will be the foreground image.

As demonstrated in above figure, Picture 1 is original picture and picture 2 is an alpha blended image with lower transparency than original picture.

Samsung Game Framework supports another example scenario which is widely used in gaming and it is with usage of varied weighted alpha blending. It is depicted as follows.

Alpha (transparency) can have values between 0-1, and computed as a weighted average of the

foreground and background images.

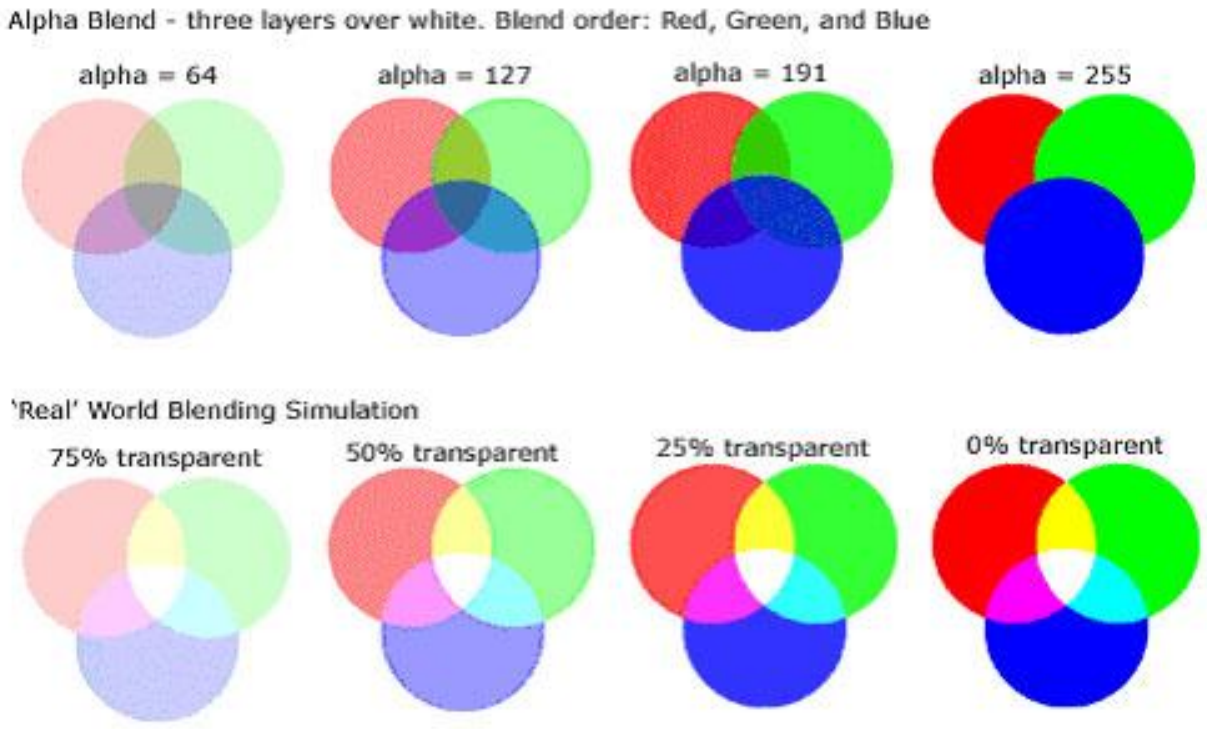


Figure: Varied alpha blending effect

Let's code for it now!
Following steps will explain how to provide alpha blending using Samsung Game Framework.

5.2.2. Steps

In previous chapters, you have well learnt how to include appropriate headers, initialize SDL subsystem, and set the video mode.

So we can proceed further with alpha blending now.

1. To start with, you have defined another variable, alpha which is the alpha value for the front image.

```
alpha = 255                -- Initial Alpha Value
```

2. Next comes loading the images that are to be displayed on screen. In this application, we load two images – frontImage and backImage to be displayed on top and background of screen respectively. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by ***chResPath***.

```
-- Load the background image to be displayed
backImage = IMG_Load( chResPath.."back.JPG" )
if(-1 == backImage) then
        print("Unable to load image: "..SDL_GetError())
        -- Close SDL
        SDL_Quit()
end
-- Load the front image to be displayed
frontImage = IMG_Load( chResPath.."front.JPG" )
if(-1 == frontImage) then
        print("Unable to load image: "..SDL_GetError())
        -- Close SDL
        SDL_Quit()
end
```

3. Draw() function sets the variable alpha value for the frontImage using ***SDL_SetAlpha***() API.

```
-- Draw the image on the screen
while(loop()) do
        Draw()
        -- Delay to visualize the Alpha Blending properly
        SDL_Delay(50)
end
```

4. Draw() then blits frontImage and backImage (taking into consideration the variable alpha value for the front image) with the screen and renders the blended image to output screen. You can write logic to vary alpha value as per game logic.

```
function Draw()
        -- Apply the alpha value to the front image
        SDL_SetAlpha(frontImage, SDL_SRCALPHA, alpha)
        local temp_frontImage = SDL_DisplayFormat(frontImage)

        -- Apply the back image to screen
        SDL_BlitSurface(backImage, nil, screen, nil)

        -- Apply the front image to the screen
        SDL_BlitSurface(temp_frontImage, nil, screen, nil)

        -- Reduce the alpha value in each iteration
        alpha = alpha – 5

        -- Update screen
        SDL_Flip(screen)

        --Free the loaded image
        SDL_FreeSurface( temp_frontImage )
end

function loop()
        -- End the loop when front image is fully blended
        if (alpha==0) then
                return 0      -- Exit
        else
                return 1      -- Continue
        end
end
```

5.2.3. Output

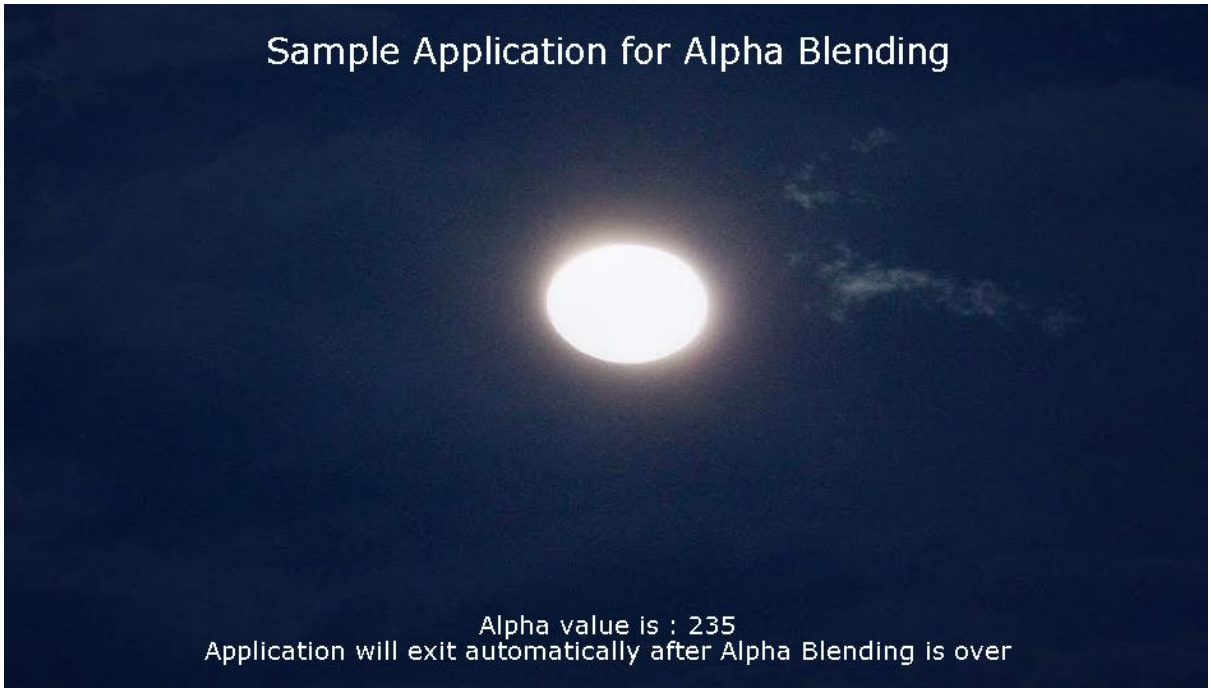


Figure: Original Image

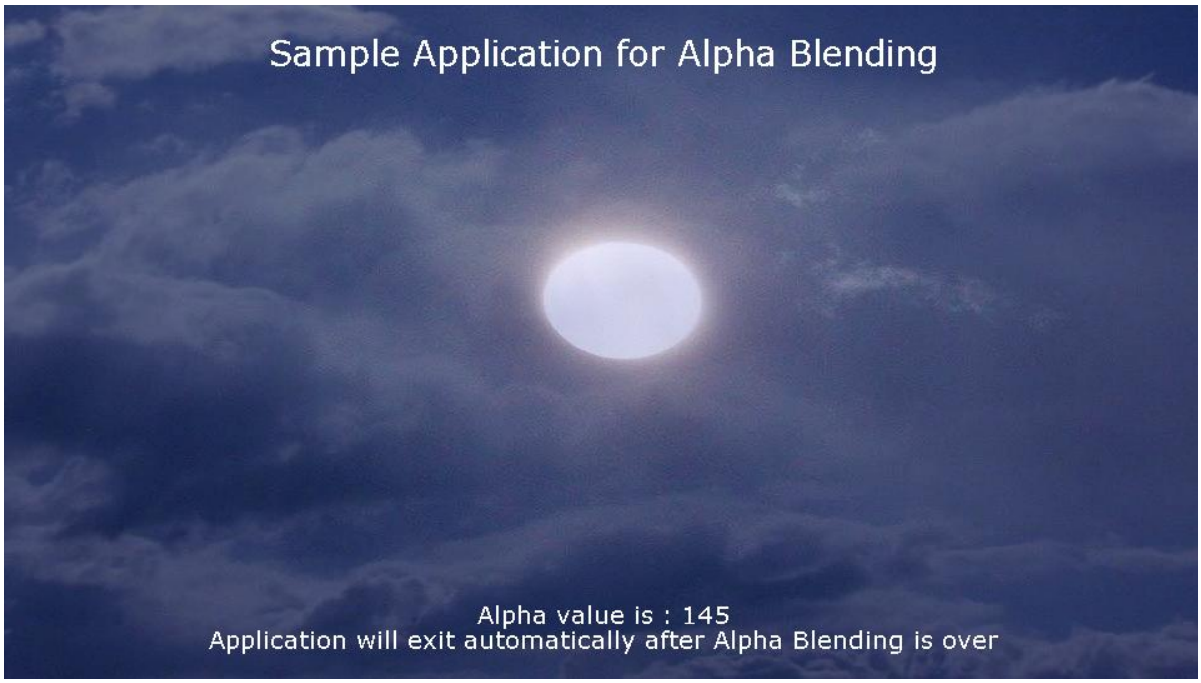


Figure: Alpha Blending – Intermediate stage1

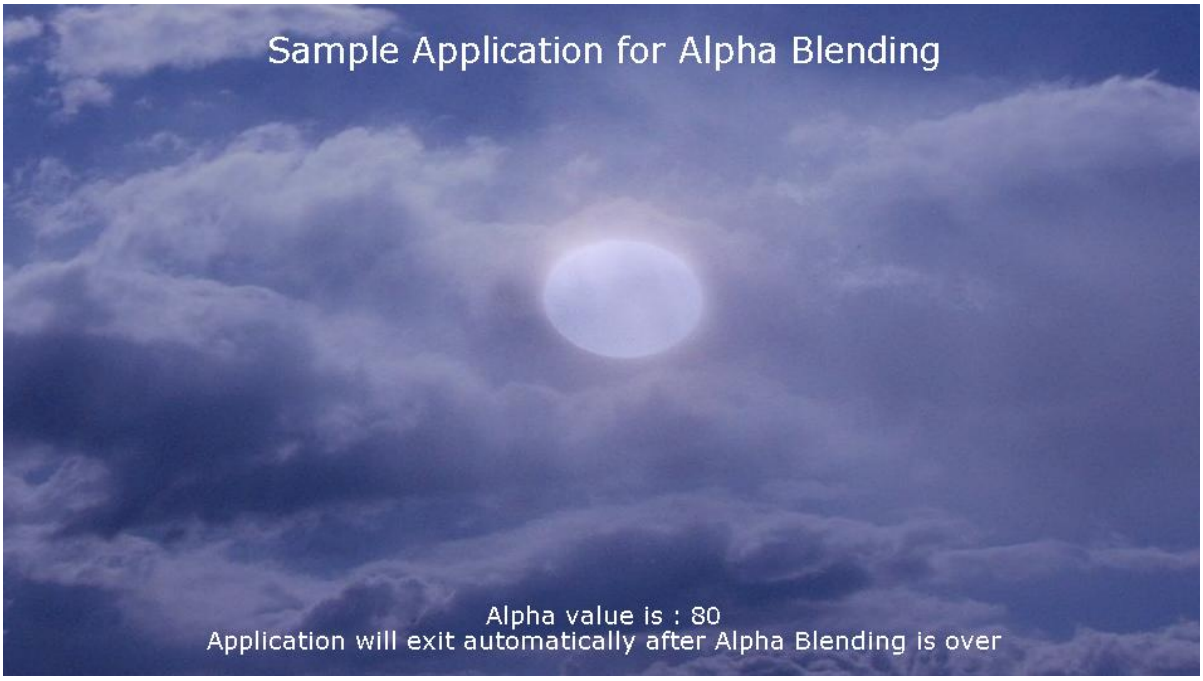
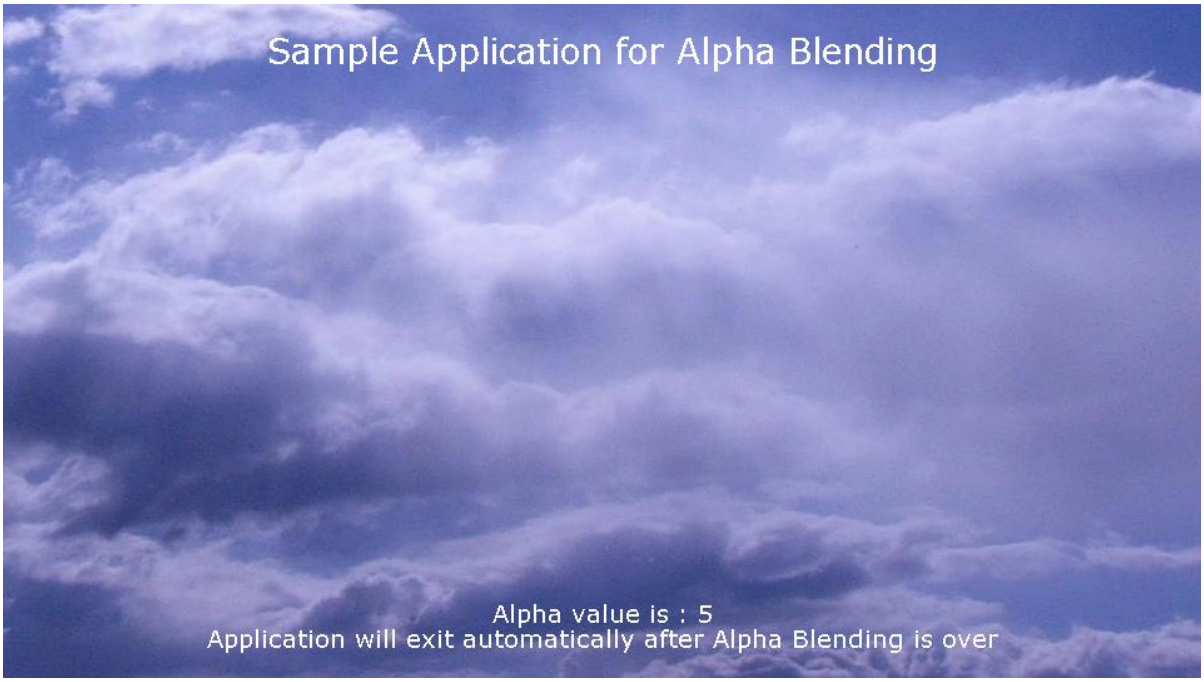


Figure: Alpha Blending – Intermediate stage2



And here you see the resultant alpha blended image

5.2.4. Remarks

None

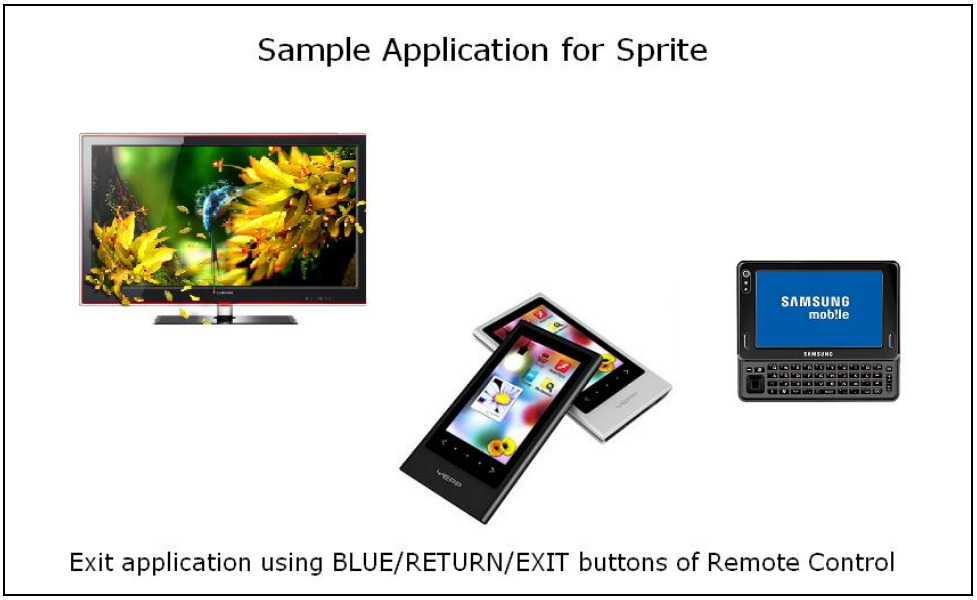
6. Sprite and Animation

6.1. Sprite

6.1.1. Introduction

A sprite is a two-dimensional image or animation that is integrated into a larger scene. Sprites were originally invented as a method of quickly compositing several images together in two-dimensional video games. Sprites are typically used for characters and other moving objects in video games. Let us understand the use of sprites in gaming applications.

Above is a sprite sheet, a collection of images is held in a single image file. It is useful when you have large number of images but don't want to deal with many image files. To render individual images separately you just need to clip the part you want and appropriately place it when blitting with screen.



The above output has been generated using a single sprite sheet.
Now that we know some basics, let's try doing it in code.

6.1.2. Steps

In previous chapters, we have learnt how to include appropriate headers, initialize SDL subsystem, and set the video mode. So we will proceed with displaying sprites.

1. To use True Type Font functionality of SDL, additionally include SDL_ttf header and library at the top of your program.

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
```

2. Using below assignment, programmers can use ***SDL_BlitSurface*** API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. As in all the previous examples, define all global variables/data (Screen height, width and the font size) that is used in the program. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by ***chResPath***. ***chResPath*** should be defined relative to ***resPath*** argument of ***Game_Main*** function.

```
SCREEN_WIDTH    = 960      -- Screen Width
SCREEN_HEIGHT = 540      -- Screen Height
chResPath = nil
event = nil
```

4. To start with, load the sprite sheet and background image. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by chResPath.

```
-- Load the sprite image
local temp_sprite = IMG_Load( chResPath.."samplesprite.bmp" )
sprite = SDL_DisplayFormat(temp_sprite)
if(nil == sprite) then
        print("Error:"..file.."Couldn't be opened:"..SDL_GetError())
end

-- Load the background image
local temp_background = IMG_Load( chResPath.."background.bmp" )

background = SDL_DisplayFormat(temp_background)

if( nil == background ) then
        print( "Error:"..file.."Couldn't be opened:"..SDL_GetError() )
end
```

5. Use color keying to hide the edges. Using color key is explained in the chapter on color Key but we will list down the code here for our reference.

```
-- Colour which needs to be transparent
COLORKEYr = 255
COLORKEYg = 255
COLORKEYb = 255

--Map the color key
color_key=SDL_MapRGB(sprite.format,COLORKEYr,COLORKEYg,
                COLORKEYb)

--Set the color key to make specified color transparent
if(-1 == SDL_SetColorKey(sprite, SDL_SRCCOLORKEY, color_key)) then
            print("Warning: colorkey will not be used, reason: "..SDL_GetError())
end
event    = SDL_Event_new()
```

6. Next we clip the desired image from the sprite sheet and position it on the screen. It is done inside the SpritesMove() function.

```
function SpritesMove()
            --Source and destination SDL_Rects
            srcRect1 = SDL_Rect_new()
            srcRect2 = SDL_Rect_new()
            srcRect3 = SDL_Rect_new()
            srcRect4 = SDL_Rect_new()

            destRect1 = SDL_Rect_new()
            destRect2 = SDL_Rect_new()
            destRect3 = SDL_Rect_new()
            destRect4 = SDL_Rect_new()
```

```
--Source SDL_Rects will copy different portions of sprite image.
srcRect1.x = 0
srcRect1.y = 0
srcRect1.w = 280
srcRect1.h = 175

srcRect2.x = 0
srcRect2.y = 190
srcRect2.w = 280
srcRect2.h = 225

srcRect3.x = 320
srcRect3.y = 25
srcRect3.w = 250
srcRect3.h = 150

srcRect4.x = 290
srcRect4.y = 180
srcRect4.w = 275
srcRect4.h = 250

--Destination SDL_Rects define the position of sprites on screen.
destRect1.x = X_Position
destRect1.y = math.random(50 , SCREEN_HEIGHT/2 )    -- random number
destRect1.w = 280
destRect1.h = 175

destRect2.x = X_Position + 280
destRect2.y = math.random(50 , SCREEN_HEIGHT/2 )
destRect2.w = 280
destRect2.h = 225
```

```
        destRect3.x = X_Position + 560
        destRect3.y = math.random(50 , SCREEN_HEIGHT/2 )
        destRect3.w = 250
        destRect3.h = 150

        destRect4.x = X_Position + 810
        destRect4.y = math.random(50 , SCREEN_HEIGHT/2 )
        destRect4.w = 275
        destRect4.h = 250

        SDL_BlitSurface(background, nil , screen, nil)
        SDL_BlitSurface(sprite, srcRect1 , screen, destRect1)
        SDL_BlitSurface(sprite, srcRect2 , screen, destRect2)
        SDL_BlitSurface(sprite, srcRect3 , screen, destRect3)
        SDL_BlitSurface(sprite, srcRect4 , screen, destRect4)

        X_Position = X_Position + 100
        if (X_Position > 500) then
                X_Position = -810
        end

        --Update screen
        SDL_Flip(screen)

end
```

6.1.3. Remarks

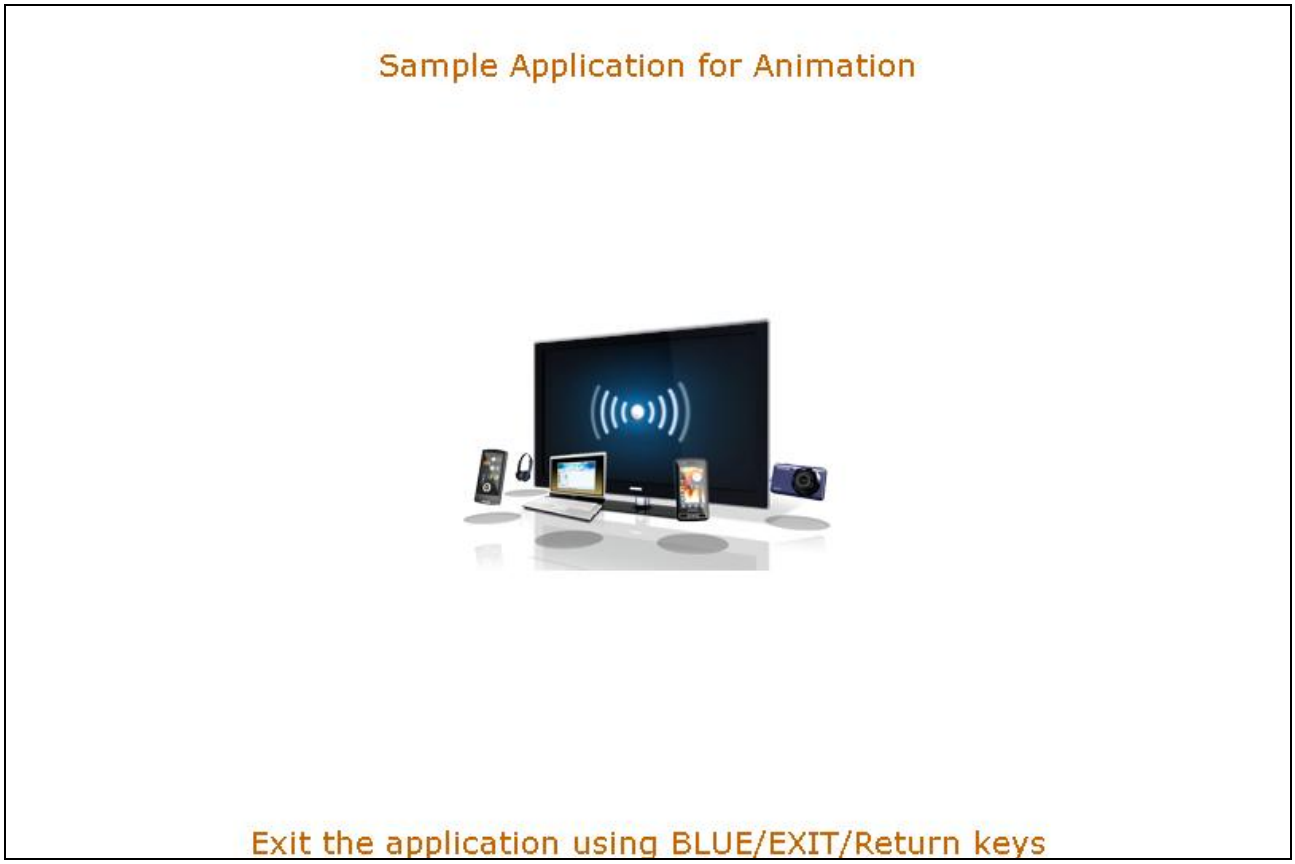
None

6.2. Animation

This sample application describes about the animation functionality on DTV.

6.2.1. Introduction

This LUA animation application highlights the steps involved to demonstrate animation functionality of SDL on screen using functionality of SDL-Lua.



6.2.2. Steps

1. Include SDL, SDL_image and SDL_ttf header and library with the current program as below:

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
```

2. Using below assignment, programmers can use SDL_BlitSurface API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. Declare all global variables/data that are used in the program. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by chResPath.

```
SCREEN_WIDTH = 960
SCREEN_HEIGHT = 540
chResPath = nil
event = nil
quit = 0
```

4. Initialize the SDL subsystem instance and set video mode surface with specified width, height and bits per pixel. Also verify the initialization of the SDL with proper error message.

```
chResPath = resPath.."/Res/"
-- Initialize SDL
if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) then
        print("Couldn't Initialize SDL: "..SDL_GetError())
        quit()
end
-- Set video mode
screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT,
        VIDEO_BPP,    SDL_HWSURFACE)
if ( nil == screen ) then
        print("Couldn't set video mode: "..SDL_GetError())
        quit()
end
```

5. Load the background image

```
background = IMG_Load( chResPath.."background.bmp" )
```

6. Then load the sprite i.e. load the bmp file and sets it colorkey

```
--Load the animation images
for i = 1, 25 do
    -- Load the image to be displayed
    image[i] = nil
    image[i] = IMG_Load( chResPath.."img_"..i..".bmp" )

    if( nil == image[i]) then
        print( "Unable to load image: "..SDL_GetError() )
        -- Close SDL
        SDL_Quit()
    end
```

```
    -- Colour which needs to be transparent
    COLORKEYr = 255
    COLORKEYg = 0
    COLORKEYb = 0
    -- Map color key
    color_key = SDL_MapRGB(image[i].format,COLORKEYr,COLORKEYg,
                                    COLORKEYb )
    -- Set color key to make specified color transparent
    if( -1 == SDL_SetColorKey( image[i], SDL_SRCCOLORKEY, color_key ) ) then
        print( "Warning: Colorkey not be used, reason: "..SDL_GetError() )
    end

    temp_image[i]= SDL_DisplayFormatAlpha(image[i])
    SDL_FreeSurface( image[i] )
end
```

7. Map a RGB color value to a pixel format. Maps the RGB color value to the specified pixel format and returns the pixel value as a 32-bit int. If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned. If the specified pixel format has an alpha component it will be returned as all 1 bits (fully opaque).

```
color_key=SDL_MapRGB(image[i].format,COLORKEYr,COLORKEYg,
                                    COLORKEYb )
```

8. Loop, blitting animation images and waiting for a RETURN or EXIT key events.

```
while( quit == 0 ) do
        for i = 1, 25 do
                --Draw the image on the screen
                 Draw( screen, temp_image[i] )
                 Loop()
                if (quit == 1) then
                        break
                end
        end
end
```

```
function Loop()
        while(SDL_PollEvent(event) ~= 0) do
                -- End the loop when Close or Escape has been done
                if( event.type == SDL_QUIT ) then
                        quit = 1
                elseif( event.key.keysym.sym == SDLK_ESCAPE ) then
                        quit = 1
                elseif( event.key.keysym.sym == SDLK_POWER ) then
                        quit = 1
                else
                        --do nothing
                end
        end
end
```

9. Draw() then blits frontImage and backImage (taking into consideration the variable alpha value for the front image) with the screen and renders the blended image to output screen.

```
function Draw( screen, image)
        ……
        -- Apply the background to screen
        SDL_BlitSurface( background, nil, screen, nil )

        -- Apply the image to screen
        SDL_BlitSurface( image, nil, screen, dest )
        …….
        --Update screen
        SDL_Flip( screen )
        SDL_Delay( 100 )
        ……
end
```

10. It is important to de-allocate the surfaces and quit the SDL sub-system before exiting from the application.

```
--Free the loaded image
for i = 1, 25 do
    SDL_FreeSurface( temp_image[i] )
end
SDL_FreeSurface( background)
SDL_FreeSurface( screen )
```

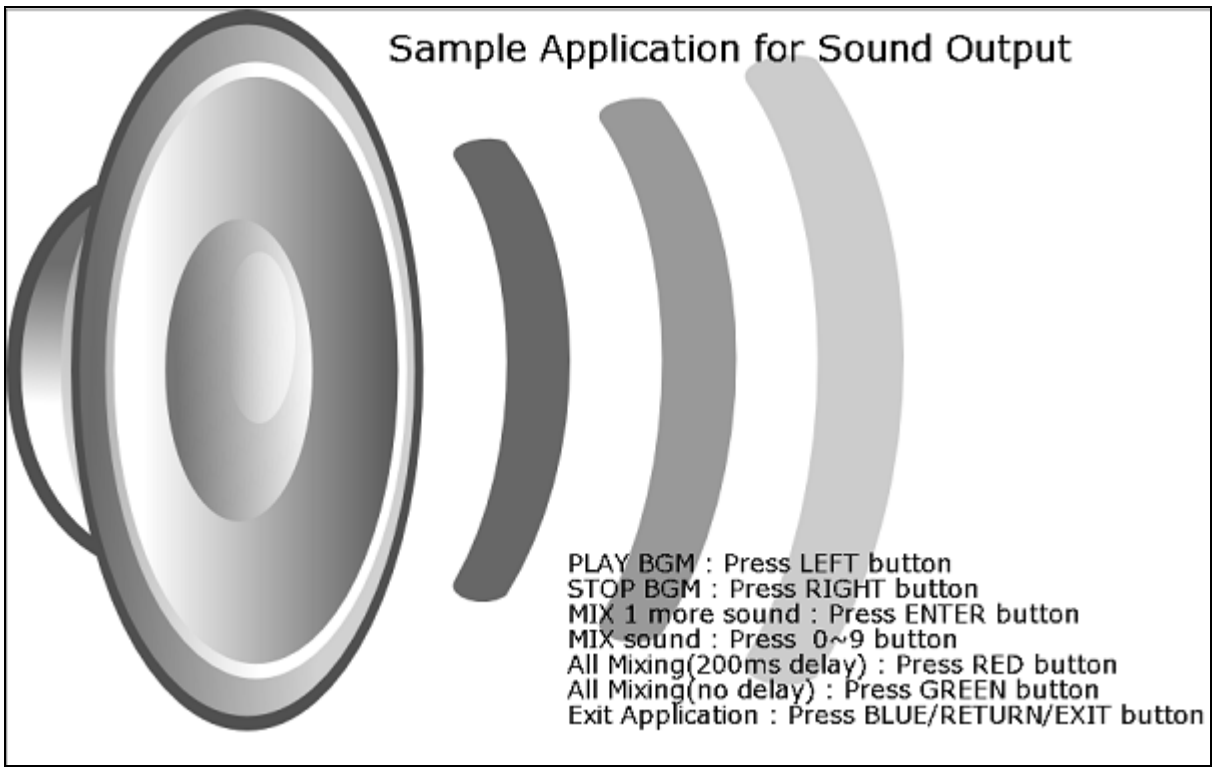
6.2.3. Remarks

Game Framework supports animation display formats RGB.

7. Sound Output

This sample application describes about the Sound functionality on DTV. In SDL sound there are two way to play the sound by using SDL_Music and SDL_Chunk.

SDL is support wav file play and internally audio operation is specified size chunk memory of raw data(same as wav file without header)



7.1. Introduction

This LUA Sound application highlights the steps involved to demonstrate Sound functionality of SDL on screen using functionality of SDL-Lua.

Samsung's support audio specification:-
SDL_AudioSpec
- freq (samples per second (Hz)) : 16000, 22050, 24000, 32000, 44100, 48000
- format (Audio Data Format) : AUDIO_S16
- channels (Number of channels : 1 mono, 2 : Stereo) : 2
Cf. 44100 Hz is CD Quality digital audio

What is Channel

Channel- Number of channels to allocate for mixing. A negative number will not do anything; it will tell you how many channels are currently allocated. Set the number of channels being mixed. This can be called multiple times, even with sounds playing. If Number channel is less than the current number of channels, then the higher channels will be stopped, freed, and therefore not mixed any longer. It's probably not a good idea to change the size 1000 times a second though. If any channels are deallocated, any callback set by Mix Channel Finished will be called when each channel is halted to be freed. Passing in zero WILL free all mixing channels, however music will still play. This API return the number of channels allocated. Never fails...but a high number of channels can seg-fault if you run out of memory.

What is Chunk

The internal format for an audio is chunk. This stores the sample data, the length in bytes of that data, and the volume to use when mixing the sample.

```
typedef struct Mix_Chunk
{
	int allocated;
	Uint8 *abuf;
	Uint32 alen;
	Uint8 volume;		-- Per-sample volume, 0-128
} Mix_Chunk;
```

allocated - a boolean indicating whether to free abuf when the chunk is freed.
0 if the memory was not allocated and thus not owned by this chunk.
1 if the memory was allocated and is thus owned by this chunk.
abuf - Pointer to the sample data, which is in the output format and sample rate.
alen - Length of abuf in bytes.
Volume - 0 = silent, 128 = max volume. This takes effect when mixing.

Note for developer:

When using SDL mixer functions developer should need to avoid the following functions from SDL:
SDL_OpenAudio - Use Mix_OpenAudio instead.
SDL_CloseAudio - Use Mix_CloseAudio instead.
SDL_PauseAudio - Use Mix_Pause(-1) and Mix_PauseMusic instead, to pause. Use Mix_Resume(-1) and Mix_ResumeMusic instead, to unpause.
SDL_LockAudio - This is just not needed since SDL mixer handles this for you.Using it may cause problems as well.
SDL_UnlockAudio - This is just not needed since SDL mixer handles this for you.Using it may cause problems as well.
SDL_AudioDriverName - This will still work as usual.
SDL_GetAudioStatus - This will still work, though it will likely return SDL AUDIO PLAYING even though SDL mixer is just playing silence.

7.2. Steps

1. In addition to the libraries and headers described in previous chapters, to use the sound functionality we need to include SDL_mixer library with the current program as below:

```
require( 'SDL')
require('SDL_image')
require('SDL_mixer')
require('SDL_ttf')
```

2. Using below assignment, programmers can use **SDL_BlitSurface** API. Declare the global variables.

```
SDL_BlitSurface = SDL_UpperBlit
SCREEN_WIDTH    = 960        -- Screen Width
SCREEN_HEIGHT = 540          -- Screen Height
FONT_SiZE = 18               -- Font Size
BPP = 32                     -- Bits per pixel
```

3. The resources for the game must be placed at a common location specified by ***chResPath***. The path chResPath can be defined relative to resPath input argument of ***Game_Main*** function.

```
chResPath = resPath.."/Res/"
ResourcePathSet(chResPath)
```

4. Inside the ResourcePathSet(chResPath) API set the path of the each and every resource those are used in this application.

```
function ResourcePathSet(chResPath)
        --Load the Data
        gARes_BGMFileName = gARes_BGMFileName .. chResPath
        gARes_BGMFileName = gARes_BGMFileName .. "play_SJ.wav"

        print("BGM File Name : ", gARes_BGMFileName)
        local nIndex
        for nIndex = 1 , 12 do
                gARes_ClipName[nIndex] = chResPath
        end
        gARes_ClipName[1] = gARes_ClipName[1] .. "Res/01-strike.wav"
        ……
        ……
        gARes_ClipName[12] = gARes_ClipName[12] .. "Res/12-perfect game.wav"

        for nIndex = 1 , 12 do
                print("Effect File Name : ", gARes_ClipName[nIndex])
        end
end
```

5. Initialize the SDL subsystem Instance. Also verify the Initialization of the SDL with proper error message.

```
--Initialize BOTH SDL video and SDL audio
if ( SDL_Init(bit_or( SDL_INIT_VIDEO , SDL_INIT_AUDIO )) < 0 ) then
        print( "Couldn't initialize SDL: ",SDL_GetError())
        return 1
end
```

6. Initialize the SDL_ttf library

```
-- Initialize TTF
if( -1 == TTF_Init() )then
        print( "Couldn't Initialize TTF: "..SDL_GetError() )
end
```

7. Load the font from the resource path to display.

```
font = LoadFont( chResPath.."verdana.TTF", FONT_SiZE)
if( nil == font ) then
        print( "Unable to load font:" ..SDL_GetError() )
        -- CleanUp
        CleanUp()
End

fontExit = LoadFont( chResPath.."verdana.TTF",(1.5 * FONT_SiZE) )
if( nil == fontExit ) then
        print( "Unable to load font:" ..SDL_GetError() )
        -- CleanUp
        CleanUp()
end
```

8. Loading a specific type of font to the library. Set the font size also.

```
function LoadFont( fontName, fontSize)
        loaded_font = TTF_OpenFont( fontname, fontSize )

        if( nil == loaded_font ) then
                print( "Error: "..fontname.."could not be opened:
                "..SDL_GetError() )
        end
        return loaded_font
end
```

9. Set the color of the font. And render the given text with the given font with fg color onto a new surface

```
-- Color for the font
color_st = SDL_Color_new()
color_st.r = 255
color_st.g = 255
color_st.b = 255
fontSurface = TTF_RenderText_Solid( font, "PLAY    BGM : TV
                Remotecontrol LEFT or RIGHT", color_st )
……….
……..
fontSurface6 = TTF_RenderText_Solid(fontExit, "Sample Application for Sound Output", color_st )
```

10. Now initialize the SDL_mixer Instance and allocate the channel to play the Music or Chunk. Define the Audio rate, audio format, audio channel and audio buffer.

```
audio_rate = 44100              --Frequency of audio playback
audio_format = AUDIO_S16        --Format of the audio we're playing
audio_channels = 2              --2 channels = stereo
audio_buffers = 4096            --Size of the audio buffers in memory

--Initialize SDL_mixer with our chosen audio settings
if  Mix_OpenAudio( audio_rate,audio_format,audio_channels,audio_buffers) ~= 0) then
        print("Unable to open audio\n")
        return 1
end
--Allocating channel to play the music
Mix_AllocateChannels(channel)
```

Note: - When already initialized, this function will not re-initialize SDL_mixer, nor fail. It will merely increment the number of times SDL_CloseAudio must be called to actually get it to uninitialized.

Syntax:
int Mix OpenAudio(int frequency, Uint16 format, int channels, int chunksize)

frequency - Output sampling frequency in samples per second (Hz).you might use MIX DEFAULT FREQUENCY(22050) since that is a good value for most games.
format - Output sample format.

channels - Number of sound channels in output.Set to 2 for stereo, 1 for mono. This has nothing to do with mixing channels.
chunksize - Bytes used per output sample.

SDL must be initialized with SDL INIT AUDIO before this call. Frequency would be 44100 for 44.1 KHz, which is CD audio rate. Most games use 22050, because 44100 require too much CPU power on older computers. chunksize is the size of each mixed sample. The smaller this is the more your hooks will be called. If make this too small on a slow system, sound may skip. If made to large, sound effects will lag behind the action more. You want a happy medium for your target computer. You also may make this 4096, or larger, if we are just playing music. MIX CHANNELS (8) mixing channels will be allocated by default. We may call this function multiple times; however we will have to call Mix_CloseAudio just as many times for the device to actually close. The format will not change on subsequent calls until fully closed. So we will have to close all the way before trying to open with different format parameters.format is based on SDL audio support.

Here are the values listed there:-
AUDIO U8 - Unsigned 8-bit samples
AUDIO S8 - Signed 8-bit samples
AUDIO U16LSB - Unsigned 16-bit samples, in little-endian byte order
AUDIO S16LSB - Signed 16-bit samples, in little-endian byte order
AUDIO U16MSB - Unsigned 16-bit samples, in big-endian byte order
AUDIO S16MSB - Signed 16-bit samples, in big-endian byte order

AUDIO U16 - same as AUDIO U16LSB (for backwards compatability probably)
AUDIO S16 - same as AUDIO S16LSB (for backwards compatability probably)
AUDIO U16SYS - Unsigned 16-bit samples, in system byte order
AUDIO S16SYS - Signed 16-bit samples, in system byte order
MIX DEFAULT FORMAT is the same as AUDIO S16SYS.

Note:
Syntax: int Mix AllocateChannels(int Numberchannel)
Numberchannel- Number of channels to allocate for mixing.A negative number will not do anything; it will tell you how many channels are currently allocated.

Set the number of channels being mixed. This can be called multiple times, even with sounds playing. If Numberchannel is less than the current number of channels, then the higher channels will be stopped, freed, and therefore not mixed any longer. It's probably not a good idea to change the size 1000 times a second though.If any channels are deallocated, any callback set by Mix_ChannelFinished will be called when each channel is halted to be freed. Passing in zero WILL free all mixing channels, however music will still play. This API return the number of channels allocated. Never fails...but a high number of channels can seg-fault if you run out of memory.

11. After allocating the channel load the WAV & MP3 file on the particular channel. Also verify its Error code.

12. As per the sample application pre load the Sound effect

```
for nIndex =1 , 12 do
        phaser[nIndex] = nil
        phaser[nIndex] = Mix_LoadWAV(gARes_ClipName[nIndex])
end
```

Example :- For SDL Chunk:

```
--Load our WAV file from disk
sound = Mix_LoadWAV("sound.wav")
if(nil == sound)then
        print("Unable to load WAV file:"..Mix_GetError())
end
```

Example :- For SDL Music:

```
--Load a sound.
mMusic = Mix_LoadMUS( "sound.mp3" )
if(nil == mMusic) then
        print("Error:".."Mix_LoadMUS(): failed " ..SDL_GetError())
        return 0
else
        print("Sucess:" .."Mix_LoadMUS(): success.")
end
```

Note: Mix_LoadWAV is a macro that creates an SDL_RWops for you then calls Mix_LoadWAV_RW to load a WAV from a file. It supports WAVE and MP3 files

13. After initialization of the sub-system Instance for video and audio mode surface. Set up a video mode with the specified width, height and bits-per-pixel.

```
screen = SDL_SetVideoMode(960, 540, 32, SDL_HWSURFACE)
if (nil == screen ) then
        print("Couldn't set video mode: ", SDL_GetError())
        return 2;
end
```

14. Now load a BMP file on the screen and display it on the allocated / initialized surface area

```
local imgPath = chResPath.." Sound1.png"
local temp_imgSurface = IMG_Load(imgPath)
local imgSurface = SDL_SDL_DisplayFormatAlpha (temp_imgSurface)
if(nil == imgSurface) then
        print("Not able to load the image\n")
        return 2
end
-- Load the background image
local temp_background = IMG_Load( chResPath.."background.bmp" )
background = SDL_DisplayFormat(temp_background)
if( nil == background ) then
        print( "Error:"..file.."Couldn't be opened:"..SDL_GetError() )
end
SDL_BlitSurface(imgSurface, nil, screen, nil)
SDL_Flip(screen)
```

IMG_Load(imgPath):- Loads a surface from a named Windows BMP file. When loading a 24-bit Windows BMP file, pixel data points are loaded as blue, green, red, and NOT red, green, blue (as one might expect).

SDL_BlitSurface(sprite, nil , screen, srcRect) :- This performs a fast blit from the source surface to the destination surface. The width and height in srcrect determine the size of the copied rectangle. Only the position is used in the dstrect (the width and height are ignored). Blits with negative dstrect coordinates will be clipped properly.

Note:- the SDL blitter does not have the capability of scaling the blitted surfaces up or down like it is the case with other more sophisticated blitting mechanisms. You have to figure something out for yourself if you want to scale images

SDL_Flip(sprite) :- Swap Screen buffer

Note:- On hardware that supports double-buffering, this function sets up a flip and returns. The hardware will wait for vertical retrace, and then swap video buffers before the next video surface blit or lock will return. On hardware that doesn't support double-buffering or if SDL_SWSURFACE was set, this is equivalent to calling SDL_UpdateRect(screen, 0, 0, 0, 0) . A software screen surface is also updated automatically when parts of a SDL window are redrawn, caused by overlapping windows or by restoring from an iconified state. As a result there is no proper double

buffer behavior in windowed mode for a software screen, in contrast to a full screen software mode.

15. Create event to handle the application state.

```
event = SDL_Event_new()
while done == 0 do
        while SDL_PollEvent(event) ~= 0 do
                if (event.type == SDL_QUIT) then
                        print("Call End..")
                        done =1
                end
                ……
                handleKey(event.key)
        end    -- while SDL_PollEvent(event) ~= 0 end
        SDL_Delay(50)
end    --while done == 0    end
```

Now handle I/O operation of the key event by using the API handleKey(event.key).

```
function    handleKey(key)
audioMixNum = 0
inputKey = key.keysym.sym
if ((inputKey == SDLK_0) or (inputKey == SDLK_1) or (inputKey == SDLK_2)
or (inputKey == SDLK_3) or(inputKey == SDLK_4) or (inputKey == SDLK_5)
or (inputKey == SDLK_6) or(inputKey == SDLK_7) or (inputKey == SDLK_8)
or (inputKey == SDLK_9) )then
        audioMixNum = inputKey - SDLK_0 + 1
        print("Mixing audio number :", audioMixNum)
if( key.type == SDL_KEYDOWN) then
        if (phaserChannel < 0 )then
                for i = 1 , audioMixNum    do
                        phaserChannel = Mix_PlayChannel(-1, phaser[i], -1)
                print(i , "-th audio Mix :", gARes_ClipName[i])
                                        SDL_Delay(200)
                end
        end
        phaserChannel = -1
        Mix_HaltChannel(phaserChannel)
        end
end

        if (inputKey == SDLK_HOME)    then -- Remote RED
                ……
                phaserChannel = -1
                Mix_HaltChannel(phaserChannel)
        end
end

        ……
        music = Mix_LoadMUS(gARes_BGMFileName)
        Mix_PlayMusic(music, -1)
        Mix_HookMusicFinished("musicDone")
        end
end
```

16. Play the load file into the buffer and then Check the status of a specific channel

- channel
  - Channel to test whether it is playing or not.
  - -1 will tell you how many channels are playing.
  - Does not check if the channel has been paused.

-Mix_HaltChannel(phaserChannel) will stop the channel specified in channel. If -1 is passed as the parameter to channel, all channels will be stopped .

-Mix_PlayMusic(music, -1) This is an opaque data type used for Music data. This should always be used as a pointer. It supports loading the following types: WAV, MOD, MID, OGG, and MP3.

-Mix_HookMusic() and Mix_HookMusicFinished() these two APIs is used as a call back function call and end.

-Example :- For playing the Chunk of music

17. Use of SDL_Delay() for few second of delay to listen the chunk of music

```
--Play our sound file, and capture the channel on which it is played
        channel = Mix_PlayChannel(-1, sound, 0)
        if(channel == -1) then
                print("Unable to play WAV file: "..Mix_GetError())
        end
        SDL_Delay(100)

--Wait until the sound has stopped playing
while(Mix_Playing(channel) == 0)do
        print(".")
        SDL_Delay(100)
        break
end
```

-Example:- Playing Mix_Music
-Use of SDL_Delay() for few second of delay to listen the MixMusic

```
--Attempt to play the sound.
        channel = Mix_PlayMusic(mMusic,-1)
        if(-1 == channel) then
                print("Error:" ..filename .."Mix_PlayChannel(): failed
                " ..SDL_GetError())
        else
                print("Sucess:".."Mix_PlayChannel(): success.")
        end
        while(0 == Mix_Playing(channel) ) do
                print("Sucess:" .."Mix_PlayChannel(): Playing")
                SDL_Delay(100)
                break
        end
```

Note:- Mix_PlayChannel(-1, sound, 0) will play the specified chunk over the specified channel. SDL_mixer will choose a channel for you if you pass -1 for channel.The chunk will be looped loops times, the total number of times played will be loops+1. Passing -1 will loop the chunk infinitely.

18. For rendering the information text / Image.

```
        fontRect = SDL_Rect_new()
        fontRect.x = 0
        fontRect.y = 400
        SDL_BlitSurface(imgSurface, nil, screen, nil)
        SDL_BlitSurface(fontSurface, nil, screen, fontRect)
        fontRect.y = fontRect.y + 15
        SDL_BlitSurface(fontSurface1, nil, screen, fontRect)
        ……
        SDL_Flip(screen)
```

19. Close the Music file and release the allocated buffer.

```
function musicDone()
        Mix_HaltMusic()
        ResourcePathSet(chResPath)
        Mix_FreeMusic(music)
        music = nil
end
```

20. Release the chunk from buffer

```
--Release the memory allocated to our sound
        Mix_FreeChunk(sound)
```

21. De-allocate the memory for the surface and close the channel as well the application.

```
function CleanUp()
        print("Audio test end. Bye bye !!\n")
        -- Close the Audio
        Mix_CloseAudio()
        -- Release the event instance
        SDL_Event_delete(event)
        -- Release the imgSurface
        SDL_FreeSurface(imgSurface)
        SDL_FreeSurface(screen)
        --Need to make sure that SDL surface have a chance to clean up
        SDL_Quit()
end
```

22. We can re-use this application‘s snippet to use to play a music file as per the music format is mentioned above discussion.

7.3. Remarks

Game Framework supports Sound formats in WAV formats.

8. True Type Font

This sample application describes about the True Type Font functionality on DTV.

True type fonts are provided by several open source libraries and they are widely used in commercial games. SDL provides one extension library SDL_ttf for true type font functionality.

SDL_ttf is a TrueType font rendering library that is used with the SDL library, and almost as portable. It depends on freetype2 to handle the TrueType font data. It allows a programmer to use multiple TrueType fonts without having to code a font rendering routine themselves. With the power of outline fonts and antialiasing, high quality text output can be obtained without much effort.

FreeType2 is a software font engine that is designed to be small, efficient, highly customizable, and portable while capable of producing high-quality output (glyph images). It can be used in graphics libraries, display servers, font conversion tools, text image generation tools, and many other products as well.

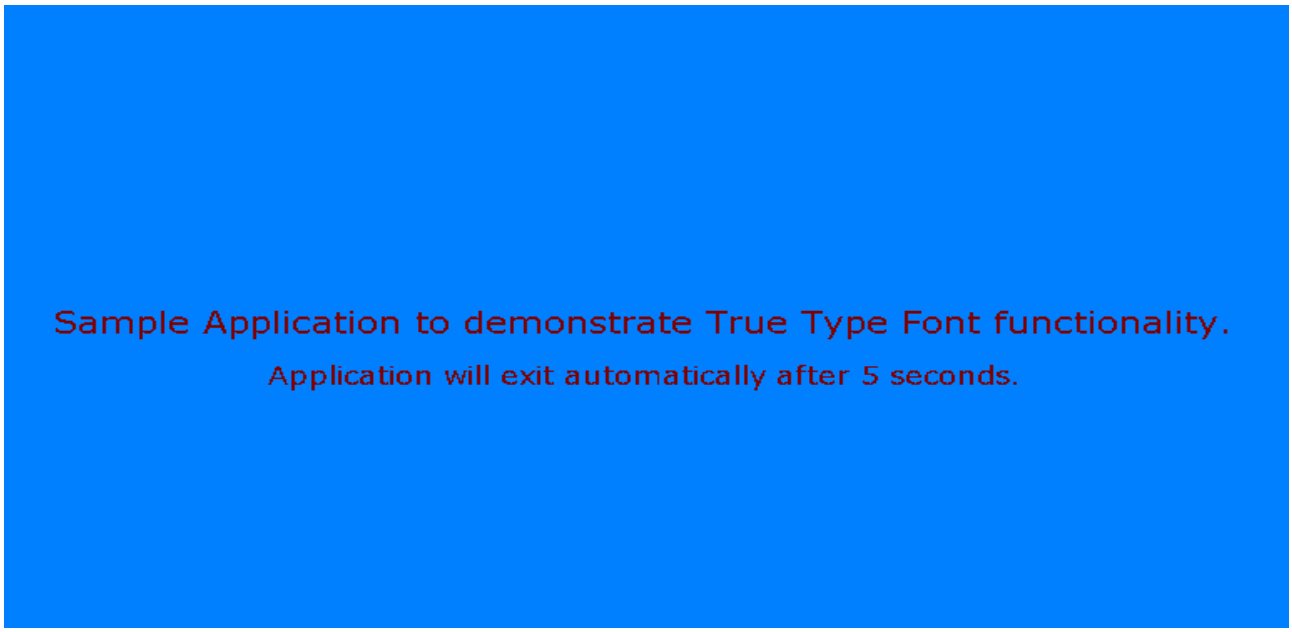
FreeType2 is a font service and doesn't provide APIs to perform higher-level features like text layout or graphics processing (e.g., colored text rendering, ‘hollowing’, etc.). However, it greatly simplifies these tasks by providing a simple, easy to use, and uniform interface to access the content of font files.
By default, FreeType 2 supports the following font formats.

- TrueType fonts (and collections)
- Type 1 fonts
- CID-keyed Type 1 fonts
- CFF fonts
- OpenType fonts (both TrueType and CFF variants)
- SFNT-based bitmap fonts
- X11 PCF fonts
- Windows FNT fonts
- BDF fonts (including anti-aliased ones)
- PFR fonts

- Type 42 fonts (limited support)

8.1. Introduction

True Type Fonts offers game application developers a high degree of control over precisely how their text fonts are displayed, right down to particular pixels, at various font sizes. This sample application highlights the steps involved to display text messages(“Sample Application to demonstrate True Type Font functionality” and “Application will exit automatically after 5 seconds”) on screen using True Type Font functionality of Samsung Game Framework.



8.2. Steps

1. To use True Type Font functionality of SDL, additionally include SDL_ttf header and library at the top of your program.

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
```

2. Using below assignment, programmers can use **SDL_BlitSurface** API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. As in all the previous examples, define all global variables/data (Screen height, width and the font size) that is used in the program. Every game would have audio, video, images etc resources. These resources must be placed at a common location specified by ***chResPath***. ***chResPath*** should be defined relative to ***resPath*** argument of ***Game_Main*** function.

```
SCREEN_WIDTH    = 960      -- Screen Width
SCREEN_HEIGHT = 540        -- Screen Height
FONT_SiZE = 20             -- Font Size
chResPath = nil
```

4. Initialize SDL sub-system and set video mode with specified width, height and bits per pixel. It is good to ensure correct initialization through API return codes.

```
--Initialize SDL
if(-1 == SDL_Init(SDL_INIT_VIDEO)) then
        print("Couldn't Initialize SDL: "..SDL_GetError())
end

--Set up screen
screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32,
                                SDL_HWSURFACE)
if(nil == screen) then
        print("Couldn't Set Video Mode: "..SDL_GetError())
end
```

5. Initialize SDL_ttf to use true type fonts using ***TTF_Init***() API.

```
--Initialize TTF
if(-1 == TTF_Init()then
        print("Couldn't Initialize TTF: "..SDL_GetError())
end
```

6. Load the image to be displayed on screen.

```
--Load the image to be displayed
local temp_image = IMG_Load( chResPath.."background.bmp" )
image = SDL_DisplayFormat(temp_image)
if(-1 == image) then
        print("Unable to load image: "..SDL_GetError())

        --CleanUp
        CleanUp()
end
```

7. Load the font style. In this application, it is verdana font style.

```
--Load the font to be displayed
font = LoadFont(chResPath.."verdana.TTF")
if(nil == font) then
        print("Unable to load font:" ..SDL_GetError())
        CleanUp()
end
-- Load the font to be displayed for Help
fontExit = LoadFont( chResPath.."verdana.TTF", (0.9 * FONT_SiZE) )
if( nil == fontExit ) then
        .....
        .....
end
```

8. The definition of LoadFont () function is below:

```
function LoadFont( fontName, fontSize)

        --Open the font
        loaded_font = TTF_OpenFont ( fontName, fontSize)
        if(nil == loaded_font) then
                print("Error: "..fontname.."could not be opened: "..SDL_GetError())
        end
        return loaded_font
end
```

9. Set the font color. In this application, it is black.

```
--Color for the font
color = SDL_Color_new()
color.r =128
color.g = 0
color.b = 0
```

10. Write the text message that is to be displayed using **TTF_RenderText_Solid**() API.

```
--Render the font text to surface
fontSurface = TTF_RenderText_Solid(font, "Sample Application to demonstrate
                True Type Font functionality.", color)
fontSurface1 = TTF_RenderText_Solid( fontExit, "Application will exit
automatically after 5 seconds.", color )
```

11. The draw() function encapsulates the instructions to blit the text surface and image to screen.

```
--Draw the font and image on screen
Draw()
```

- Below is the definition of Draw():

```
function Draw()
        --Apply the image to screen
        SDL_BlitSurface(image, nil, screen, nil)

        -- Apply the font to the screen for the headline
        posHeadline = SDL_Rect_new()
        posHeadline.x = ( SCREEN_WIDTH - fontSurface.w )/2
        posHeadline.y = ( SCREEN_HEIGHT - fontSurface.h )/2

        -- Apply the font to the screen
        SDL_BlitSurface( fontSurface, nil, screen, posHeadline )

        posHeadline.x = ( SCREEN_WIDTH - fontSurface1.w )/2
        posHeadline.y = posHeadline.y + (2 * FONT_SiZE)

        SDL_BlitSurface( fontSurface1, nil, screen, posHeadline )

        --Update screen
        SDL_Flip(screen)

        --Pause to display the image for longer duration
        SDL_Delay(5000)
end
```

12. After the text and image is displayed, it is important to free allocated surfaces and close TTF and SDL sub-systems. It is done inside CleanUp() function.

```
--CleanUp
CleanUp()
```

13. De-allocate all the memories used by the instance of image,fontSurface,fontSurface1 and font. Finally quite(deallocate) the SDL subsystem TTF and SDL by using TTF_Quit() and SDL_Quit().

```
function CleanUp()

        --Free the surfaces
        SDL_FreeSurface(image)
        SDL_FreeSurface(fontSurface)
        SDL_FreeSurface( fontSurface1 )

        --Close the font
          if(nil ~= font) then
                TTF_CloseFont( font )
                font = nil
          end
        --Close the SDL_ttf
        TTF_Quit()

        --Close the SDL
        SDL_Quit()
```

8.3. Remarks

None

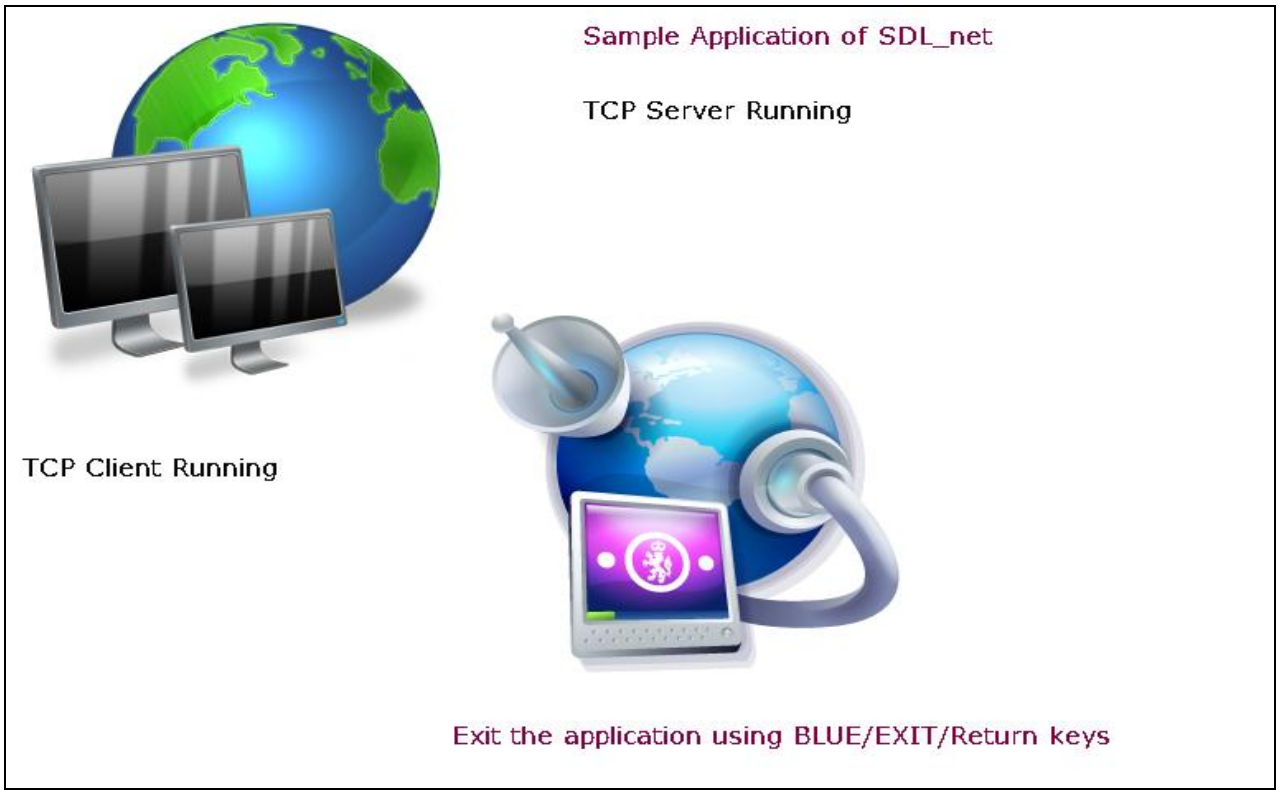
9. How to use Network (SDL_net)

This sample application describes about the SDL_net functionality on DTV. Both the client and server communication is in SDL-LUA standard format.

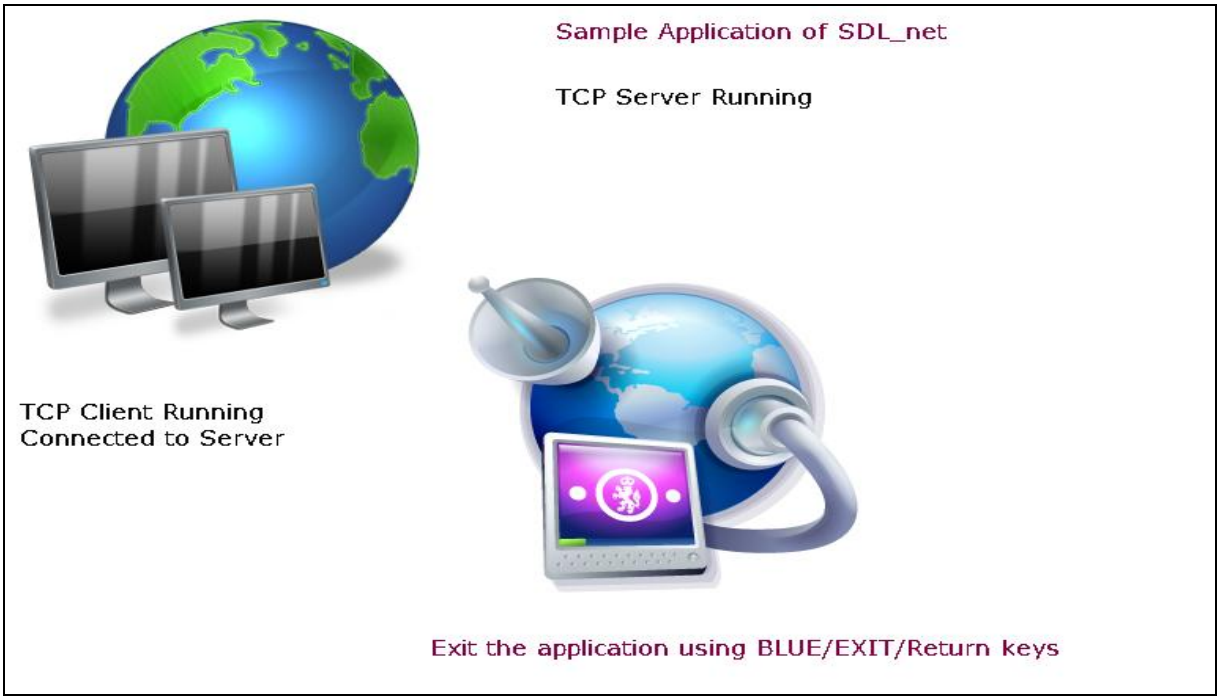
9.1. Introduction

This LUA SDL_net application highlights the steps involved to demonstrate client–server communication functionality of SDL on screen using functionality of SDL-Lua. The three way hand shaking of TCP client and 4 way hands shaking of TCP communication closing is done properly. And on pressing 'BLUE / EXIT / RETURN Keys' the application will close.

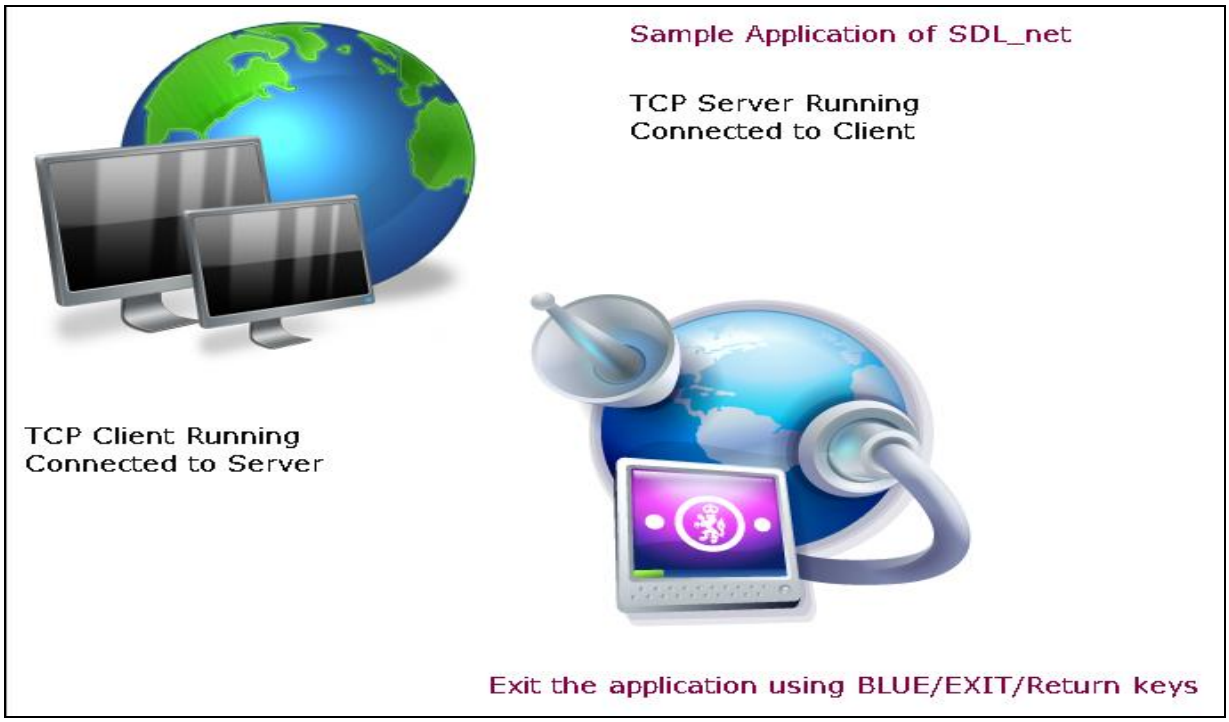
- Client and Server both are running(both are running in different thread)
- ServerThread is running before the Client Thread



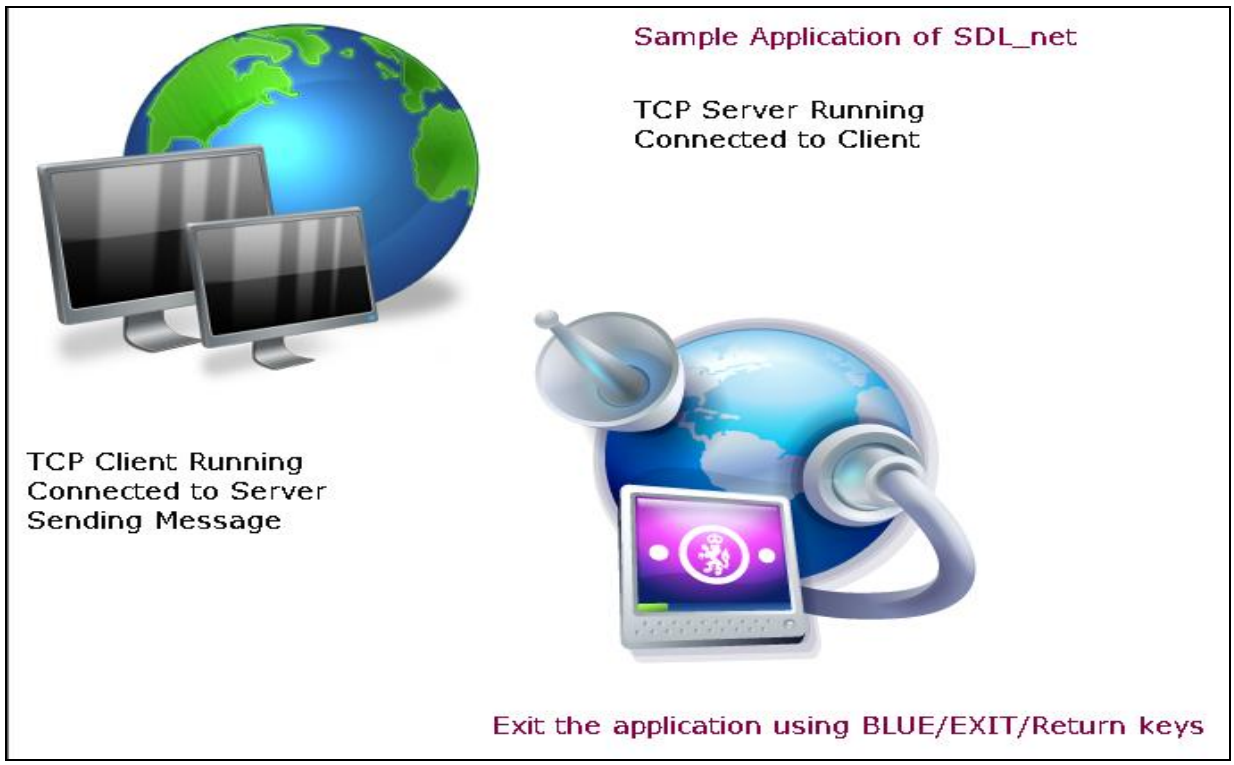
- Client side connectivity – Client send request for communication establishment.



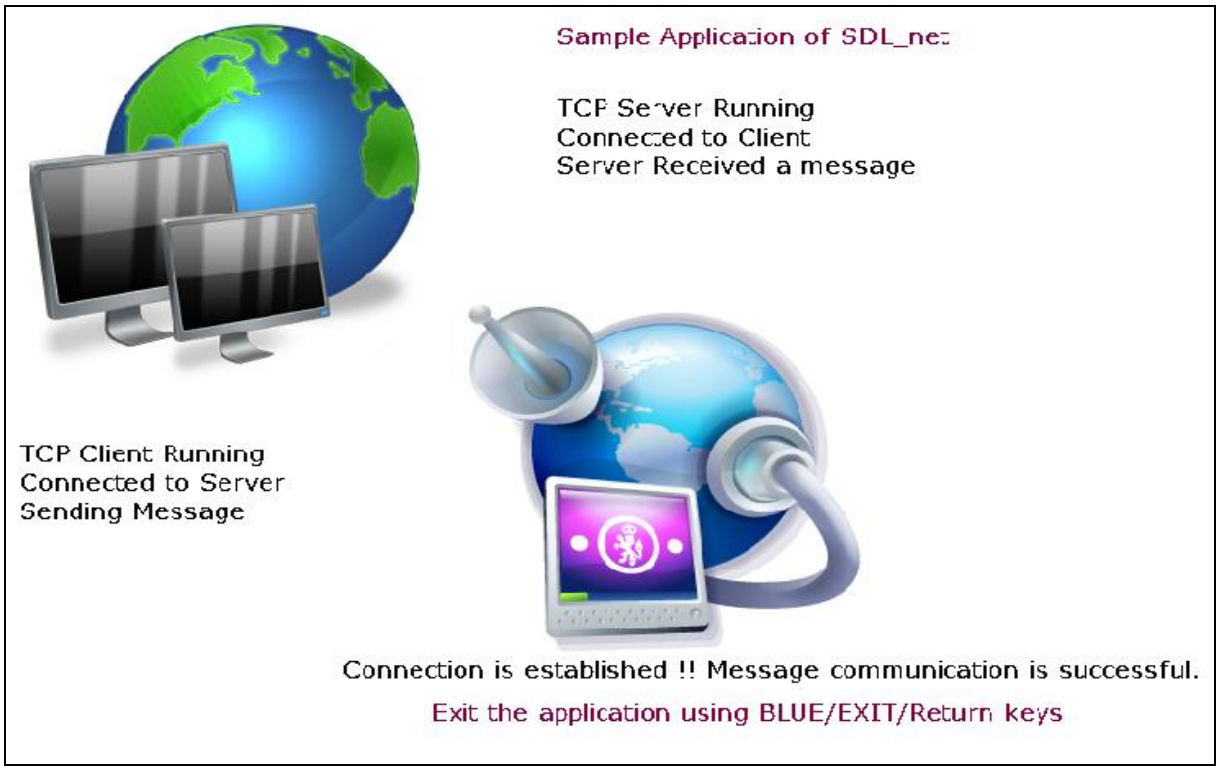
- Conection between Client and Server Thread are done properly(3-way handshaking is done)



-Client Thread sending a message to the Server Thread.



- Server Thread getting the message.



If client is get disconnected /not getting any host name. So, Before running the Client / Server application ensure the IP and port number are correct. On the top of the screen the sample application header("Sample Application of SDL_net") and at the end of the screen a message ("Exit the application using BLUE/EXIT/Return keys") is showing to gudie the user for quit from the application.

9.2. Steps – TCP Communication for both Client and Server

1. These steps are common among Server and Client applications.
2. To use TCPServer /TCPClient functionality of SDL, include SDL_net header and library with the current program as below:

```
require('SDL')
require('SDL_net')
require('SDL_image')
require('SDL_ttf')
```

3. Global variables are declared

```
SCREEN_WIDTH    = 960      -- Screen Width
SCREEN_HEIGHT = 540      -- Screen Height
VIDEO_BPP    = 32                -- Bits per pixel
VIDEO_WIDTH = SCREEN_WIDTH    -- Video Width
VIDEO_HEIGHT = SCREEN_HEIGHT    -- Video Height
local gMessage1 = nil    --SDL Surface for displaying message
local gMessage2 = nil    --SDL Surface for displaying message
local gTextColor = nil --SDL_Color for displaying text with perticular colour
local gTextColorKey = nil --SDL_Color for displaying text with perticular colour
local pScreen = nil          --SDL surface representing for screen
local gFont = nil            -- a pointer to the Font as a TTF_Font
TestSDLnet = {}           --Class SDL Net
myEvent = nil
```

4. Game_Main() api is the entry point to the application. In which the resource path is set for getting the resource at the run time of the application.

```
function Game_Main(resPath)
        UpdatePackagePath(resPath)
        gchResPath = resPath
        chResPath = resPath.."/Res/"
        TestSDLnet:setUp()
        TestSDLnet:test_SDL_Net_tcp()
        TestSDLnet:tearDown()
end
```

5. Initialize the SDL subsystem Instance and also verify the Initialization of the SDL with proper error message.This must be called before using other functions in this library.SDL , SDL_net and SDL_ttf must be initialized before this call. It returns 0 on success, -1 on error. And also initialize set the video mode.

```
function TestSDLnet:setUp()
        if(SDLNet_Init()==-1) then
                print("SDLNet_Init: ",SDLNet_GetError())
                return err
        end
        if(SDL_Init(SDL_INIT_VIDEO)==-1) then
                print("Error:" ..filename .."Unable to initialize SDL:
        " ..SDL_GetError())
                return 1
        end
        if( -1 == TTF_Init() )then
                print( "Couldn't Initialize TTF: "..SDL_GetError() )
        end

end
```

- TestSDLnet:test_SDL_Net_tcp() In this api, initialize the SDL sub-system and create two threads, waits for hand shake mechanism, finally kills both the thread and showing the appropriate message in between the client server communication. Key event is handled in this api for exit of the application.

```
function TestSDLnet:test_SDL_Net_tcp()
        --Initialize the SDL sub sytem and variables
        init()
        SDL_FillRect( pScreen, nil, SDL_MapRGB( pScreen.format, 255, 255,   255 ) )
        imgFileName = chResPath.."network_connected.png"
        ShowImage(10,10,250,250,imgFileName)    -- Update a Rectangle on Screen

        imgFileName = chResPath.."network_connection.png"
        ShowImage(290,200,250,250,imgFileName) -- Update a Rectangle on       Screen
        gMessage1 = TTF_RenderText_Solid( gFont, "Sample Application of SDL_net",
                        gTextColorKey )
        ShowFont(350,10,gMessage1)

        gMessage1 = TTF_RenderText_Solid( gFont, "Exit the application using
                        BLUE/EXIT/Return keys", gTextColorKey )
        ShowFont(270,480,gMessage1)
        ……..
        ……
        --Create threads
        local str1 = charp_to_voidp("#1")
        local thread1 = SDL_CreateThread("tcpServerThread",str1)
        SDL_Delay(1000)
        local str2 = charp_to_voidp("#2")
        local thread2 = SDL_CreateThread("tcpClientThread",str2)
```

```
        SDL_Delay(2000)
        SDL_KillThread(thread2)
        thread2 = nil
        SDL_KillThread(thread1)
        thread1 = nil
        KeyEvent()
        --De-allocate the resource allocated for the application
        CleanUp()
end
```

6. In init() api, the variables, color instance, SDL_TTF and SDL sub system are initialized. **TTF_OpenFont** () - Load the font from the resource path to display and set the font size also.

```
--- initialises basic functions
function init()
        gMessage1 = nil
        gMessage2 = nil
        pScreen =SDL_SetVideoMode( VIDEO_WIDTH, VIDEO_HEIGHT,
                                    VIDEO_BPP, SDL_HWSURFACE)
        gFont = TTF_OpenFont( chResPath.."verdana.ttf", 16 )
          if( nil == gFont) then
                print( "Error: "..fontname.."could not be opened:
                "..SDL_GetError() )
        end

        gTextColor = SDL_Color_new()
        ……..
        gTextColorKey = SDL_Color_new()
        ……..
end
```

7. Display the loaded the font on the screen.

```
local function ShowFont(x,y,gMessage1)
	local offset = SDL_Rect_new()
	offset.x=x
	offset.y = y

	local textColor = SDL_Color_new()
	textColor.r= 10
	textColor.g= 10
	textColor.b= 10

	SDL_BlitSurface( gMessage1, nil, pScreen, offset )
	SDL_Flip(pScreen)

	if offset ~= nil then
		SDL_Rect_delete(offset)
		offset = nil
	end
	if textColor ~= nil then
		SDL_Color_delete(textColor)
		textColor = nil
	end
end
```

8. Displays the background image on the screen. This api is taking the parameter as height,width, x-coordinate , y- coordinate and image file name.

```
local function ShowImage(x,y,w,h,imgFileName)

        local bmpSurface = IMG_Load(imgFileName)
        local dst = SDL_Rect_new()
        ……….
        SDL_BlitSurface(bmpSurface, nil, pScreen, dst)
        SDL_Flip(pScreen)

        if dst ~= nil then
                SDL_Rect_delete(dst)
                dst = nil
        end

        if (bmpSurface ~= nil) then
                SDL_FreeSurface(bmpSurface)
                bmpSurface = nil
        end
end
```

9. For handling the key events and the api for quite from the application.

```
local function KeyEvent()
        ……
        gQuit = 1
        while gQuit == 1 do
                while SDL_PollEvent(myEvent) ~= 0 do
                        if myEvent.type == SDL_QUIT then
                                print ("Bye bye ..")
                                gQuit = 0
                                break
```

```
        elseif myEvent.type == SDL_KEYDOWN then
                print("key pressed in Show\n")
                if    myEvent.key.keysym.sym == SDL_QUIT or
                      myEvent.key.keysym.sym == SDLK_POWER or
                      myEvent.key.keysym.sym ==SDLK_ESCAPE then
                        print ("Bye bye in the Show")
                        gQuit = 0
                        break
                end
            end
      end
end

        return 0
end
```

-TCPServer Thread - After Initializing SDL_net Instance properly, now get the IP and PORT number for communication.

```
function tcpServerThread(data1)
        print(" TCP Server Running ")
        ……..
        ……..
        local port = 3366
        local m = nil
        local msg = nil
        msg = tovoid("char", 1024)

        ……..
        …….
        local ip = nil
        ip = IPaddress_new()

        if(SDLNet_ResolveHost(ip, 0,port)==-1) then
                print("\nSDLNet_ResolveHost: ",SDLNet_GetError())
                return err
        end
        server=SDLNet_TCP_Open(ip)
        if(server == nil) then
                print("\nSDLNet_TCP_Open: ",SDLNet_GetError())
                return err
        end
```

```
local ret = true
while ret == true do
        client=SDLNet_TCP_Accept(server)
        if client ~= nil then
                remoteip=SDLNet_TCP_GetPeerAddress(client)
                if (remoteip ~= nil ) then
                ipaddr=SDL_SwapBE32(remoteip.host)
                print("peer ip address = ", ipaddr)
                --read the buffer from client
                len=SDLNet_TCP_Recv(client,msg,1024)
                if(len ~= 0) then
                        m = voidp_to_charp(msg)
                      print("Server Received a Message:
                      ",len.."message = ", m)
                      SDLNet_TCP_Close(client)
                      client = nil
                      ret = false
                end
        ……..
        if msg ~= nil then
                deletemem(msg)
                msg = nil
        end

        if ip ~= nil then
                IPaddress_delete(ip)
                ip = nil
        end

        if server ~= nil then
                SDLNet_TCP_Close(server)
                server= nil
        end
end
```

Please note:- SDLNet_ResolveHost resolve the string host, and fill in the IPaddress pointed to by address with the resolved IP and the port number passed in through port. This is the best way to fill in the IPaddress struct for later use. This function does not actually open any sockets, it is used to prepare the arguments for the socket opening functions. This function will put the host and port into Network Byte Order into the address fields, so make sure you pass in the data in your hosts byte order. (Normally not an issue) .It returns 0 on success. -1 on errors, plus address. Host will be INADDR_NONE. An error would likely be that the address could not be resolved.

- Connect to the host and port using a TCP connection. Now open the server socket. Connect to the host and port contained in ip using a TCP connection. If the host is INADDR ANY, then only the port number is used, and a socket is created that can be used to later accept incoming TCP connections.

ip - This points to the IPaddress that contains the resolved IP address and port number to use.

10. For accepting a socket from Client request for communication. Accept an incoming connection on the server TCPsocket, accept a connection by using SDLNet_TCP_Accept().

Client = **SDLNet_TCP_Accept(**server**)**

- Server -This is the server TCPsocket which was previously created by SDLNet_TCP_Open.
- Accept an incoming connection on the server TCPsocket. Do not use this function on a connected socket. Server sockets are never connected to a remote host. What you get back is a new TCPsocket that is connected to the remote host. This is a non-blocking call, so if no connections are there to be accepted, you will get a nil TCPsocket and the program will continue going.
- It returns a valid TCPsocket on success, which indicates a successful connection, has been established. nil is returned on errors, such as when it's not able to create a socket, or it cannot finish connecting to the originating host and port. There also may not be a

connection attempt in progress, so of course you cannot accept anything, and you get a nil in this case as well.

- SDLNet_TCP_GetPeerAddress -- get the address of the remote end of a socket. It returns the address of the peer's end of the socket. This contains the peer's IP address and the TCP port number of that end of the connection
- SDLNet_TCP_Recv - read the buffer from client. SDLNet_TCP_Recv gets available data that has been received on the socket. Up to maxlen bytes of data is read and stored in the location referenced by data. If no data has been received on the socket, this routine waits (blocks) until some comes in. The actual number of bytes copied into the location may be less than maxlen bytes. A non-blocking way of using this function is to check the socket with SDLNet_CheckSockets and SDLNet_SocketReady and call SDLNet_TCP_Recv only if the socket is active. Because of the behavior of TCP, a block of data sent by the peer will usually not be read at one time with SDLNet_TCP_Recv. TCP is a streaming protocol. This means that data will always be delivered in order, but not necessarily in the same chunks that it was sent. Applications that communicate with lines of text may need to call this routine several times and check for a newline character.
- SDLNet_TCP_Send - sends the data over the socket. If the data cannot be sent immediately, the routine waits until all of the data may be delivered properly (it is a blocking operation). This routine is not used for server sockets.

11. TCPClient Thread - Get the server’s port number for sending request for establishment of a communication.

```
function tcpClientThread(data2)
        print(" TCP Client Running ")
        ipadrs = IPaddress_new()
        ……..
        --declare the port number
        port = 3366
        local ip_address = "127.0.0.1"

        -- Resolve the argument into an IPaddress type
        errRet = SDLNet_ResolveHost(ipadrs,ip_address,port)

        -- open the server socket
        local tcpsock = SDLNet_TCP_Open(ipadrs)

        message = "Hi this is Cient"
        print("Message to be send by Client:", message)
        local len_send = string.len(message)

        msg_send = charp_to_voidp(message)
        if len_send ~= nil then
                local result
                print("Client Sending: "," message length =
                "..len_send,"message = "..message)
                result = SDLNet_TCP_Send(tcpsock,msg_send,len_send) --
                        add 1 for the NULL
        end
        if ipadrs ~= nil then
                IPaddress_delete(ipadrs)
                ipadrs = nil
        end
        if tcpsock ~= nil then
                SDLNet_TCP_Close(tcpsock)
                tcpsock = nil
        end
end
```

- Please note:- SDLNet_ResolveHost resolve the string host, and fill in the IPaddress pointed to by address with the resolved IP and the port number passed in through port. This is the best way to fill in the IPaddress struct for later use. This function does not actually open any sockets, it is used to prepare the arguments for the socket opening functions. This function will put the host and port into Network Byte Order into the address fields, so make sure you pass in the data in your hosts' byte order. (Normally not an issue) .It returns 0 on success. -1 on errors, plus address. Host will be INADDR_NONE. An error would likely be that the address could not be resolved.

12. Connect to the host and port using a TCP connection. Connect to the host and port contained in ip using a TCP connection. If the host is INADDR ANY, then only the port number is used, and a socket is created that can be used to later accept incoming TCP connections.

13. Now Client can send and receiving the message to the Server by using the below Apis.

- SDLNet_TCP_Recv - read the buffer from client. SDLNet_TCP_Recv gets available data that has been received on the socket. Up to maxlen bytes of data is read and stored in the location referenced by data. If no data has been received on the socket, this routine waits (blocks) until some comes in. The actual number of bytes copied into the location may be less than maxlen bytes. A non-blocking way of using this function is to check the socket with SDLNet_CheckSockets and SDLNet_SocketReady and call SDLNet_TCP_Recv only if the socket is active. Because of the behavior of TCP, a block of data sent by the peer will usually not be read at one time with SDLNet_TCP_Recv. TCP is a streaming protocol. This means that data will always be delivered in order, but not necessarily in the same chunks that it was sent. Applications that communicate with lines of text may need to call this routine several times and check for a newline character.

- SDLNet_TCP_Send - sends the data over the socket. If the data cannot be sent immediately, the routine waits until all of the data may be delivered properly (it is a blocking operation). This routine is not used for server sockets.

14. On closing / quit of application follow the below steps- call the tearDown() api. SDLNet_Quit shutdown and cleanup the network API. After calling this all sockets are

closed, and the SDL_net functions should not be used. Use SDLNet_Init to use the functionality again.

```
function TestSDLnet:tearDown()
        SDL_Event_delete(myEvent)
        SDLNet_Quit()
        print( 'TestSDLnet:tearDown' )
        TTF_Quit()
        SDL_Quit()
        pScreen = nil
end
```

- In CleanUp() api deallocate all the allocated memory.

```
local function CleanUp()
        if gTextColor ~= nil then
                SDL_Color_delete(gTextColor)
                gTextColor = nil
        end
        if gTextColorKey ~= nil then
                SDL_Color_delete(gTextColorKey)
                gTextColorKey = nil
        end
        if gFont ~= nil then
                TTF_CloseFont( gFont )
                gFont = nil
        end
        if gMessage1 ~= nil then
                SDL_FreeSurface( gMessage1 )
                gMessage1 = nil
        end
        if gMessage2 ~= nil then
                SDL_FreeSurface( gMessage2 )
                gMessage2 = nil
        end
              pScreen = nil
        end
```

9.3. Remarks

The server and Client are two different applications. Run server first to bind the port and then run Client to connect to the same port on which server was bind for Game Framework.

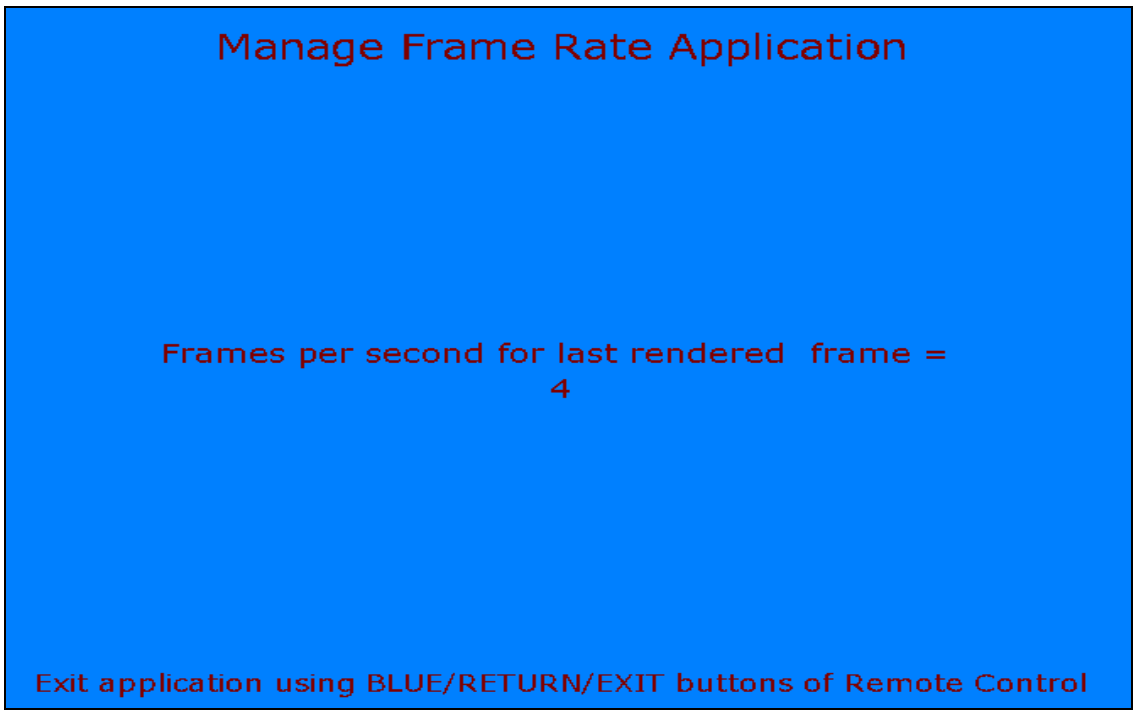
10. Manage Frame Rate

The aim of this chapter is to make an understanding of how to manage frame rate of a gr aphics application so that application is good for human eye visualization.

10.1. Introduction

In a graphics application once a frame is rendered then frame buffer is swapped to render the other frame. In this way, though an application looks continuously displayed but in actual it is not continuous. The rendering of frames are so fast that it appears to be continuous. The human eye cannot differentiate if frames are rendered fast enough.

The number of frames rendered per second (fps) is called frame rate. If the frame rate is too less in an application rendering then application will look jittered and if it is too fast then it will not be good enough for visualization for a human eye. So this presents the need of managing frame rate in a range which is good for visualization. In this chapter we will learn how write an application for this purpose.



10.2.Steps

1. As you have learnt in previous chapters that to use the Game Framework and to use the SDL functionality we need to set various header files, global variables, and initialize the SDL subsystem. Also we need to specify the path for the all game resources.

2. To manage the frame rate we will need to have several variables needed for counting the number of frames and time before start of a frame and after rendering the frame.

```
start_time = 0
last_time = 0    -- Time after rendering the specified number of frames
diff_time = 0 – Difference time
```

3. As with all previous applications, the application entry point for this application also is Game_Main() function.

```
function Game_Main(resPath)
        LoadGameContents(resPath)
        chResPath = resPath.."/Res/"
…
end

function LoadGameContents(dirPath)

        package.path = package.path..";"..dirPath.."/?.lua;"
end
```

4. As in previous applications, at start of the application to use the SDL functionality and creating the screen surface needed for rendering we will initialize the SDL subsystem Instance and set video mode surface with specified width, height and bits per pixel. Also verify the Initialization of the SDL with proper error message. Also we will load true type font for displaying frame rate on the surface. In this application, we will go for verdana font.

5. As discussed in chapter of image loading, to load the background image we will use SDL_Image module. IMG_Load API will be used to load the specified image. Input parameter to the API is the image file which needs to be loaded and it returns the loaded image.

```
--Load the image
local tempImage = IMG_Load(chResPath.."background.bmp")
if ( nil == tempImage ) then
        print( "Unable to load image:" ..SDL_GetError() )
        CleanUp()
end

-- Set Display Format
image = SDL_DisplayFormat(tempImage)
if ( nil == image ) then
        print( "Couldn't Set Display Format:" ..SDL_GetError() )
        CleanUp()
end

-- Map RGB components
SDL_FillRect( screen, screen.clip_rect,
                SDL_MapRGB( screen.format, 0xFF, 0xFF, 0xFF ) )
```

6. Now, as evident in the application snapshot in Introduction chapter, we are displaying application information with the fps value. We render text messages like “Manage Frame Rate Application” onto the display surface using **TTF_RenderText_Solid** API.

```
-- Render the font text to surface
fontSurface1 = TTF_RenderText_Solid( font, "Manage Frame Rate Application",
                color )
fontSurface2 = TTF_RenderText_Solid( font1, "Frames per second for last rendered
                frame = ", color )
fontSurface3 = TTF_RenderText_Solid( font2, "Exit application using
                BLUE/RETURN/EXIT buttons of Remote Control", color )
```

7. So we have created the basic framework of running our application and now here comes the actual part of this chapter i.e. managing frame rate. We will create a function regulateFPS() to manage the frame rate and display the FPS value after managing frame rate.

```
function regulateFPS()
        if( flag == 0    ) then       -- Initialize timer
                start_time = SDL_GetTicks()
                flag = 1
        else            -- Regulate the graphical rendering by delay
                last_time = SDL_GetTicks()
                diff_time = last_time - start_time
                if ( diff_time < (1000 / FRAMES_PER_SECOND) ) then
                        SDL_Delay( (1000 / FRAMES_PER_SECOND) -
                                        diff_time )
                        last_time = SDL_GetTicks()
                end
                diff_time = (last_time - start_time)
                fps = (1000/diff_time)
                dec = (fps%1)
                str = fps - dec
                fontSurfaceFPS = TTF_RenderText_Solid( font1, str, color )
                start_time = last_time
        end
end
```

8. If we look at this function deeply then we notice that when called first time this function will set starting time and increment the frame counter. In all subsequent calls function will find out one later time value and calculate the difference with the start time. FRAMES_PER_SECOND is specified FPS value which needs to be maintained (for this application we have kept the value as 5). So, to manage that frame rate we will use SDL_Delay API and the application will be delayed for the duration of time which is short for running the application at specified frame rate. So that application runs at the specified FPS value. The running FPS value will be calculated. The calculated FPS value

will be converted to a surface which can be rendered at the screen using TTF_RenderText_Solid API.

9. After having the background image, the application information and the renderable FPS surface. So in the Draw function, which will be called for each frame, we will render all these surfaces using SDL_BlitSurface API. Appropriate positions have been calculated for all the text items. Draw function will be called in the Game loop. We have already learnt the game loop in Game procedure chapter. Same mechanism will be used here.

```
function Draw()

    -- Apply the image to the screen
    SDL_BlitSurface( image, nil, screen, nil )

    -- Apply the font to the screen for the headline
    posHeadline1 = SDL_Rect_new()
    posHeadline1.x = ( SCREEN_WIDTH - fontSurface1.w )/2
    posHeadline1.y = FONT_SiZE

    SDL_BlitSurface( fontSurface1, nil, screen, posHeadline1 )

    -- Apply the font to the screen for the headline
    posHeadline2 = SDL_Rect_new()
    posHeadline2.x = ( SCREEN_WIDTH - fontSurface2.w )/2
    posHeadline2.y = ( SCREEN_HEIGHT - fontSurface2.h )/2 -
                                    fontSurfaceFPS.h + 20

    SDL_BlitSurface( fontSurface2, nil, screen, posHeadline2 )
```

```
        -- Set the position and render to display FPS
        position = SDL_Rect_new()
        position.x = ( SCREEN_WIDTH - fontSurfaceFPS.w )/2
        position.y = ( SCREEN_HEIGHT - fontSurfaceFPS.h )/2 + 20

        SDL_BlitSurface( fontSurfaceFPS, nil, screen, position )

        -- Apply the font to the screen for the help
        posHeadline3 = SDL_Rect_new()
        posHeadline3.x = ( SCREEN_WIDTH - fontSurface3.w )/2
        posHeadline3.y = ( SCREEN_HEIGHT - fontSurface3.h ) - FONT_SiZE

        SDL_BlitSurface( fontSurface3, nil, screen, posHeadline3 )
        -- Update screen
        SDL_Flip( screen )

end
```

10. In the last, we will need to free all the resources as we have done in all previous chapters by calling CleanUp() Api. The frame rate will be displayed in the application window. So, we were able to manage the frame rate.

```
-- CleanUp
CleanUp()
```

11. In the CleanUp() api , De-allocate allthe allocated memory in SDL.

```
function CleanUp()

        -- Free the loaded image
        SDL_FreeSurface( image )

        -- Free the SDL surfaces
        SDL_FreeSurface( fontSurface1 )
        SDL_FreeSurface( fontSurface2 )
        SDL_FreeSurface( fontSurface3 )
        SDL_FreeSurface( fontSurfaceFPS )

        --Close the font
        TTF_CloseFont( font )
        TTF_CloseFont( font1 )
        TTF_CloseFont( font2 )

        -- Quit SDL_ttf
        TTF_Quit()

        -- Quit SDL
        SDL_Quit()

end
```

10.3.Remarks

None.

11. Collision Detection (CD)

In physical simulations, video games and computational geometry, collision detection involves algorithms for checking for collision, i.e. intersection, of two given solids. Simulating what happens once a collision is detected is sometimes referred to as "collision response" (see collision response, physics engine and ragdoll physics). Collision detection algorithms are a basic component of 2D and 3D video games. Without them, characters could go through walls and other obstacles.

In a 2D game, collision detection can be done using two methods – Collision Detection by Bounding Box and Collision Detection by Pixel.

If an object is perfectly rectangular then Collision Detection by bounding box method is an optimal method, but in real world of gaming everything is not rectangular so in this case collision detection by bounding box is not an accurate methodology.

We will go through these methods one by one.

11.1. CD by Bounding Box

This sample application describes about the collision by bounding box functionality on DTV.

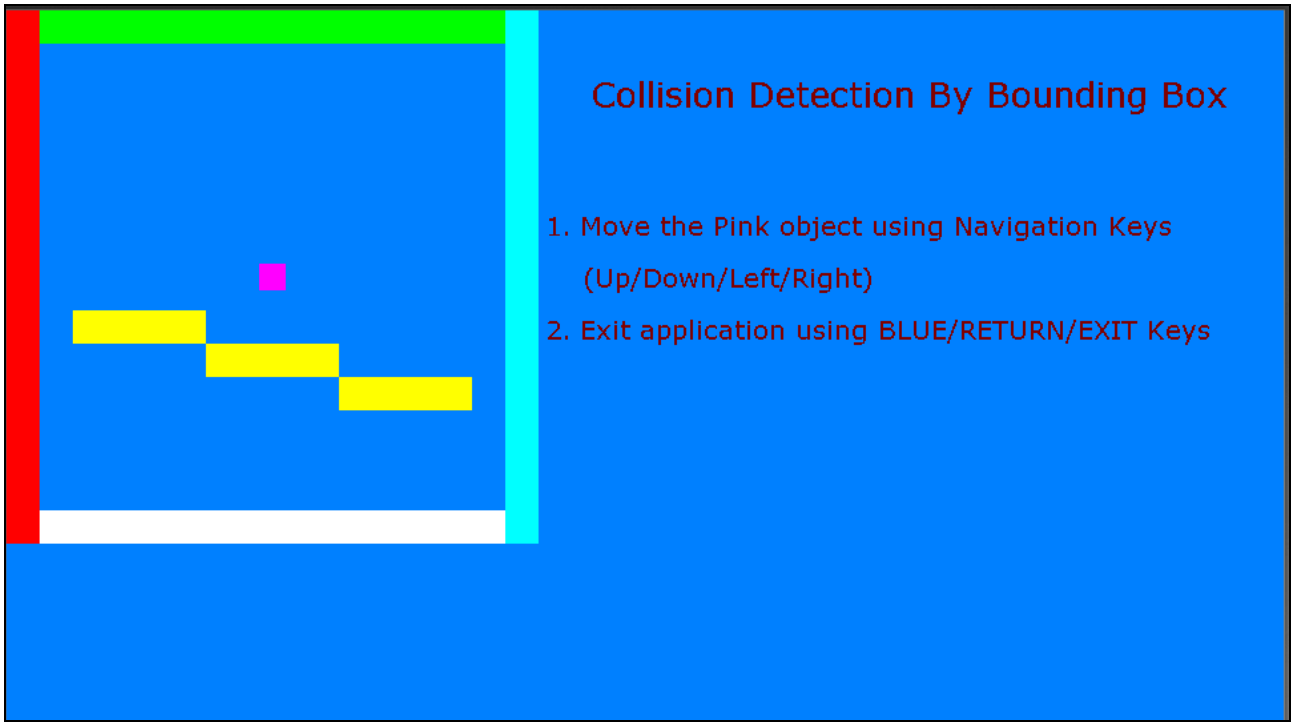
11.1.1. Introduction

If two objects are sticking through each other, and they are both closed poly-hedra, then at least one face of one object will be penetrating at least one face of the other object. The detection of this case is known as Collision detection.

When the square hits the edge of the screen you need to push the square back onto the screen (like with the player) and negate the velocity in the appropriate direction. If it hits the left or right edges of the bounding on the screen the negate x velocity, and similarly for the top or bottom you negate the v velocity.

Collision detection in game application is carried out into two steps:

(1) Determine which pairs of shapes need to be tested for collision (broad phase)
(2) Determine collision results for each pair identified in step (1) (narrow phase)



In N, step (1) is implemented using a uniform "loose" grid of square cells; each shape is stored in the cell which contains its center, and each shape is collided against any shapes in its current cell, or the 8 cells touching the current cell.

Later tutorials will explain this system in greater detail; this tutorial will explain how step (2) was implemented in N.

The algorithms we used to handle this step are of use to any game which requires fast collision detection that provides more than a simple Boolean result.

Physical simulators differ in the way they react on a collision. Some use the softness of the material to calculate a force, which will resolve the collision in the following time steps like it is in reality. Due to the low softness of some materials this is very CPU intensive. Some simulators estimate the time of collision by linear interpolation, roll back the simulation, and calculate the collision by the more abstract methods of conservation laws.

Some iterate the linear interpolation (Newton's method) to calculate the time of collision with a much higher precision than the rest of the simulation. Collision detection utilizes time coherence to allow even finer time steps without much increasing CPU demand, such as in air traffic control.

The obvious approaches to collision detection for multiple objects are very slow. Checking every object against every other object will, of course, work, but is too inefficient to be used when the number of objects is at all large. Checking objects with complex geometry against each other in the obvious way, by checking each face against each other face is itself quite slow. Thus, considerable research has been applied to speeding up the problem.

11.1.2. Steps

Collision detection by bounding box is explained below

1. We begin with including appropriate SDL headers and libraries at the start of the program. Only after that one can use SDL APIs to build the game application.

   We start by including the desired includes, Dlls, shared libs, Lua files. Here, 'SDL' is the Lua wrapper on SDL file. Then we define all global variables/data that are used in the program.

   Note: Developers must use SDL_UpperBlit() to use SDL_BlitSurface() API.
   Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by chResPath. The path chResPath should be defined relative to resPath argument of Game_Main function

```
require('SDL')
require('SDL_image')
require('SDL_ttf')

SDL_BlitSurface = SDL_UpperBlit

SCREEN_WIDTH = 960          -- Screen Width
SCREEN_HEIGHT = 540         -- Screen Height
UNIT_DISTANCE = 5           -- Unit of movement
NUM_WALL_UNITS = 7          -- Number of units constituting the wall
event = nil
```

Note: Game_Main(resPath) is the main execution function which this guide uses in all the chapters.

2. Set the font color used to render on the screen.

```
color = SDL_Color_new() -- Font color
color.r = 128
color.g = 0
color.b = 0
```

3. We now define the execution function that encapsulates the program logic:

```
function Game_Main(resPath)
        LoadGameContents(resPath)
        chResPath = resPath.."/Res/"
…
end

function LoadGameContents(dirPath)

        package.path = package.path..";"..dirPath.."/?.lua;"
end
```

4. Initialize SDL sub-system and set the video mode with specified width, height and bits per pixel.
   It is good to ensure correct initialization through API return codes.

```
-- Initialize the SDL
 if (0 ~= SDL_Init(SDL_INIT_VIDEO))then
        print("Error:".."Unable to initialize SDL: " ..SDL_GetError())
        return 1
end

--Initialize TTF
if ( -1 == TTF_Init() ) then
        print( "Couldn't Initialize TTF: " ..SDL_GetError() )
end

-- Set up screen
Screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32,
          SDL_HWSURFACE)
 if (nil == screen)then
        print("Error:Unable to set video mode:" ..SDL_GetError())
        cleanup()
        return 1
 end
```

5. Load the image that is to be displayed on screen. The BMP image can be loaded using SDL_LoadBMP() API.

```
-- Load the background image to be displayed
image = LoadImage(chResPath.."background.bmp")
if (nil == image)then
        print("Error:".."Unable to load image: " ..SDL_GetError())
        cleanup()
        return 1
end
```

6. In LoadImage() API as per filename specified (.BMP).
   Also create an optimized image of the current image.

```
function LoadImage(filename)

    -- Load the image
    local tempImage = IMG_Load(filename)

    if(nil == tempImage)then
            print("Error:".."could not be opened the Image: " ..IMG_GetError())
        return nil
    end

    -- Set Display Format
    loaded_image = SDL_DisplayFormat(tempImage)
    if ( nil == loaded_image ) then
            print( "Couldn't Set Display Format:" ..SDL_GetError() )
            return nil
    end

    return loaded_image
end
```

7. Initialize the wall (bounding box) for the Image. And also initialize the Rect bounding box.

```
--Initialize the Wall and Moving Rectangular Box
initWall()
```

8. In the initWall() API initialize some of wall/ bounding for the imageInside the initRectBox() API Initialize the Rect:

```
function initRectBox()
        MovingObject = SDL_Rect_new()
        MovingObject.x = 190
        MovingObject.y = 190
        MovingObject.w = 20
        MovingObject.h = 20
end
```

```
function initWall()

        WallUnits = {}
        for i=1, NUM_WALL_UNITS do
                WallUnits[i] = SDL_Rect_new()
        end
        ……
        ……
        ……
        WallUnits[4].x = 25;
        WallUnits[4].y = 375;
        WallUnits[4].w = 350;
        WallUnits[4].h = 25;
        ……
        ……
end
```

9. Initialise the TTF font to be loaded.

```
-- Load the font to be displayed
font = LoadFont(chResPath.."verdana.TTF", 27 )
if ( nil == font ) then
        print( "Unable to load font:" ..SDL_GetError() )
        CleanUp()
end
-- Render the font text to surface
fontSurface1 = TTF_RenderText_Solid( font, "Collision Detection By Bounding
                Box", color )
TTF_CloseFont(font)

font = LoadFont(chResPath.."verdana.TTF", 20 )
if ( nil == font ) then
        print( "Unable to load font:" ..SDL_GetError() )
        CleanUp()
end

fontSurface2 = TTF_RenderText_Solid( font, "1. Move the Pink object using
                Navigation Keys", color)

fontSurface3 = TTF_RenderText_Solid( font, " (Up/Down/Left/Right)", color )

fontSurface4 = TTF_RenderText_Solid(font,"2. Exit application using
                BLUE/RETURN/EXIT Keys ", color )
```

10. Inside the game application loop draw the image on the screen after creating a new event. Destroy the event afterwards.

```
event = SDL_Event_new()
-- Game Loop: Draw the image on the screen
while(1 == loop()) do
        Draw()
end
SDL_Event_delete(event)
```

11. Loop() API is for taking the use event , and as per the event Update the moving object position as per the user input

```
function loop()
        -- Pressed arrow key status
        if( SDL_PollEvent(event)    ~= 0 ) then

        --End the loop when Close or Escape has been done
        if((event.type == SDL_QUIT) or
        (event.key.keysym.sym == SDLK_ESCAPE))then
                return 0 –Exit
        -- if any arrow key is pressed
        elseif(event.type == SDL_KEYDOWN)then
                if(event.key.keysym.sym == SDLK_LEFT)then
                        left = true
                        ……
        --if any pressed arrow key is released
        elseif(event.type == SDL_KEYUP)then
                if(event.key.keysym.sym == SDLK_LEFT)then
                        left = false
                        ……
                else
                        print("No matched event.key.keysym.sym")
                end
        end

        -- Update the moving object position as per the user input.
        moveRectBox(left, right, up, down)
    end
        return 1 – Continue
end
```

12. Now move the rectangle box inside the bounding by using the API moveRectBox(left, right, up, down)

```
function moveRectBox(left, right, up, down)

	if(true == left) then
		MovingObject.x = (MovingObject.x) - UNIT_DISTANCE
		if(true == check_collision()) then
			MovingObject.x = (MovingObject.x) + UNIT_DISTANCE
		end

	elseif(true == right) then
		MovingObject.x = (MovingObject.x) + UNIT_DISTANCE
		if(true == check_collision()) then
			MovingObject.x = (MovingObject.x) - UNIT_DISTANCE
		end

	elseif(true == up) then
		MovingObject.y = (MovingObject.y) - UNIT_DISTANCE
		if(true == check_collision())then
			MovingObject.y = (MovingObject.y) + UNIT_DISTANCE
		end

	elseif(true == down) then
		MovingObject.y = (MovingObject.y) + UNIT_DISTANCE;
		if(true == check_collision())then
			MovingObject.y = (MovingObject.y) - UNIT_DISTANCE
		end
	end
end
```

13. When the object is moving around the bounding box check its collision with other pixels. check if collision occurs between walls and moving object.

```
function check_collision ()
        for i=1, NUM_WALL_UNITS do
                if(true == (isCollision(WallUnits[i], MovingObject))) then
                        return true
                end
        end
        return false
end
```

14. To check if a collision occurs between two SDL_Rect objects. In this API the detection of collision is occurring. The API isCollision(RectA, RectB) returns true if collision is detected or false if not.

```
function isCollision( RectA, RectB)
        leftA = RectA.x
        rightA = leftA + RectA.w
        topA = RectA.y
        bottomA = topA + RectA.h

        leftB = RectB.x
        rightB = leftB + RectB.w
        topB = RectB.y
        bottomB = topB + RectB.h

        if( ( (rightA > leftB) and (leftA < rightB) ) and ( (bottomA > topB) and (topA
           < bottomB) ) ) then
                return true
        end
        return false
end
```

15. In Draw API Show the wall / bounding, the Rect and Blit the loaded image with screen using SDL_BlitSurface() API. SDL_BlitSurface() performs a fast blit from the source surface to the destination surface. To display the image, the source surface is image and the destination surface is screen.

```
function Draw()

    -- Apply the background image to the screen
    SDL_BlitSurface(image, nil, screen, nil)

    -- Draw wall and rectangular moving object
    showWall()
    showRectBox()

    --Update screen
    SDL_Flip(screen)

end
```

16. To draw the different walls on the screen.

```
function showWall()
        -- Blit the each of wall units here
        SDL_FillRect(screen, WallUnits[1], SDL_MapRGB(screen.format,255,0, 0 ))
        ……
        ……
        SDL_FillRect(screen, WallUnits[7], SDL_MapRGB(screen.format, 255,
        255, 0))

end
```

17. To draw the moving object on screen

```
function showRectBox()
        SDL_FillRect(screen, MovingObject, SDL_MapRGB( screen.format, 255,
        0, 255))
end
```

18. To render the image on screen, we call SDL_Flip() API. For the displayed image to be perceived by viewer, it must stay on the screen for some time. SDL_Delay() API pauses the program execution so that the image is visible for this time period.

```
-- Update Screen
if(-1 == SDL_Flip(screen)) then
        print("Couldn't Update Screen: "..SDL_GetError())
end
```

19. Before the program exits, it is important to free all allocated surfaces and quit the SDL sub-system by calling SDL_Quit() API.

```
-- Free the memory allocated for Wall and RectBox
cleanUpWall()
cleanUpRectBox()

-- Release the envent instance
SDL_Event_delete(event)

--Free the loaded image
SDL_FreeSurface(image)
SDL_FreeSurface( fontSurface1 )
SDL_FreeSurface( fontSurface2 )
SDL_FreeSurface( fontSurface3 )

-- Quit SDL_ttf
TTF_Quit()

--Close the SDL
SDL_Quit()
```

20. Clean or de-allocate the memory allocated for the wall /Bounding box instance, by using cleanUpWall()

```
function cleanUpWall()
        for i=1, NUM_WALL_UNITS do
                SDL_Rect_delete(WallUnits)
        end
end
```

21. Clean up or de –allocate the memory allocated for the Rectangle (SDL_Rect) by using cleanUpRectBox()

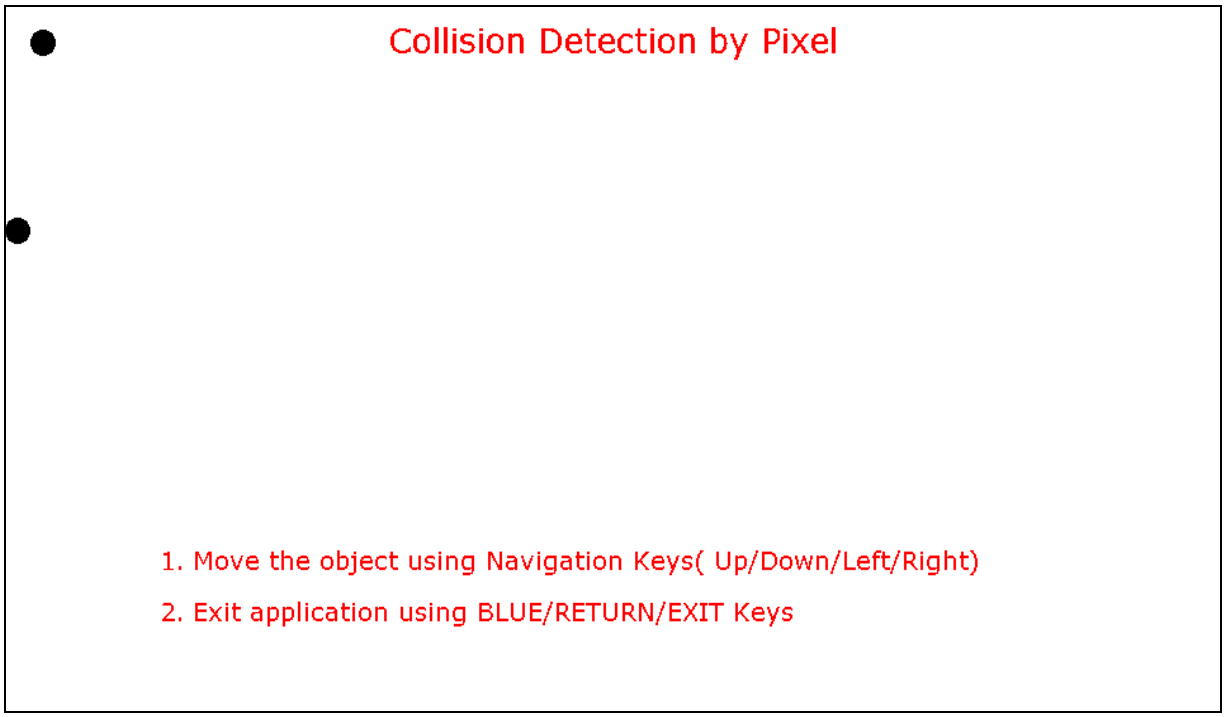
```
function cleanUpRectBox()
        SDL_Rect_delete(MovingObject)
end
```

11.1.3. Remarks

None

11.2.CD by Pixel

In this subsection we will learn about collision detection using pixel method.



11.2.1. Introduction

TV/computer screen is made up of pixels, e.g. 960x540 pixels mean there are 960 pixels in horizontal direction and 540 pixels in vertical direction. Pixels are basically squares (rectangular) boxes. Every image displayed on the screen is made up of pixels.

Collision detection by pixel is the methodology in which we try to find out whether any pixel of an object A is colliding with any pixel of adjacent object B. So, here we break the object into smaller rectangles or squares and then find out the collision.

Now, we will walkthrough an application which will detect the collision by this method.

11.2.2. Steps

1. As described in previous chapters, we will need to include SDL header files, and declare the global variables with the current program as below. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by

chResPath. The path chResPath should be defined relative to resPath argument of the Game_Main function.

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
chResPath = nil
```

2. Using below assignment, programmers can use SDL_BlitSurface API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. As in the above examples, declare all global variables/data that is used in the program

```
-- The number of boxes in which Dot is divided
NUM_BOX = 11

-- Screen attributes
SCREEN_WIDTH = 960
SCREEN_HEIGHT = 540
SCREEN_BPP = 32

-- The frame rate
FRAMES_PER_SECOND = 20

--Quit flag
quit = false

-- key motion indicators
left = false
right = false
up = false
down = false
Dot = {}
local Dot_mt = nil
```

4. In the Game_Main() , to demonstrate the collision detection we will start with creating two dot circles.

```
-- Make the dots
v1 = Dot:new(0, 0)
v2 = Dot:new(20,20)
```

5. Now we will initialize the SDL subsystem Instance and set video mode surface with specified width, height and bits per pixel. Also we will verify the initialization of the SDL with proper error message. For this we are creating a function init() and calling in main().

```
function Initialize()
        --Initialize all SDL subsystems
        if( SDL_Init( SDL_INIT_VIDEO ) == -1 ) then
                return false;
        end

        screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT,
                  SCREEN_BPP, SDL_HWSURFACE)
        if(screen == nil) then
                return false
        end

        -- Enable key repeat
        SDL_EnableKeyRepeat( 10,80)
        -- If everything initialized fine
        return true
end
```

6. So, our basic SDL setup is complete and we will load the image that is to be displayed on screen. To preserve readability of code, the operations to load the image are moved into a separate function, load_files. The function will load the dot image which we want to display. Function will return true if image is successfully loaded.

```
function load_files()

        -- Load the square image
        temp_dot = load_image( chResPath.."dot.bmp")

        -- If there was a problem in loading the square
        ……
        ……
        -- Colour which needs to be transparent
        COLORKEYr = 0
        COLORKEYg = 255
        COLORKEYb = 255

        -- Map color key
        color_key = SDL_MapRGB( temp_dot.format, COLORKEYr,
        COLORKEYg, COLORKEYb )

        -- Set color key to make specified color transparent
        if( -1 == SDL_SetColorKey( temp_dot, SDL_SRCCOLORKEY,
        color_key ) ) then
                print( "Warning: Colorkey not be used, reason:
                "..SDL_GetError() )
        end

        dot = SDL_DisplayFormatAlpha(temp_dot)

        --If everything loaded fine
        return true
end
```

7. Next, to handle the user inputs we are adding the Game loop. Handle_input function will be used to manage the UP, DOWN, LEFT and RIGHT key presses.

```
function Dot:Handle_Input()
        self.yVel = 0
        self.xVel = 0
        local    keySymbol = 0

        -- The event structure
        event = nil                                          -- event Instance
        event = SDL_Event_new()              -- event Instance initialize

        -- While there's events to handle
        SDL_PollEvent( event )

        -- If the user has Xed out the window
            if event.type == SDL_QUIT or event.key.keysym.sym ==
            SDLK_ESCAPE or    event.key.keysym.sym == SDL_QUIT
            or event.key.keysym.sym == SDLK_POWER then
                        ……
            end
            --If a key was pressed
            if( event.type == SDL_KEYDOWN ) then

            --Adjust the velocity
            keySymbol = event.key.keysym.sym

             if keySymbol == SDLK_UP then
                        ……..
             elseif keySymbol == SDLK_DOWN then
                        ……
            elseif keySymbol == SDLK_LEFT then
                        ……
                        ……..
```

```
                    ……
            elseif keySymbol == SDLK_RIGHT then
                    ……..
                    end
            if (left == true) then
                    self.xVel = - 10
            elseif(right == true) then
                    self.xVel = 10
            elseif(up == true) then
                            self.yVel = - 10
            elseif(down == true) then
                    ……
            end
      end
          SDL_Event_delete(event)
end
```

8. It will be assumed that either of the two dot circles is made up of 11 rectangular boxes stacked on one another to form the circular dot.

```
Dot = {}
local Dot_mt = nil
function LoadGameContents(dirPath)
        package.path = package.path..";"..dirPath.."/?.lua;"
        require('Class')
        Dot_mt = Class(Dot)
end
function Dot:new(X,Y)
t =     {
        x=X,
        y=Y,
        xVel=0,
        yVel=0,
        box =   {
                {x = 0, y =0,w = 6, h = 1},
                {x = 0, y =0, w = 10, h = 1},
                {x = 0, y =0, w = 14, h = 1},
                {x = 0, y =0, w = 16, h = 2},
                {x = 0, y = 0, w = 18, h = 2},
                {x = 0, y =0, w = 20, h = 6},
                {x = 0, y =0, w = 18, h = 2},
                {x = 0, y =0, w = 16, h = 2},
                {x = 0, y =0, w = 14, h = 1},
                {x = 0, y =0, w = 10, h = 1},
                {x = 0, y =0, w = 6, h = 1}
                 }
        }
        return setmetatable(t, Dot_mt)
end
```

9. We have assumed that each circular dot is made up of rectangular boxes. So we will think dot as collection of rectangular boxes. Now, suppose if we want to move the dot circle, then moving a dot circle is equivalent to moving the collection of its constituent rectangular boxes. So, to move a dot circle we will need to move its rectangular boxes.

   Move function will do this task with help of shift_boxes function.

```
function Dot:move(rects)

        -- Move the dot left or right
        self.x = self.x + self.xVel

        -- Move the collision boxes
        self:shift_boxes()

        -- If the dot went too far to the left or right or has collided
        -- with the other dot
        if((self.x < 0) or ((self.x + DOT_WIDTH) > SCREEN_WIDTH) or
        (check_collision(self.box, rects) == true)) then
                -- Move back
                self.x = self.x - self.xVel
                self:shift_boxes()
        end
        ……..
        -- Move the dot up or down
        self.y = self.y + self.yVel

        -- Move the collision boxes
        self:shift_boxes()
        ……..
```

```
        -- If the dot went too far up or down or has collided with the other dot
        if (( self.y < 0 ) or ((self.y + DOT_HEIGHT) > SCREEN_HEIGHT)
        or (check_collision( self.box, rects ) == true)) then
                -- Move back
                self.y = self.y - self.yVel
                self:shift_boxes()
        end
end
```

```
function Dot:shift_boxes()
        --The row offset
        local r = 0
        local set = 1

        -- Go through the dot's collision boxes
        for set = 1, NUM_BOX do
                --Center the collision box
                self.box[set].x = self.x + math.floor((DOT_WIDTH -
                self.box[set].w ) / 2)

                -- Set the collision box at its row offset
                self.box[set].y = self.y + r

                -- Move the row offset down the height of the collision box
                r = r + self.box[set].h
        end
end
```

10. While moving one of the dot circle, we will always need to check whether it is colliding with the second dot circle or not. Movement of the moving circle should be allowed only if

there is no collision between the two circles. To detect the collision we will use the CheckCollision function. In this function we will check the collision of rectangular boxes of one circle with another circle. If none of the rectangular boxes of first circle is colliding with any of the second circle then collision is not detected and movement is allowed.

```
function CheckCollision ( A, B )
	--The sides of the rectangles
	local leftA = 0
	local leftB = 0
	local rightA = 0
	local rightB = 0
	local topA = 0
	local topB = 0
	local bottomA = 0
	local bottomB = 0

	-- Go through the A boxes
	local Abox = 1
	local Bbox = 1

	for Abox = 1, #A do
		-- Calculate the sides of rect A
		leftA = A[Abox].x
		rightA = A[Abox].x + A[Abox].w
		topA = A[Abox].y;
		bottomA = A[Abox].y + A[Abox].h;
		-- Go through the B boxes
		for Bbox = 1, #B do
			-- Calculate the sides of rect B
			leftB = B[Bbox].x
			rightB = B[Bbox].x + B[Bbox].w
			topB = B[Bbox].y
			bottomB = B[Bbox].y + B[Bbox].h
			-- If no sides from A are outside of B
			if( ( ( bottomA <= topB ) or ( topA >= bottomB )
			or ( rightA <= leftB )
```

```
                        or ( leftA >= rightB ) ) == false ) then
                            -- A collision is detected
                            return true
                        end
                    end
                end

                -- If neither set of collision boxes touched
                return false
end
```

11. After giving any key input by user collision will be checked using CheckCollision and if no collision is found then circle will move ahead in the requested direction.

```
v1:move(v2:get_rects())
```

12. To show the dot circles on the screen, blitting is done as described in all previous chapters.

```
v1:move(v2:get_rects())

-- Fill the screen white
SDL_FillRect(screen, screen.clip_rect, SDL_MapRGB(screen.format, 0xFF, 0xFF, 0xFF))

-- Show the dots on the screen
v2:show()
v1:show()
```

```
function Dot:show()
        -- Show the dot
        ApplySurface ( self.x, self.y, dot, screen )
end

function ApplySurface (X,  Y,  source,  destination,  clip)
        -- Holds offsets
        local offset = SDL_Rect_new()
        clip = nil
        -- Get offsets
        offset.x = X
        offset.y = Y
        offset.w = 0
        offset.h = 0
        SDL_BlitSurface(source, clip, destination, offset)
end
```

13. After quiting the application, all resources will be freed and SDL will be quit.

```
function Cleanup ()

    -- Free the surface
    SDL_FreeSurface( dot )
    --Quit TTF
    TTF_Quit()
    -- Quit SDL
    SDL_Quit()

end
```

11.2.3. Remarks

None

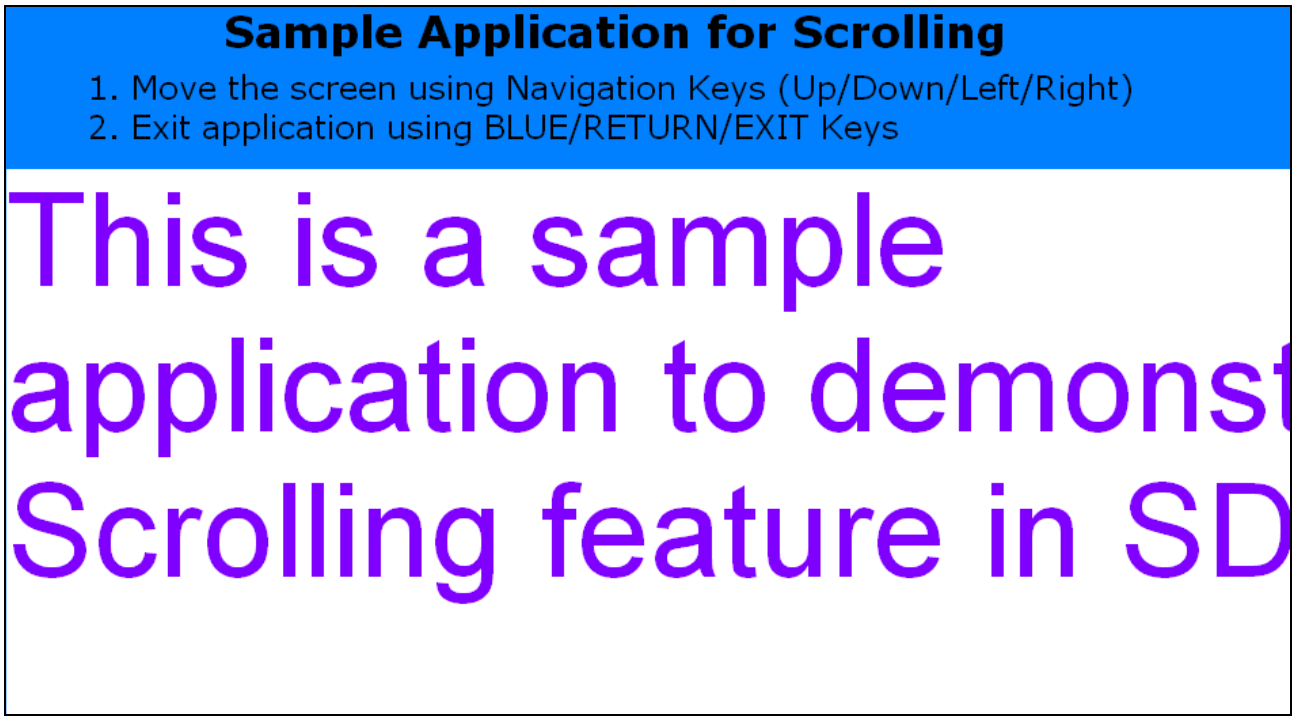
12. Scrolling and Tiling

12.1. Scrolling

This sample application describes about the scrolling functionality on DTV.

12.1.1. Introduction

This LUA scrolling application highlights the steps involved to demonstrate scrolling functionality of SDL on screen using functionality of SDL-Lua. Scrolling of camera by using the left, right, Up and down arrow keys



12.1.2. Steps

1. To use scrolling of camera by using the left ,right, Up and down arrow keys functionality of SDL, include SDL_image header and library with the current program as below

```
require('SDL')
require('SDL_image')
require('SDL_ttf')
```

2. Using below assignment, programmers can use SDL_BlitSurface API.

```
SDL_BlitSurface = SDL_UpperBlit
```

3. Set the font color to be rendered on the screen.

```
color = SDL_Color_new() -- Font color
color.r = 128
color.g = 0
color.b = 0
```

4. As in the above examples, declare all global variables/data that is used in the program. Every game would have audio, video, images, etc resources. These resources must be placed at a common location specified by chResPath. The resource path chResPath should be defined relative to resPath argument of Game_Main().

```
--Screen Variables
SCREEN_WIDTH = 960          -- Screen Width
SCREEN_HEIGHT = 540         -- Screen Height
--Scene Variables
SCENE_WIDTH = 1280          -- Scene Width
SCENE_HEIGHT = 960          -- Scene Height
--Unit of movement
UNIT_DISTANCE = 10

x = SCREEN_WIDTH / 2        -- Initial Camera X Position
y = SCREEN_HEIGHT / 2       -- Initial Camera Y Position

chResPath = nil
```

5. Initialize the SDL subsystem Instance and set video mode surface with specified width, height and bits per pixel. Also verify the Initialization of the SDL with proper error message.

```
-- Initialize SDL
if(SDL_Init(SDL_INIT_VIDEO)==-1) then
        print("Couldn't Initialize SDL: "..SDL_GetError())
end

-- Set up screen
Screen = SDL_SetVideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32,
          SDL_HWSURFACE)
if(screen==nil) then
        print("Couldn't Set Video Mode: "..SDL_GetError())
end
```

6. Initialize TTF that needs to be loaded.

```
--Initialize TTF
if ( -1 == TTF_Init() ) then
        print( "Couldn't Initialize TTF: " ..SDL_GetError() )
end
```

7. Load the image that is to be displayed on screen. To preserve readability of code, the operations to load the image are moved into a separate function, Loadimage. Input parameter to LoadImage function is the image file which needs to be loaded and it returns the loaded image. Below is the definition of LoadImage function:

```
function LoadImage(filename)
    -- Load the image
    local tempImage = IMG_Load(filename)
    if(nil == tempImage)then
        print("Error:".."could not be opened the Image: " ..IMG_GetError())
        return nil
    end
    -- Set Display Format
    loaded_image = SDL_DisplayFormat(tempImage)
    if ( nil == loaded_image ) then
            print( "Couldn't Set Display Format:" ..SDL_GetError() )
            return nil
    end
    return loaded_image
end
```

Load Image function is called with appropriate image file name as the input parameter inside the program.

Load the appropriate font to be displayed and render it to the surface as done in the previous chapters.

8. Choose the display format as an optimize format, and then after the while loop is for verifying the event type for drawing an Image. Before drawing an image the camera position must be set. Scrolling of camera by using the left ,right, Up and down arrow keys.

```
while( 0 ~= loop()) do
        -- Update the position of camera.
        position_camera()

       -- Draw the image with updated camera position.
        Draw()

end
```

-Inside the loop API the Event Instance is initiated for verifying the event type
-If the event type is SDL_QUIT or event.key.keysym.sym = SDLK_ESCAPE the while loop will break.

```
function Loop()
    event = SDL_Event_new()
    SDL_PollEvent(event)
    --End the loop when Close or Escape has been done
    if(event.type == SDL_QUIT) then
        return 0
    --if any arrow key is pressed
    elseif(event.type == SDL_KEYDOWN) then
                if(event.key.keysym.sym == SDLK_LEFT)then
                        left = 1
                        right = 0
                        up = 0
                        down = 0
                elseif(event.key.keysym.sym == SDLK_RIGHT)then
                ---
                elseif(event.key.keysym.sym == SDLK_ESCAPE) then
                        return 0
                elseif(event.key.keysym.sym == SDLK_POWER)then
                        return 0
                else
                        --print("NO event.key.keysym.sym matched")
                end
        else
                ---
        end
  --Increment or decrement the x and y values based on the pressed key for movement
  if(0 ~= left)then
        x = x + UNIT_DISTANCE
  end
  ---
  return 1                  --Continue
end
```

-For setting the position of camera and it's bound

```
function position_camera()
        --camera positions
        camera.xmin = x - SCREEN_WIDTH/2
        camera.ymin = y - SCREEN_HEIGHT/2
        camera.xmax = x + SCREEN_WIDTH/2
        camera.ymax = y + SCREEN_HEIGHT/2

        -- camera bounds.
        if(camera.xmin < -SCENE_WIDTH + SCREEN_WIDTH)then
                camera.xmin =    -SCENE_WIDTH + SCREEN_WIDTH
        end
        ……
        ……
        ……
        -- Update x and y to be in sync with xmin and ymin
        x = camera.xmin + SCREEN_WIDTH/2
        y = camera.ymin + SCREEN_HEIGHT/2
end
```

9. For rendering the images on the screen

```
-- Draw the image on the screen with appropriate surface
Draw ()
```

-Apply the image to screen (appropriate surface), this performs a fast blit from the source surface to the destination surface.

```
-- Setting the offset for display location
offset = SDL_Rect_new()
offset.x = camera.xmin
offset.y = camera.ymin

SDL_BlitSurface(image, nil, screen, nil)
SDL_BlitSurface (help, nil, screen, nil)
```

10. It is important to de-allocated the surfaces and quit the SDL sub-system before exiting from the scrolling application.

```
-- cleanup
cleanup()

- In cleanup() deallocate all the Instance and free the SDL surface.
-- Free the SDL Rect
SDL_Rect_delete(offset)

-- Free SDL Event
SDL_Event_delete(event)

-- Free the loaded image
SDL_FreeSurface(image)

--Quit TTF
TTF_Quit()

-- Close SDL
SDL_Quit()
```

12.1.3. Remarks

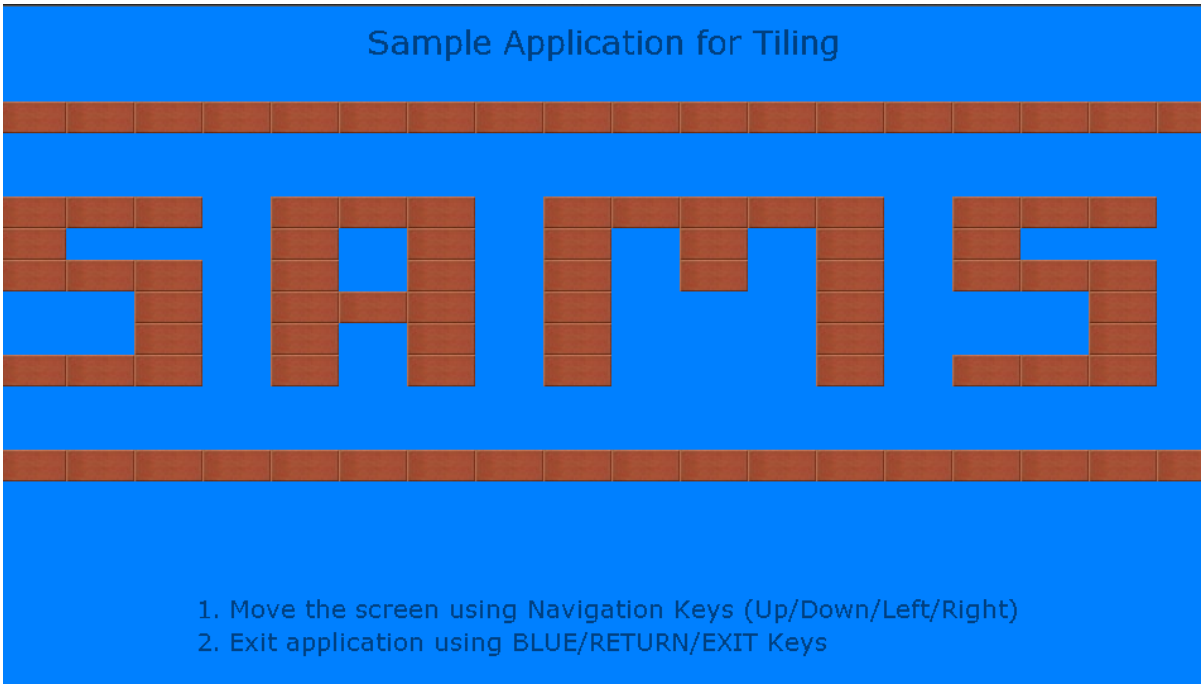
Game Framework supports scrolling of camera by using the left, right, Up and down arrow keys.

12.2.Tiling

12.2.1. Introduction

A tile is a block image that is used to create a patterned background. Tiling requires only a portion of the image to be loaded in memory and constructs the background by decisively placing the tiles next to each other and generating a patterned background scene. Thus tiling helps to save large chunk of memory otherwise required for the backgrounds. Let us understand the use of sprites in gaming applications.

Above is an image tile. It resembles a brick. Just as bricks build a wall, tiles build the background scene.
The below pattern is generated using the above tile.



Now that you know some basics, let's try doing it in code.

12.2.2. Steps

In previous chapters, you have learnt how to include appropriate headers, initialize SDL subsystem, and set the video mode.

So we will proceed with tiling.

1. To start with, declare the global variables/data that are used in the program. We define a 2D array to store tiling pattern for generating the background.

```
SCREEN_WIDTH    = 960       -- Screen Width
SCREEN_HEIGHT = 540         -- Screen Height
chResPath = nil

-- Tile Sizes
TILE_LENGTH = 54            -- Tile length
TILE_HEIGHT = 25            -- Tile height

-- Tile Count
NUM_TILES_X = 32            -- Number of columns
NUM_TILES_Y = 18            -- Number of rows

-- Unit of movement
UNIT_DISTANCE = 10

-- Arrow key press status
left = false
right = false
up = false
down = false
```

2. Define the tile map. Tile map contains the information for placing the tiles. In this map the entries are comma separated. An entry of 1 means a tile should be placed at the corresponding coordinate. Below is the tile map used for this application:

```
-- Start and End Co-ordinates
startlocX = 0
startlocY = 0
endlocX = (NUM_TILES_X * TILE_LENGTH)
endlocY = (NUM_TILES_Y * TILE_HEIGHT)

tiles = {
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0},
{1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0},
{1, 1, 1, 0, 1, 0, 1, 0 ,1 ,0 ,1 ,0 ,1 ,0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0},
{0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
{0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
{1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
}
```

3. Set the font color

```
--color
color = SDL_Color_new() -- Font color
color.r =    0
color.g = 64
color.b = 128
```

4. Load the tile image and background image.

```
-- Load the background image to be displayed
image = LoadImage(chResPath.."background.bmp")
    if(nil == image) then
        print("Error: File Couldn't be opened:"..IMG_GetError())
        CleanUp()
end

-- Load the tile to be displayed
tileImage = LoadImage(chResPath.."tile.jpg")
if(nil == tileImage) then
        print("Error: File Couldn't be opened:"..IMG_GetError())
        CleanUp()
end
```

5. Draw() calls DrawTiles() to compute the position co-ordinates for placing tiles. Then it blits tiles to the screen.

```
function DrawTiles()
        offset = SDL_Rect_new()
        for j = 1, NUM_TILES_Y, 1 do
                for i = 0, NUM_TILES_X, 1 do
                        tile = getTile(i, j)
                        if(0 ~= tile) then
                                offset.x = (i-1) * TILE_LENGTH + getStartlocX()
                                offset.y = (j-1) * TILE_HEIGHT + getStartlocY()
                                SDL_BlitSurface(tileImage, nil, screen, offset)
                        end
                end
        end
end
```

12.2.3. Remarks

None

13. Appendix

13.1. SDL GLUE Layer APIs

API Name	API Descriptions
General	
SDL_Init	The SDL_Init function initializes the Simple Directmedia Library and the subsystems specified by flags. It should be called before all other SDL functions.
SDL_InitSubSystem	After SDL has been initialized with SDL_Init ,initialize the subsystems with SDL_InitSubSystem. The flags parameter is the same as that used in SDL_Init
SDL_QuitSubSystem	SDL_QuitSubSystem allows you to shut down a subsystem that has been previously initialized by SDL_Init or SDL_InitSubSystem. The flags tells SDL_QuitSubSystem which subsystems to shut down, it uses the same values that are passed to SDL_Init.
SDL_Quit	Shuts down all SDL subsystems, unloads the dynamically linked library and frees the allocated resources
SDL_WasInit	SDL_WasInit allows to see which SDL subsytems have been initialized. flags is a bitwise OR'd combination of the subsystems to check (see SDL_Init for a list of subsystem flags). If 'flags' is 0 or SDL_INIT_EVERYTHING, it returns a mask of all initialized subsystems (this does not include SDL_INIT_EVENTTHREAD or SDL_INIT_NOPARACHUTE).
SDL_GetError	Returns a nil terminated string containing information about the last internal SDL error.
SDL_SetError	sets the SDL error to a printf style formatted string.
SDL_Error	Sets the SDL error message to the predefined string specified by code.
SDL_ClearError	SDL_ClearError deletes all information about the last internal SDL error. Useful if the error has been handled by the program

SDL_Linked_Version	The SDL_Linked_Version function gets the version of the current dynamically linked SDL library and returns a instance to a SDL_version structure containing the version information.
SDL_version	The SDL_version structure is used by the SDL_Linked_Version function and the SDL_VERSION macro. The SDL_Linked_Version function returns the link-time SDL version whereas the SDL_VERSION macro returns the compile-time SDL version.
SDL_VERSION(macro)	The SDL_VERSION macro is a helper macro that fills out a SDL_version structure with the compile-time SDL version.

SDL Video	
SDL_GetVideoSurface	This function returns a instance to the current display surface. If SDL is doing format conversion on the display surface, this function returns the publicly visible surface, not the real video surface.
SDL_GetVideoInfo	This function returns a read-only instance to a structure containing information about the video hardware. If it is called before SDL_SetVideoMode, the vfmt member of the returned structure will contain the pixel format of the best video mode.
SDL_VideoDriverName	The buffer pointed to by namebuf is filled up to a maximum of maxlen characters (including the nil character) with the name of the initialized video driver. The driver name is a simple one word identifier like "x11", "windib" or "directx". Some platforms allow selection of the video driver through the SDL_VIDEODRIVER environment variable.
SDL_ListModes	The returned instance is an array of SDL_Rect instances to available screen dimensions for the given pixel format and video flags. The array is not guaranteed to be sorted in any particular order. The function returns nil if there is not any mode available for the particular format, or -1 if any dimension is okay for the given format.
SDL_VideoModeOK	SDL_VideoModeOK returns 0 if the requested mode is not supported under any bit depth, or returns the bits-per-pixel of the closest available mode with the given width, height and requested surface flags (see SDL_SetVideoMode). The bits-per-pixel value returned is only a suggested mode. You can usually request any bpp you want when setting the video mode and SDL will emulate that color depth with a shadow video surface.
SDL_SetVideoMode	Set up a video mode with the specified width, height and bitsperpixel.

SDL_UpdateRect	The given area is updated on the given screen. In other words, it makes sure any changes to the given area of the screen are made visible. The rectangle must be confined within the screen boundaries because there's no clipping.
SDL_UpdateRects	The given list of rectangles is updated on the given screen. The rectangles must all be confined within the screen boundaries (no clipping is done). passing rectangles not confined within the screen boundaries to this function can cause very nasty crashes, at least with SDL 1.2.8, at least in Windows and Linux. This function should not be called while screen is locked. It is advised to call this function only once per frame, since each call has some processing overhead. This is no restriction since you can pass any number of rectangles each time. The rectangles are not automatically merged or checked for overlap. The programmer can use knowledge about the particular rectangles to merge them in an efficient way, to avoid overdraw.
SDL_DisplayFormat	This function takes a surface and copies it to a new surface of the pixel format and colors of the video framebuffer, suitable for fast blitting onto the display surface. It calls SDL_ConvertSurface. If you want to take advantage of hardware colorkey or alpha blit acceleration, you should set the colorkey and alpha value before calling this function.

SDL_Flip	Swap screen buffer.On hardware that supports double-buffering, this function sets up a flip and returns. The hardware will wait for vertical retrace, and then swap video buffers before the next video surface blit or lock will return. On hardware that doesn't support double-buffering or if SDL_SWSURFACE was set, this is equivalent to calling SDL_UpdateRect(screen, 0, 0, 0, 0) .A software screen surface is also updated automatically when parts of a SDL window are redrawn, caused by overlapping windows or by restoring from an iconified state. As a result there is no proper double buffer behavior in windowed mode for a software screen, in contrast to a full screen software mode. The SDL_DOUBLEBUF flag must have been passed to SDL_SetVideoMode, when setting the video mode for this function to perform hardware flipping.
SDL_MapRGB	Maps the RGB color value to the specified pixel format and returns the pixel value as a 32-bit int. If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned. If the specified pixel format has an alpha component it will be returned as all 1 bits (fully opaque).
SDL_MapRGBA	Maps the RGBA color value to the specified pixel format and returns the pixel value as a 32-bit int. The format has a palette (8-bit) the index of the closest matching color in the palette will be returned. The specified pixel format has no alpha component the alpha value will be ignored (as it will be in formats with a palette).
SDL_GetRGB	Get RGB component values from a pixel stored in the specified pixel format. This function uses the entire 8-bit [0..255] range when converting color components from pixel formats with less than 8-bits per RGB component (e.g., a completely white pixel in 16-bit RGB565 format would return [0xff, 0xff, 0xff] not [0xf8, 0xfc, 0xf8]).

SDL_GetRGBA	Get RGBA component values from a pixel stored in the specified pixel format.This function uses the entire 8-bit [0..255] range when converting color components from pixel formats with less than 8-bits per RGB component (e.g., a completely white pixel in 16-bit RGB565 format would return [0xff, 0xff, 0xff] not [0xf8, 0xfc, 0xf8]). The surface has no alpha component, the alpha will be returned as 0xff (100% opaque).
SDL_LoadBMP	Loads a surface from a named Windows BMP file. When loading a 24-bit Windows BMP file, pixel data points are loaded as blue, green, red, and NOT red, green, blue (as one might expect).
SDL_SetColorKey	Sets the color key (transparent pixel) in a blittable surface and enables or disables RLE blit acceleration. RLE acceleration can substantially speed up blitting of images with large horizontal runs of transparent pixels (i.e., pixels that match the key value). The key must be of the same pixel format as the surface, SDL_MapRGB is often useful for obtaining an acceptable value. If flag is SDL_SRCCOLORKEY then key is the transparent pixel value in the source image of a blit. If flag is OR'd with SDL_RLEACCEL then the surface will be drawn using RLE acceleration when drawn with SDL_BlitSurface. The surface will actually be encoded for RLE acceleration the first time SDL_BlitSurface or SDL_DisplayFormat is called on the surface.
SDL_SetClipRect	Sets the clipping rectangle for a surface. When this surface is the destination of a blit, only the area within the clip rectangle will be drawn into. The rectangle pointed to by rect will be clipped to the edges of the surface so that the clip rectangle for a surface can never fall outside the edges of the surface. If rect is nil the clipping rectangle will be set to the full size of the surface.
SDL_GetClipRect	Gets the clipping rectangle for a surface. When this surface is the destination of a blit, only the area within the clip rectangle is drawn into.The rectangle pointed to by rect will be filled with the clipping rectangle of the surface.
SDL_BlitSurface	This performs a fast blit from the source surface to the destination surface. The width and height in srcrect determine the size of the copied rectangle. Only the position is used in the dstrect (the

	width and height are ignored). Blits with negative dstrect coordinates will be clipped properly. If srcrect is nil, the entire surface is copied. If dstrect is nil, then the destination position (upper left corner) is (0, 0). The final blit rectangle is saved in dstrect after all clipping is performed (srcrect is not modified). The blit function should not be called on a locked surface. I.e. drawing functions need to lock a surface, but this is not the case with SDL_BlitSurface.
SDL_FillRect	This function performs a fast fill of the given rectangle with color. If dstrect is nil, the whole surface will be filled with color. The color should be a pixel of the format used by the surface, and can be generated by the SDL_MapRGB or SDL_MapRGBA functions. If the color value contains an alpha value then the destination is simply "filled" with that alpha information, no blending takes place. If there is a clip rectangle set on the destination (set via SDL_SetClipRect), then this function will clip based on the intersection of the clip rectangle and the dstrect rectangle, and the dstrect rectangle will be modified to represent the area actually filled.
SDL_SaveBMP	Saves the SDL_Surface surface as a Windows BMP file named file.
SDL_SetPalette	Sets the colors in the palette of an 8-bit surface.The color components of a SDL_Color structure are 8-bits in size, giving you a total of 2563 = 16777216 colors.

SDL_CreateRGBSurface	Create an empty SDL_Surface.If bitsPerPixel is 8 an empty palette is allocated for the surface, otherwise a 'packed-pixel' SDL_PixelFormat is created using the [RGBA]mask's provided (see SDL_PixelFormat). The flags specifies the type of surface that should be created, it is an OR'd combination of the following possible values. If an alpha-channel is specified (that is, if Amask is nonzero), then the SDL_SRCALPHA flag is automatically set. To remove this flag by calling SDL_SetAlpha after surface creation. Also, if the SDL_HWSURFACE flag is set on the returned surface, its format might not match the requested format. Sometimes specifying an Alpha mask value could cause strange results. This can be worked around by setting the Amask parameter to 0, but still including the SDL_SRCALPHA flag, and then using SDL_SetAlpha, also with the SDL_SRCALPHA flag.
SDL_CreateRGBSurfaceFrom	Create an SDL_Surface from pixel data.The pixel data is considered to be in software memory. If the pixel data lies in hardware memory (as pixel data from a hardware surface), the appropriate surface flag has to be set manually.
SDL_FreeSurface	Frees the resources used by a previously created SDL_Surface. If the surface was created using SDL_CreateRGBSurfaceFrom then the pixel data is not freed.
SDL_LockSurface	SDL_LockSurface sets up a surface for directly accessing the pixels. Between calls to SDL_LockSurface and SDL_UnlockSurface, you can write to and read from surface->pixels, using the pixel format stored in surface->format. Once you are done accessing the surface, you should use SDL_UnlockSurface to release the lock. Not all surfaces require locking. If SDL_MUSTLOCK(surface) evaluates to 0, then reading and writing pixels to the surface can be performed at any time, and the pixel format of the surface will not change. No operating system or library calls should be made between the lock/unlock pairs, as critical system locks may be held during this time. In SDL 1.1.8, the surface locks are recursive. This means that the lock a surface multiple times, but each lock must have a matching unlock.

SDL_UnlockSurface	Surfaces that were previously locked using SDL_LockSurface must be unlocked with SDL_UnlockSurface. Surfaces should be unlocked as soon as possible.In SDL 1.1.8, the surface locks are recursive. See SDL_LockSurface for more information.
SDL_GetGammaRamp	Gets the gamma translation lookup tables currently used by the display.
SDL_SetGammaRamp	Sets the gamma lookup tables for the display for each color component. Each table is an array of 256 Uint16 values, representing a mapping between the input and output for that channel. The input is the index into the array, and the output is the 16-bit gamma value at that index, scaled to the output color precision. To pass nil to any of the channels to leave them unchanged.
SDL_SetGamma	Sets the "gamma function" for the display of each color component. Gamma controls the brightness/contrast of colors displayed on the screen. A gamma value of 1.0 is identity.
SDL_ConvertSurface	Creates a new surface of the specified format, and then copies and maps the given surface to it. If this function fails, it returns nil. It is also useful for making a copy of a surface.
SDL_CreateYUVOverlay	SDL_CreateYUVOverlay creates a YUV overlay of the specified width, height and format (see SDL_Overlay for a list of available formats), for the provided display. A SDL_Overlay structure is returned. The term 'overlay' is a misnomer since, unless the overlay is created in hardware, the contents for the display surface underneath the area where the overlay is shown will be overwritten when the overlay is displayed.
SDL_LockYUVOverlay	Equivalent to SDL_LockSurface, SDL_LockYUVOverlay locks the overlay for direct access to pixel data.
SDL_UnlockYUVOverlay	The opposite to SDL_LockYUVOverlay. Unlocks a previously locked overlay. An overlay must be unlocked before it can be displayed.

SDL_DisplayYUVOverlay	Blit the overlay to the display surface specified when the overlay was created. The SDL_Rect structure, dstrect, specifies a rectangle on the display where the overlay is drawn. The .x and .y fields of dstrect specify the upper left location in display coordinates. The overlay is scaled (independently in x and y dimensions) to the size specified by dstrect, and is optimized for 2x scaling.
SDL_FreeYUVOverlay	Frees an overlay created by SDL_CreateYUVOverlay.
SDL_SetAlpha	Adjust the alpha properties of a surface .This function and the semantics of SDL alpha blending have changed since version 1.1.4. Up until version 1.1.5, an alpha value of 0 was considered opaque and a value of 255 was considered transparent. This has now been inverted: 0 (SDL_ALPHA_TRANSPARENT) is now considered transparent and 255 (SDL_ALPHA_OPAQUE) is now considered opaque. SDL_SetAlpha is used for setting the per-surface alpha value and/or enabling and disabling alpha blending.
SDL_SetColors	Sets a portion of the colormap for the given 8-bit surface. When surface is the surface associated with the current display, the display colormap will be updated with the requested colors. If SDL_HWPALETTE was set in SDL_SetVideoMode flags, SDL_SetColors will always return 1, and the palette is guaranteed to be set the way you desire, even if the window colormap has to be warped or run under emulation. The color components of a SDL_Color structure are 8-bits in size, giving you a total of 2563 = 16777216 colors.

SDL_DisplayFormatAlpha	Convert a surface to the display format. This function takes a surface and copies it to a new surface of the pixel format and colors of the video framebuffer plus an alpha channel, suitable for fast blitting onto the display surface. It calls SDL_ConvertSurface. The advantage of hardware colorkey or alpha blit acceleration, it should set the colorkey and alpha value before calling this function. This function can be used to convert a colorkey to an alpha channel, if the SDL_SRCCOLORKEY flag is set on the surface. The generated surface will then be transparent (alpha=0) where the pixels match the colorkey, and opaque (alpha=255) elsewhere. It returns a copy of surface converted to the display format. This surface must be freed using SDL_FreeSurface. If the conversion fails or runs out of memory, it returns nil.
SDL_Overlay	A SDL_Overlay is similar to a SDL_Surface except it stores a YUV overlay. All the fields are read only, except for pixels which should be locked before use. The format field stores the format of the overlay which is one of the following: #define SDL_YV12_OVERLAY 0x32315659 /* Planar mode: Y + V + U */ #define SDL_IYUV_OVERLAY 0x56555949 /* Planar mode: Y + U + V */ #define SDL_YUY2_OVERLAY 0x32595559 /* Packed mode: Y0+U0+Y1+V0 */ #define SDL_UYVY_OVERLAY 0x59565955 /* Packed mode: U0+Y0+V0+Y1 */ #define SDL_YVYU_OVERLAY 0x55595659 /* Packed mode: Y0+V0+Y1+U0 */

SDL_Event	
SDL_WaitEvent	Waits indefinitely for the next available event, returning 0 if there was an error while waiting for events, 1 otherwise. If event is not nil, the next event is removed from the queue and stored in that area.
SDL_GetModState	The return value is an OR'd combination of the SDLMod enum.
SDL_SetModState	Set the current key modifier state.
SDL_EnableUNICODE	Enables/Disables Unicode keyboard translation. To obtain the character codes corresponding to received keyboard events, Unicode translation must first be turned on using this function. The translation incurs a slight overhead for each keyboard event and is therefore disabled by default. For each subsequently received key down event, the unicode member of the SDL_keysym structure will then contain the corresponding character code, or zero for keysyms that do not correspond to any character code.
SDL_EnableKeyRepeat	Enables or disables the keyboard repeat rate. delay specifies how long the key must be pressed before it begins repeating, it then repeats at the speed specified by interval. Both delay and interval are expressed in milliseconds. Setting delay to 0 disables key repeating completely. Good default values are SDL_DEFAULT_REPEAT_DELAY and SDL_DEFAULT_REPEAT_INTERVAL.
SDL_PollEvent	Polls for currently pending events. If event is not nil, the next event is removed from the queue and stored in the SDL_Event structure pointed to by event.
SDL_PushEvent	The event queue can actually be used as a two way communication channel. Not only can events be read from the queue, but the user can also push their own events onto it. event is a instance to the event structure to push onto the queue. The event is copied into the queue, and the caller may dispose of the memory pointed to after SDL_PushEvent returns.
SDL_GetKeyName	Get the name of an SDL virtual keysym.
SDL_PumpEvents	Pumps the event loop, gathering events from the input devices.

SDL_PeepEvents	Checks the event queue for messages and optionally returns them. If action is SDL_ADDEVENT, up to numevents events will be added to the back of the event queue. If action is SDL_PEEKEVENT, up to numevents events at the front of the event queue, matching mask, will be returned and will not be removed from the queue. If action is SDL_GETEVENT, up to numevents events at the front of the event queue, matching mask, will be returned and will be removed from the queue. The mask parameter is a bitwise OR of SDL_EVENTMASK(event_type), for all event types. This function returns the number of events actually stored, or -1 if there was an error.This function is thread-safe.
SDL_GetEventFilter	Retrieves an instance to the event filter function. It returns a pointer to the event filter or NULL if no filter has been set.
SDL_SetEventFilter	This function sets up a filter to process all events before they are posted to the event queue. This is a very powerful and flexible feature. If the filter returns 1, then the event will be added to the internal queue. If it returns 0, then the event will be dropped from the queue. This allows dynamically-selective filtering of events. There is one caveat when dealing with the SDL_QUITEVENT event type. The event filter is only called when the window manager desires to close the application window. If the event filter returns 1, then the window will be closed, otherwise the window will remain open if possible. If the quit event is generated by an interrupt signal, it will bypass the internal queue and be delivered to the application at the next event poll. **Note:** Events pushed onto the queue with SDL_PushEvent or SDL_PeepEvents do not get passed through the event filter.
SDL_EventState	This function allows you to set the state of processing certain events. SDL_EventState must be called following the SDL_VideoInfo call. Otherwise the back-end data structure will be erased.
SDL_GetKeyState	Gets a snapshot of the current keyboard state. The current state is returned as an instance to an table. The size of this array is stored

<table>
<tr><td></td><td>in numkeys. The table is indexed by theSDLK_* symbols (see SDLKey). A value of 1 means that the key is pressed and a value of 0 means that it is not. The instance returned is an instance to an internal SDL table . It will be valid for the whole lifetime of the application and should not be freed by the caller.
Note: Use SDL_PumpEvents to update the state table.This function gives the current state after all events have been processed, so if a key or button has been pressed and released before you process events, then the pressed state will never show up in the getstate calls.This function doesn't take into account whether shift has been pressed or not.</td></tr>
<tr><td>SDL_GetAppState</td><td>This function returns the current state of the application. The value returned is a bitwise combination of:
<table>
<tr><td>SDL_APPMOUSEFOCUS</td><td>The application has mouse focus.</td></tr>
<tr><td>SDL_APPINPUTFOCUS</td><td>The application has keyboard focus</td></tr>
<tr><td>SDL_APPACTIVE</td><td>The application is visible</td></tr>
</table>
</td></tr>
</table>

SDL Audio	
SDL_OpenAudio	This function opens the audio device with the desired parameters, and returns 0 if successful, placing the actual hardware parameters in the structure pointed to by obtained. If obtained is nil, the audio data passed to the callback function will be guaranteed to be in the requested format, and will be automatically converted to the hardware audio format if necessary. This function returns -1 if it failed to open the audio device, or couldn't set up the audio thread.
SDL_PauseAudio	This function pauses and unpauses the audio callback processing. It should be called with pause_on=0 after opening the audio device to start playing sound. This is safely initialize data for your callback function after opening the audio device. Silence will be written to the audio device during the pause.
SDL_GetAudioStatus	Get the current audio state.It is an Enum of SDL_AUDIO_STOPPED, SDL_AUDIO_PLAYING, SDL_AUDIO_PAUSED.
SDL_LoadWAV(double Uint8** instance)	This function loads a WAVE file into memory. If this function succeeds, it returns the given SDL_AudioSpec, filled with the audio data format of the wave data, and sets audio_buf to a malloc'd buffer containing the audio data, and sets audio_len to the length of that audio buffer, in bytes. To free the audio buffer with SDL_FreeWAV when you are done with it. This function returns nil and sets the SDL error message if the wave file cannot be opened, uses an unknown data format, or is corrupt. Currently raw, MS-ADPCM and IMA-ADPCM WAVE files are supported.
SDL_FreeWAV	After a WAVE file has been opened with SDL_LoadWAV its data can eventually be freed with SDL_FreeWAV. audio_buf is a instance to the buffer created by SDL_LoadWAV.
SDL_AudioCVT	The SDL_AudioCVT is used to convert audio data between different formats. A SDL_AudioCVT structure is created with the SDL_BuildAudioCVT function, while the actual conversion is done by the SDL_ConvertAudio function.

SDL_BuildAudioCVT	Before an SDL_AudioCVT structure can be used to convert audio data it must be initialized with source and destination information. src_format and dst_format are the source and destination format of the conversion. (For information on audio formats see SDL_AudioSpec). src_channels and dst_channels are the number of channels in the source and destination formats. Finally, src_rate and dst_rate are the frequency or samples-per-second of the source and destination formats. Currently (SDL-1.2.11) only rate conversions of 2x and (1/2)x with x > 0 are done, nearing the requested rate conversion.
SDL_ConvertAudio	SDL_ConvertAudio takes one parameter, cvt, which was previously initialized. Initializing a SDL_AudioCVT is a two step process. First of all, the structure must be passed to SDL_BuildAudioCVT along with source and destination format parameters. Secondly, the cvt->buf and cvt->len fields must be setup. cvt->buf should point to the audio data buffer beeing source and destination at once and cvt->len should be set to the buffer length in bytes. Remember, the length of the buffer pointed to by buf should be len*len_mult bytes in length. Once the SDL_AudioCVT structure is initialized, we can pass it to SDL_ConvertAudio, which will convert the audio data pointed to by cvt->buf. If SDL_ConvertAudio returned 0 then the conversion was completed successfully, otherwise -1 is returned. If the conversion completed successfully then the converted audio data can be read from cvt->buf. The amount of valid, converted, audio data in the buffer is equal to cvt->len*cvt->len_ratio.
SDL_MixAudio	This function takes two audio buffers of len bytes each of the playing audio format and mixes them, performing addition, volume adjustment, and overflow clipping. The volume ranges from 0 to SDL_MIX_MAXVOLUME and should be set to the maximum value for full audio volume. Note this does not change hardware volume.
SDL_LockAudio	The lock manipulated by these functions protects the callback function. During a LockAudio period, the callback function is not running. Do not call this from the callback function or you will cause deadlock.
SDL_UnlockAudio	Unlock the callback function .

SDL_CloseAudio	This function shuts down audio processing and closes the audio device.

SDL Multi-threaded programming	
SDL_CreateThread	SDL_CreateThread creates a new thread of execution that shares all of its parent's global memory, signal handlers, file descriptors, etc, and runs the function fn passing it the void instance data. The thread quits when fn returns.
SDL_ThreadID	Get the 32-bit thread identifier for the current thread.
SDL_GetThreadID	Get the SDL thread ID of a SDL_Thread .
SDL_WaitThread	Wait for a thread to finish.The return code for the thread function is placed in the area pointed to by status, if status is not nil.
SDL_KillThread	Gracelessly terminates the thread.
SDL_CreateMutex	Create a new, unlocked mutex
SDL_DestroyMutex	Destroy a previously created mutex.
SDL_mutexP	Locks the mutex, which was previously created with SDL_CreateMutex. If the mutex is already locked by another thread then SDL_mutexP will not return until the thread that locked it unlocks it (with SDL_mutexV). If called repeatedly on a mutex, SDL_mutexV must be called equal amount of times to return the mutex to unlocked state.
SDL_mutexV	Unlocks the mutex, which was previously created with SDL_CreateMutex. Returns 0 on success, or -1 on an error.
SDL_CreateSemaphore	Creates a new semaphore and assigns an initial value to it.Each locking operation on the semaphore by SDL_SemWait, SDL_SemTryWait or SDL_SemWaitTimeout will atomically decrement the semaphore value. The locking operation will be blocked if the semaphore value is not positive.
SDL_DestroySemaphore	It destroys the semaphore pointed to by sem that was created by SDL_CreateSemaphore. It is not safe to destroy a semaphore if there are threads currently blocked waiting on it.

SDL_SemWait	It suspends the calling thread until either the semaphore pointed to by sem has a positive value or the call is interrupted by a signal or error. If the call is successful it will atomically decrement the semaphore value. After SDL_SemWait is successful, the semaphore can be released and its count atomically incremented by a successful call to SDL_SemPost.
SDL_SemTryWait	It is a non-blocking variant of SDL_SemWait. If the value of the semaphore pointed to by sem is positive it will atomically decrement the semaphore value and return 0, otherwise it will return SDL_MUTEX_TIMEDOUT instead of suspending the thread.
SDL_SemWaitTimeout	It is a variant of SDL_SemWait with a maximum timeout value. If the value of the semaphore pointed to by sem is positive, it will atomically decrement the semaphore value and return 0, otherwise it will wait up to timeout milliseconds trying to lock the semaphore. This function is to be avoided if possible since on some platforms it is implemented by polling the semaphore every millisecond in a busy loop.
SDL_SemPost	It unlocks the semaphore pointed to by sem and atomically increments the semaphore's value. Threads that were blocking on the semaphore may be scheduled after this call succeeds.
SDL_SemValue	It returns the current semaphore value from the semaphore pointed to by sem.
SDL_CreateCond	Creates a condition variable.
SDL_DestroyCond	Destroys a condition variable.
SDL_CondSignal	Restart one of the threads that are waiting on the condition variable, cond. Returns 0 on success or -1 on an error.
SDL_CondBroadcast	Restarts all threads that are waiting on the condition variable, cond. Returns 0 on success, or -1 on an error.
SDL_CondWait	Unlock the provided mutex and wait for another thread to call SDL_CondSignal or SDL_CondBroadcast on the condition variable cond, then re-lock the mutex and return. The mutex must be locked before entering this function. Returns 0 when it is signalled, or -1 on an error.

SDL_CondWaitTimeout	Wait on the condition variable cond for, at most, ms milliseconds. mut is unlocked so it must be locked when the function is called and it will re-lock afterward. Returns SDL_MUTEX_TIMEDOUT if the condition is not signalled in the allotted time, 0 if it was signalled or -1 on an error.

SDL Time	
SDL_GetTicks	Gets the number of milliseconds since SDL library initialization.Returns the number of milliseconds since SDL library initialization. This value wraps around if the program runs for more than 49.7 days
SDL_Delay	Waits a specified number of milliseconds before returning.This function waits a specified number of milliseconds before returning. It waits at least the specified time, but possible longer due to OS scheduling. The delay granularity is at least 10 ms. Some platforms have shorter clock ticks but this is the most common.
SDL_AddTimer	Adds a timer which will call a callback after the specified number of milliseconds has elapsed. The timer callback function may run in a different thread than your main program, and so shouldn't call any functions from within itself. However, you may always call SDL_PushEvent. for useing this function, you need to pass SDL_INIT_TIMER to SDL_Init.
SDL_RemoveTimer	Removes a timer which was added with SDL_AddTimer.
SDL_SetTimer	Set a callback to run after the specified number of milliseconds has elapsed. The callback function is passed the current timer interval and returns the next timer interval. If the returned value is the same as the one passed in, the periodic alarm continues, otherwise a new alarm is scheduled.

SDL Files (RWops)	
SDL_RWFromFile	opens a file.In order to open a file as a binary file, a "b" character has to be included in the mode string. This additional "b" character can either be appended at the end of the string (thus making the following compound modes: "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the "+" sign for the mixed modes ("rb+", "wb+", "ab+"). Additional characters may follow the sequence, although they should have no effect. For example, "t" is sometimes appended to make explicit the file is a text file.
SDL_RWFromMem	This Api prepares a memory area for use with Rwops.SDL_RWFromMem sets up a RWops struct based on a chunk of memory of a certain size.If the memory is not writable, use SDL_RWFromConstMem instead.
SDL_RWFromConstMem	SDL_RWFromConstMem sets up a RWops struct based on a memory area of a certain size. It assumes the memory area is not writable.
SDL_AllocRW	SDL_AllocRW allocates an empty, unpopulated SDL_RWops structure.Returns a instance to the allocated memory on success, or nil on error.
SDL_FreeRW	SDL_FreeRW frees an SDL_RWops structure previously allocated by SDL_AllocRW. Only use it on memory allocated by SDL_AllocRW. Any extra memory allocated during creation of the RWops is not freed by SDL_FreeRW; the programmer must be responsible for it.
SDL_RWops	These functions are used to provide an SDL_RWops structure from various sources. The names of the functions should describe what they do (FP is a file instance). Then of course, don't forget to free the structure when you've finished with it using SDL_RWclose() or SDL_FreeRW(). Note that SDL_FreeRW() only frees the actual structure, not any other resources (such as file instances) that may have been allocated when it

	was created.

SDL_RWseek	
SDL_RWtell	SDL_RWtell is a macro that performs a do-nothing seek to get the current offset in an SDL_RWops stream. It takes one parameter, a instance to an SDL_RWops structure.Returns the offset in the stream. This is not a built-in function. This is a macro that calls whatever seek function that happens to be pointed to in an SDL_RWops structure. It should return the final offset in the data source.
SDL_RWread	SDL_RWseek is a macro that calls the seek function instance in an SDL_RWops structure. It takes the same 3 parameters as the function instance: 1.A instance to an SDL_RWops structure 2.An offset in bytes. This can be a negative value. 3.SEEK_SET, SEEK_CUR, or SEEK_END. SEEK_SET seeks from the beginning of the file, SEEK_CUR from the current position, and SEEK_END from the end of the file.
SDL_RWwrite	SDL_RWwrite is a macro that calls the write function in an SDL_RWops structure. It takes the same parameters as the write function given in the SDL_RWops structure: 1.A instance to an SDL_RWops structure 2.A instance to an area in memory to read data from 3.The size of the memory blocks to write 4.The exact number of memory blocks to write .On success, it returns the number of memory blocks you told it to write. If it couldn't write that exact number of blocks, or the write didn't work at all, it returns -1.
SDL_RWclose	SDL_RWclose calls the close function in an SDL_RWops structure. It only takes one parameter, a instance to an SDL_RWops structure. Returns 0 on success, -1 on error.
SDL_RWops	These functions are used to provide an SDL_RWops structure from various sources. The names of the functions should describe what they do (FP is a file instance). Then of course, don't forget to free the structure when finished with it using SDL_RWclose() or SDL_FreeRW(). SDL_FreeRW() only frees the actual structure, not any other resources (such as file instances) that may have been allocated when it was created.

SDL_Image	
IMG_LoadTyped_RW	Load src for use as a surface. This can load all supported image formats. This method does not guarantee that the format specified by type is the format of the loaded image, except in the case when TGA format is specified (or any other non-magicable format in the future). Using SDL_RWops is not covered here, but they enable to load from almost any source.
IMG_LoadBMP_RW	Load src as a BMP image for use as a surface, if BMP support is compiled into the SDL_image library. A instance to the image as a new SDL_Surface. nil is returned on errors, like if BMP is not supported, or a read error.
IMG_LoadGIF_RW	Load src as a GIF image for use as a surface, if GIF support is compiled into the SDL_image library. A instance to the image as a new SDL_Surface. nil is returned on errors, like if GIF is not supported, or a read error.
IMG_LoadJPG_RW	Load src as a JPG image for use as a surface, if JPG support is compiled into the SDL_image library.If the image format loader requires initialization, it will attempt to do that the first time it is needed if it have not already called IMG_Init to load support for image format.A instance to the image as a new SDL_Surface. nil is returned on errors, like if JPG is not supported, or a read error.
IMG_LoadLBM_RW	Load src as a LBM image for use as a surface, if LBM support is compiled into the SDL_image library. A instance to the image as a new SDL_Surface. nil is returned on errors, like if LBM is not supported, or a read error.
IMG_LoadPCX_RW	Load src as a PCX image for use as a surface, if PCX support is compiled into the SDL_image library. A instance to the image as a new SDL_Surface. nil is returned on errors, like if PCX is not supported, or a read error.
IMG_LoadPNG_RW	Load src as a PNG image for use as a surface, if PNG support is compiled into the SDL_image library.A instance to the image as a new SDL_Surface. nil is returned on errors, like if PNG is not supported, or a read error.

IMG_LoadPNM_RW	Load src as a PNM image for use as a surface, if PNM support is compiled into the SDL_image library.A instance to the image as a new SDL_Surface. nil is returned on errors, like if PNM is not supported, or a read error.
IMG_LoadTGA_RW	Load src as a TGA image for use as a surface, if TGA support is compiled into the SDL_image library. It try to load a non TGA image, it might succeed even when it's not TGA image formatted data, this is because the TGA has no magic, which is a way of identifying a filetype from a signature in it's contents. So be careful with this. A instance to the image as a new SDL_Surface. nil is returned on errors, like if TGA is not supported, or a read error.
IMG_LoadTIF_RW	Load src as a TIF image for use as a surface, if TIF support is compiled into the SDL_image library.A instance to the image as a new SDL_Surface. nil is returned on errors, like if TIF is not supported, or a read error.
IMG_LoadXCF_RW	Load src as a XCF image for use as a surface, if XCF support is compiled into the SDL_image library. A instance to the image as a new SDL_Surface. nil is returned on errors, like if XCF is not supported, or a read error.
IMG_LoadXPM_RW	Load src as a XPM image for use as a surface, if XPM support is compiled into the SDL_image library.A instance to the image as a new SDL_Surface. nil is returned on errors, like if XPM is not supported, or a read error.
IMG_LoadXV_RW	Load src as a XV thumbnail image for use as a surface, if XV support is compiled into the SDL_image library.A instance to the image as a new SDL_Surface. nil is returned on errors, like if XV is not supported, or a read error.
IMG_ReadXPMFromArray	Load xpm as a XPM image for use as a surface, if XPM support is compiled into the SDL_image library. A instance to the image as a new SDL_Surface. nil is returned on errors, like if XPM is not supported, or a read error.
IMG_isBMP	The BMP image data is tested to see if it is readable as a BMP, otherwise it returns false (Zero). It return 1 if the image is a BMP and the BMP format support is compiled into SDL_image. 0 is returned otherwise.

IMG_isGIF	The GIF image data is tested to see if it is readable as a GIF, otherwise it returns false (Zero). It return 1 if the image is a GIF and the GIF format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isJPG	The JPG image data is tested to see if it is readable as a JPG, otherwise it returns false (Zero). It returns 1 if the image is a JPG and the JPG format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isLBM	The LBM image data is tested to see if it is readable as a LBM, otherwise it returns false (Zero). It returns 1 if the image is a LBM and the LBM format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isPCX	The PCX image data is tested to see if it is readable as a PCX, otherwise it returns false (Zero). It returns 1 if the image is a PCX and the PCX format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isPNG	The PNG image data is tested to see if it is readable as a PNG, otherwise it returns false (Zero). It return 1 if the image is a PNG and the PNG format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isPNM	The PNM image data is tested to see if it is readable as a PNM, otherwise it returns false (Zero). It returns 1 if the image is a PNM and the PNM format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isTIF	The TIF image data is tested to see if it is readable as a TIF, otherwise it returns false (Zero). returns 1 if the image is a TIF and the TIF format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isXCF	The XCF image data is tested to see if it is readable as a XCF, otherwise it returns false (Zero). It returns 1 if the image is a XCF and the XCF format support is compiled into SDL_image. 0 is returned otherwise.
IMG_isXPM	The XPM image data is tested to see if it is readable as a XPM, otherwise it returns false (Zero).

IMG_isXV	The XV image data is tested to see if it is readable as an XV thumbnail, otherwise it returns false (Zero). It return 1 if the image is an XV thumbnail and the XV format support is compiled into SDL_image. 0 is returned otherwise.
IMG_SetError	This is the same as SDL_SetError, which sets the error string which may be fetched with IMG_GetError (or SDL_GetError). This functions acts like printf, except that it is limited to SDL_ERRBUFIZE(1024) chars in length. It only accepts the following format types: %s, %d, %f, %p. No variations are supported, like %.2f would not work.
IMG_GetError	This is the same as SDL_GetError, which returns the last error set as a string which you may use to tell the user what happened when an error status has been returned from an SDL_image function call.A char instance (string) containing a humam readble version or the reason for the last error that occured.
IMG_Linked_Version	This works similar to SDL_Linked_Version and SDL_VERSION. Using these you can compare the runtime version to the version that you compiled with.These functions/macros do not require any library initialization calls before using them.

SDL Mixer	
Mix_QuerySpec	It gets the actual audio format in use by the opened audio device.This may or may not match the parameters which passed to Mix_OpenAudio().Returns 0 on error.
Mix_Linked_Version	It works similar to SDL_Linked_Version() and SDL_VERSION. Using these you can compare the runtime version to the version that you compiled with.Returns a constant instance to an SDL_version structure containing the version information
Mix_OpenAudio	It will initialize SDL_mixer if it is not yet initialized .SDL_mixer() may not be able to provide the exact specifications which provided, however it will automatically translate between the expected format and the real one.Returns 0 on success, -1 on error.When already initialized, this function will not re-initialize SDL_mixer(), nor fail
Mix_CloseAudio	It cleans up and then shuts down the Mixer API.After calling this API all audio is stopped, the device is closed, and the SDL_mixer functions should not be used.Return value none.
Mix_SetError	It is the same as SDL_SetError(), which sets the error string which may be fetched with Mix_GetError (or SDL_GetError()).
Mix_GetError	It is the same as SDL_GetError, which returns the last error set as a string which may use to tell the user what happened when an error status has been returned from an SDL_mixer() function call.Returns a char instance (string) containing a human readable version or the reason for the last error that occurred.
Mix_LoadWAV	It is a macro that creates an SDL_RWops then calls Mix_LoadWAV_RW() to load a WAV from a file. It supports WAVE-, MOD-, MIDI-, OGG- and MP3 files .Returns a Mix_Chunk containing the whole sample on success, or nil on error.
Mix_LoadWAV_RW	It is a macro that loads a sound sample from a a block of memory. It supports WAVE-, MOD-, MIDI-, OGG- and MP3 files .It accepts two arguments, the first being a instance to the RWops structure from which to read and the second being a flag to free the source memory after loading is complete or not.Returns a Mix_Chunk containing the whole sample on success, or nil on error.

Mix_VolumeChunk	It sets the volume of a chunk specified by chunk. The amount passed to volume can be between 0 and MIX_MAX_VOLUME which equals 128.Returns the previous volume of the chunk.
Mix_FreeChunk	It Frees the memory used in chunk, and free chunk itself as well. Do not use chunk after this without loading a new sample to it. It's a bad idea to free a chunk that is still being played.Return value none.
Mix_AllocateChannels	It will allocate the number of channels specified in number of channels (numchans).Returns the number of channels that it was able to allocate.
Mix_Volume	It changes the volume of the channel specified in channel by the amount set in volume(argument).The range of volume is from 0 to MIX_MAX_VOLUME which is 128. Passing -1 to channel will change the volume of all channels. If the specified volume is -1, it will just return the current volume.Returns the previously set volume of the channel.
Mix_PlayChannel	It will play the specified chunk over the specified channel(as argument). SDL_mixer() will choose a specific channel if the input is passed as -1 for channel.The chunk will be looped loops(argument) times, the total number of times played will be loops+1. Passing -1 will loop the chunk infinitely.Returns the channel the chunk will be played on, or -1 on error.
Mix_PlayChannelTimed	If the sample is long enough and has enough loops then the sample will stop after ticks milliseconds. Otherwise this function is the same as Mix_PlayChannel().Returns the channel the sample is played on. On any errors, -1 is returned.
Mix_FadeInChannel	It is like Mix_PlayChannel() but with a smooth fade in effect. -1 for channel will cause the sound to be played in any available channel. -1 in loops will cause the music to loop forever. The number of miliseconds for the fade in duration can be specified in the ms parameter of the function.Returns the channel that the chunk will be played on.
Mix_FadeInChannelTimed	If the sample is long enough and has enough loops then the sample will stop after ticks milliseconds. Otherwise this function is the same as Mix_FadeInChannel().Returns the channel the sample is played on. On any errors, -1 is returned.

Mix_Pause	It will pause the channel specified in channel. If -1 is passed as the parameter to channel, all channels will be paused.Returns nothing.
Mix_Resume	It will resume a channel that has been paused.Returns nothing.
Mix_HaltChannel	It will stop the channel specified in channel. If -1 is passed as the parameter to channel, all channels will be stopped.Returns 0 all the time.
Mix_ExpireChannel	It will stop the channel specified in channel after the time (in milliseconds) specified in ticks. Passing -1 in channels will cause all channels to expire.Returns the number of channels stopped.
Mix_FadeOutChannel	It fades out a channel specified in which with a duration specified in ms in milliseconds.Returns the the number of channels that will be faded out.
Mix_ChannelFinished	It will add a own callback when a channel has finished playing. nil to disable callback. The callback may be called from the mixer's audio callback or it could be called as a result of Mix_HaltChannel(), etc. do not call SDL_LockAudio() from this callback; it should either be inside the audio callback, or SDL_mixer will explicitly lock the audio before calling the callback.Returns void.
Mix_Playing	It tells if channel is playing, or not.Returns zero if the channel is not playing. Otherwise if input passed is -1, the number of channels playing is returned. If the input passed as a specific channel, then 1 is returned if it is playing.
Mix_Paused	It tells if channel is paused, or not.Returns zero if the channel is not paused. Otherwise if you passed in -1, the number of paused channels is returned. If the input passed as a specific channel, then 1 is returned if it is paused.
Mix_FadingChannel	It tells you if which channel is fading in, out, or not. It does not tell if the channel is playing anything, or paused, so that need to be tested separately.Returns the fading status. Never returns an error.
Mix_GetChunk	It gets the most recent sample chunk instance played on channel. This instance may be currently playing, or just the last used.Returns instance to the Mix_Chunk(). nil is returned if the channel is not allocated, or if the channel has not played any samples yet.

Mix_ReserveChannels	It reserve number of channels from being used when playing samples when passing in -1 as a channel number to playback functions. The channels are reserved starting from channel 0 to num-1. Passing in zero will unreserve all channels. Normally SDL_mixer() starts without any channels reserved .Returns the number of channels reserved. Never fails, but may return less channels than you ask for, depending on the number of channels previously allocated.
Mix_GroupCount	It counts the number of channels in group tag.Returns the number of channels found in the group. This function never fails.
Mix_GroupAvailable	It finds the first available (not playing) channel in group tag.Returns the channel found on success. -1 is returned when no channels in the group are available.
Mix_GroupOldest	It finds the oldest actively playing channel in group tag.Returns the channel found on success. -1 is returned when no channels in the group are playing or the group is empty.
Mix_GroupNewer	It finds the newest, most recently started, actively playing channel in group tag.Returns the channel found on success. -1 is returned when no channels in the group are playing or the group is empty.
Mix_FadeOutGroup	It gradually fade out channels in group tag over ms milliseconds starting from now. The channels will be halted after the fade out is completed. Only channels that are playing are set to fade out, including paused channels. Any callback set by Mix_ChannelFinished() will be called when each channel finishes fading out.Returns the number of channels set to fade out.
Mix_HaltGroup	It halt playback on all channels in group tag.Returns always zero.

Mix_RegisterEffect	It hook a processor function f into a channel for post processing effects. May be just reading the data and displaying it, or it may be altering the stream to add an echo. Most processors also have state data that they allocate as they are in use, this would be stored in the arg instance data space. When a processor is finished being used, any function passed into d will be called, which is when the processor should clean up the data in the arg data space. The effects are put into a linked list, and always appended to the end, meaning they always work on previously registered effects output. Effects may be added multiple times in a row. Effects are cumulative this way.Returns zero on errors, such as a nonexisting channel.
Mix_UnregisterEffect	It remove the oldest (first found) registered effect function f from the effect list for channel. This only removes the first found occurance of that function, so it may need to be called multiple times if you added the same function multiple times, just stop removing when Mix_UnregisterEffect() returns an error, to remove all occurances of f from a channel.If the channel is active the registered effect will have its Mix_EffectDone_t function called, if it was specified in Mix_RegisterEffect().Returns zero on errors, such as invalid channel, or effect function not registered on channel.
Mix_UnregisterAllEffects	It removes all effects registered to channel. If the channel is active all the registered effects will have their Mix_EffectDone_t functions called, if they were specified in Mix_RegisterEffect().Returns zero on errors, such as channel not existing.
Mix_LoadMUS	It Load music file to use. This can load WAVE, MOD, MIDI, OGG, MP3, FLAC, and any file that you use a command to play with. If an external command is used to play the music, then the Mix_SetMusicCMD() should be called before this, otherwise the internal players will be used. Alternatively, if an external command is setted up and don't want to use it, then this Mix_SetMusicCMD(nil) function should be called to use the built-in players again.Returns a instance to a Mix_Music. nil is returned on errors.

Mix_FreeMusic	It free the loaded music. If music is playing it will be halted. If music is fading out, then this function will wait (blocking) until the fade out is complete.Returns void.
Mix_PlayMusic	It play the loaded music loop times through from start to finish. The previous music will be halted, or if fading out it waits (blocking) for that to finish.Returns 0 on success, or -1 on errors.
Mix_FadeInMusic	It fade in over ms milliseconds of time, the loaded music, playing it loop times through from start to finish.The fade in effect only applies to the first loop. Any previous music will be halted, or if it is fading out it will wait (blocking) for the fade to complete.This function is the same as Mix_FadeInMusicPos(music, loops, ms, 0).Returns 0 on success, or -1 on errors.
Mix_FadeInMusicPos	It fade in over ms milliseconds of time, the loaded music, playing it loop times through from start to finish. It fade in effect only applies to the first loop. The first time the music is played, it posistion will be set to posistion, which means different things for different types of music files, Look Mix_SetMusicPosition() for more info on that.Any previous music will be halted, or if it is fading out it will wait (blocking) for the fade to complete.Returns 0 on success, or -1 on errors.
Mix_HookMusic	It sets up a custom music player function. The function will be called with arg passed into the udata parameter when the mix_func is called. The stream parameter passes in the audio stream buffer to be filled with len bytes of music. The music player will then be called automatically when the mixer needs it. Music playing will start as soon as this is called. All the music playing and stopping functions have no effect on music after this. Pause and resume will work. Using a custom music player and the internal music player is not possible, the custom music player takes priority. To stop the custom music player call Mix_HookMusic(nil, nil).Returns void.

Mix_VolumeMusic	It set the volume to volume, if it is 0 or greater, and return the previous volume setting. Setting the volume during a fade will not work, the faders use this function to perform their effect! Setting volume while using an external music player set by Mix_SetMusicCMD() will have no effect, and Mix_GetError() will show the reason why not. Returns the previous volume setting
Mix_PauseMusic	It pause the music playback. And halt paused music.Music can only be paused if it is actively playing.Returns void.
Mix_ResumeMusic	It unpause the music. This is safe to use on halted, paused, and already playing music.Returns void.
Mix_RewindMusic	It rewind the music to the start. This is safe to use on halted, paused, and already playing music. It is not useful to rewind the music immediately after starting playback, because it starts at the beginning by default.This function only works for these streams: MOD, OGG, MP3, Native MIDI.Returns void.
Mix_SetMusicPosition	It sets the position of the currently playing music. The position takes different meanings for different music sources. It only works on the music sources like MOD ,OGG and MP3.
Mix_SetMusicCMD	It setup a command line music player to use to play music.Any music playing will be halted. The music file to play is set by calling Mix_LoadMUS(filename), and the filename is appended as the last argument on the commandline. This allows you to reuse the music command to play multiple files.Returns 0 on success, or -1 on any errors, such as running out of memory.
Mix_HaltMusic	It halt playback of music. This interrupts music fader effects. Any callback set by Mix_HookMusicFinished() will be called when the music stops.Returns always zero.
Mix_FadeOutMusic	It gradually fade out the music over ms milliseconds starting from now. The music will be halted after the fade out is completed. Only when music is playing and not fading already are set to fade out, including paused channels. Any callback set by Mix_HookMusicFinished() will be called when the music finishes fading out.

Mix_HookMusicFinished	It sets up a function to be called when music playback is halted. Any time music stops, the music_finished function will be called. Call with nil to remove the callback.NOTE: NEVER call SDL_Mixer functions, nor SDL_LockAudio, from a callback function.Returns void.
Mix_GetMusicType	It tells the file format encoding of the music. This may be handy when used with Mix_SetMusicPosition(), and other music functions that vary based on the type of music being played. To find the type of music currently being played, pass in nil to music.Returns the type of music or if music is nil then the currently playing music type, otherwise MUS_NONE if no music is playing.
Mix_PlayingMusic	It tells the music is actively playing, or not.Does not check if the channel has been paused.Returns zero if the music is not playing, or 1 if it is playing.
Mix_PausedMusic	It tells the music is paused, or not.Does not check if the music was been halted after it was paused, which may seem a little weird.Returns zero if music is not paused. 1 if it is paused.
Mix_FadingMusic	It tells if the music is fading in, out, or not at all. Does not tell if the channel is playing anything, or paused, so this has to be tested separately.Returns the fading status. Never returns an error.
Mix_GetMusicHookData	It get the arg passed into Mix_HookMusic().Returns the arg instance.
Mix_SetPanning	This effect will only work on stereo audio. Meaning you called Mix_OpenAudio() with 2 channels (MIX_DEFAULT_CHANNELS).Returns zero on errors, such as bad channel, or if Mix_RegisterEffect() failed.
Mix_SetDistance	This effect simulates a simple attenuation of volume due to distance. The volume never quite reaches silence, even at max distance.Returns zero on errors, such as an invalid channel, or if Mix_RegisterEffect() failed.
Mix_SetPosition	This effect emulates a simple 3D audio effect. It's not all that realistic, but it can help improve some level of realism. By giving it the angle and distance from the camera's point of view, the effect pans and attenuates volumes.Returns zero on errors, such as an invalid channel, or if Mix_RegisterEffect() failed.

Mix_SetReverseStereo	Simple reverse stereo, swaps left and right channel sound.Returns zero on errors, such as an invalid channel, or if Mix_RegisterEffect() failed.

SDL_net	
SDLNet_Init	Start up SDL_net functionality.Initialize the network API.This must be called before using other functions in this library.SDL must be initialized before this call. It returns 0 on success, -1 on errors.
SDLNet_Quit	Shutdown and cleanup the network API.After calling this all sockets are closed, and the SDL_net functions should not be used.
SDLNet_GetError	This is the same as SDL_GetError, which returns the last error set as a string which is use to tell the user what happened when an error status has been returned from an SDLNet_function. It returns a char instace(string) containing a humam readble version or the reason for the last error that occured.
SDLNet_ResolveHost	Resolve the string host, and fill in the IPaddress pointed to by address with the resolved IP and the port number passed in through port.This is the best way to fill in the IPaddress struct for later use. This function does not actually open any sockets, it is used to prepare the arguments for the socket opening functions. This function will put the host and port into Network Byte Order into the address fields, so make sure to pass in the data in hosts byte order. It return 0 on success. -1 on errors, plus address.host will be INADDR_NONE. An error would likely be that the address could not be resolved.
SDLNet_ResolveIP	Resolve the IPv4 numeric address in address->host, and return the hostname as a string. It returns a valid char instance (string) on success. the returned hostname will have host and domain, as in "host.domain.ext". nil is returned on errors, such as when it's not able to resolve the host name. The returned instance is not to be freed. Each time you call this function the previous instance's data will change to the new value, so copy it into a local buffer to keep it around longer.

SDLNet_TCP Open	Connect to the host and port contained in ip using a TCP connection.If the host is INADDR_ANY, then only the port number is used, and a socket is created that can be used to later accept incoming TCP connections. It returns a valid TCPsocket on success, which indicates a successful connection has been established, or a socket has been created that is valid to accept incoming TCP connections. nil is returned on errors, such as when it's not able to create a socket, or it cannot connect to host and/or port contained in ip.
SDLNet_TCP Close	This shutsdown, disconnects, and closes the TCPsocket sock. Do not try to use any other functions on a closed socket, as it is now invalid.It returns nothing, this always succeeds for all we need to know.
SDLNet_TCP Accept	Accept a connection on a server socket. This function is non-blocking. It returns a valid TCPsocket on success, which indicates a successful connection has been established. nil is returned on errors, such as when it's not able to create a socket, or it cannot finish connecting to the originating host and port. There also may not be a connection attempt in progress, so of course you cannot accept nothing, and you get a nil in this case as well.
SDLNet_TCP GetPeerAddress	Get the remote host address and port number. It returns an IPaddress. nil is returned on errors, or when sock is a server socket.
SDLNet_TCP Send	Send data over a connected socket. It return the number of bytes sent. If the number returned is less than len, then an error occured, such as the client disconnecting.
SDLNet_TCP Recv	Receive data from a connected socket. This function does not wait for receiving maxlen bytes, it can returns before maxlen is reached. This function is blocking.
SDLNet_UDP Open	Create a UDP socket, if port is not 0, the bind is done in this function.If a non-zero port is given it will be used, otherwise any open port number will be used automatically.A valid UDPsocket on success. nil is returned on errors, such as when it's not able to create a socket, or it cannot assign the non-zero port as requested.
SDLNet_UDP Close	Close and free a UDP socket. This Api always sucessed

SDLNet_UDP Bind	Assign an IP address number to a socket channel.Incoming packets are only allowed from bound addresses for the socket channel.
SDLNet_UDP Unbind	Remove all assigned IP addresses from a socket channel. This removes all previously assigned (bound) addresses from a socket channel.Always success.
SDLNet_UDP GetPeerAddress	Get IP address for a socket channel or get the port you opened the socket with.An instance to an IPaddress. nil is returned for unbound channels and on any errors.
SDLNet_UDP Send	Send a UDPpacket. Send packet using the specified socket sock, use ing the specified channel or else the packet's address.The number of destinations sent to that worked. 0 is returned on errors.
SDLNet_UDP Recv	Receive into a UDPpacket. This function is nonblocking. 1 is returned when a packet is received. 0 is returned when no packets are received. -1 is returned on errors.
SDLNet_UDP SendV	Send a UDPpacket vector. Don't forget to set the length of the packets in the len element of the packets you are sending.It returns the number of destinations sent to that worked, for each packet in the vector, all summed up. 0 is returned on errors.
SDLNet_UDP RecvV	Receive into a UDPpacket vector.the number of packets received. 0 is returned when no packets are received. -1 is returned on errors.
SDLNet_AllocPacket	Allocate a new UDP packet with a data buffer. An instance to a new empty UDPpacket. NULL is returned on errors, such as out-of-memory.
SDLNet_ResizePacket	Resize the data buffer in a UDPpacket. Resize a UDPpackets data buffer to size bytes. The old data buffer will not be retained, so the new buffer is invalid after this call. It returns the new size of the data in the packet. If the number returned is less than what you asked for, that's an error.
SDLNet_FreePacket	Free a previously allocated UDPpacket. this always succeeds.
SDLNet_AllocPacketV	Allocate a vector of UDPpackets. An instance to a new empty UDPpacket vector. NULL is returned on errors, such as out-of-memory.
SDLNet_FreePacketV	Free a vector of UDPpackets. this always succeeds.

SDLNet_AllocSocketSet	Create a new socket set. Create a socket set that will be able to watch up to maxsockets number of sockets. The same socket set can be used for both UDP and TCP sockets. It return a new, empty, SDLNet_SocketSet. NULL is returned on errors, such as out-of-memory.
SDLNet_FreeSocketSet	Free a socket set. this call always succeeds.
SDLNet_AddSocket	Add a socket to a socket set.Add a socket to a socket set that will be watched. It return the number of sockets used in the set on success. -1 is returned on errors.
SDLNet_DelSocket	Remove a socket from a socket set.Remove a socket from a socket set. Use this before closing a socket that you are watching with a socket set. This doesn't close the socket. It return the number of sockets used in the set on success. -1 is returned on errors.
SDLNet_CheckSockets	Check and wait for sockets in a set to have activity.Check all sockets in the socket set for activity. If a non-zero timeout is given then this function will wait for activity, or else it will wait for timeout milliseconds. "activity" also includes disconnections and other errors, which would be determined by a failed read/write attempt. It return the number of sockets with activity. -1 is returned on errors, and you may not get a meaningful error message. -1 is also returned for an empty set (nothing to check).
SDLNet_SocketReady	See if a socket has activity.non-zero for activity. zero is returned for no activity.

SDL_ttf	
TTF_Init	Initialize the truetype font API. This must be called before using other functions in this library,except TTF_WasInit. SDL does not have to be initialized before this call.Returns 0 on success, -1 on any error
TTF_WasInit	Query the initilization status of the truetype font API. Used before TTF_Init to avoid initializing twice or to check if TTF_Quit call is required. Returns 1 if already initialized, 0 if not initialized.
TTF_Quit	Shutdown and cleanup the truetype font API. After calling this the SDL_ttf functions should not be used, excepting TTF_WasInit.
TTF_SetError	This is a defined macro for SDL_SetError, which sets the error string which may be fetched with TTF_GetError (or SDL_GetError). This functions acts like printf, except that it is limited to SDL_ERRBUFIZE(1024) chars in length.
TTF_GetError	This is a defined macro for SDL_GetError. It returns the last error set by TTF_SetError (or SDL_SetError) as a string. Used to tell the user what happened when an error status has been returned from an SDL_ttf function call.
TTF_OpenFont	Load file for use as a font, at ptsize size. Returns an instance to the font as a TTF_Font. nil is returned on errors.
TTF_OpenFontRW	Load src for use as a font, at ptsize size. The source is SDL_RWops as an instance. The font is loaded from this. A non-zero value of freesrc means it will automatically close and free the src after it finishes using the src, even if a noncritical error occured. Returns an instance of the font as a TTF_Font. nil is returned on errors.
TTF_OpenFontIndex	Load file, face index, for use as a font, at ptsize size. Returns an instance of the font as a TTF_Font. nil is returned on errors.

TTF_OpenFontIndexRW	Load src, face index, for use as a font, at ptsize size. This can load TTF and FON formats. Using SDL_RWops is not covered here, but they enable you to load from almost any source. Returns an instance of the font as a TTF_Font. nil is returned on errors. NOTE: src is not checked for nil.
TTF_CloseFont	Free the memory used by font, and free font itself as well. Do not use font after this without loading a new font to it.
TTF_ByteSwappedUNICODE	This function tells SDL_ttf whether UNICODE (Uint16 per character) text is generally byteswapped. A UNICODE_BOM_NATIVE or UNICODE_BOM_SWAPPED character in a string will temporarily override this setting for the remainder of that string, however this setting will be restored for the next one. The default mode is non-swapped, native endianness of the CPU. If swapped argument is non-zero then UNICODE data is byte swapped relative to the CPU's native endianness otherwise do not swap UNICODE data, use the CPU's native endianness.
TTF_GetFontStyle	Get the rendering style of the loaded font. Returns the style as a bitmask composed of the following masks: TTF_STYLE_BOLD TTF_STYLE_ITALIC TTF_STYLE_UNDERLINE TTF_STYLE_STRIKETHROUGH If no style is set then TTF_STYLE_NORMAL is returned. NOTE: Passing a nil font into this function will cause a segfault.
TTF_SetFontStyle	Set the rendering style of the loaded font. NOTE: Passing a nil font into this function will cause a segfault.
TTF_FontHeight	Get the maximum pixel height of all glyphs of the loaded font. NOTE: Passing a nil font into this function will cause a segfault.

TTF_FontAscent	Get the maximum pixel ascent of all glyphs of the loaded font. This can also be interpreted as the distance from the top of the font to the baseline. NOTE: Passing a nil font into this function will cause a segfault.
TTF_FontDescent	Get the maximum pixel descent of all glyphs of the loaded font. This can also be interpreted as the distance from the baseline to the bottom of the font. NOTE:Passing a nil font into this function will cause a segfault.
TTF_FontLineSkip	Get the recommended pixel height of a rendered line of text of the loaded font. This is usually larger than the TTF_FontHeight of the font. NOTE: Passing a nil font into this function will cause a segfault.
TTF_FontFaces	Get the number of faces ("sub-fonts") available in the loaded font. This is a count of the number of specific fonts (based on size and style and other typographical features perhaps) contained in the font itself. NOTE: Passing a nil font into this function will cause a segfault.
TTF_FontFaceIsFixedWidth	Test if the current font face of the loaded font is a fixed width font. Fixed width fonts are monospace, meaning every character that exists in the font is the same width, thus you can assume that a rendered string's width is going to be the result of a simple calculation:glyph_width * string_length. Returns >0 if font is a fixed width font and 0 if not a fixed width font. NOTE: Passing a nil font into this function will cause a segfault.
TTF_FontFaceFamilyName	Get the current font face family name from the loaded font. NOTE: Passing a nil font into this function will cause a segfault.
TTF_FontFaceStyleName	Get the current font face style name from the loaded font. NOTE: Passing a nil font into this function will cause a segfault.

TTF_GlyphMetrics	Get desired glyph metrics of the UNICODE chargiven in ch from the loaded font. Returns 0 on success, with all non-nil parameters set to the glyph metric as appropriate. -1 on errors, such as when the glyph does not exist in the font. NOTE: Passing a nil font into this function will cause a segfault.
TTF_SizeText	Calculate the resulting surface size of the LATIN1 encoded text rendered using font. No actual rendering is done, however correct kerning is done to get the actual width. The height returned is the same as you can get using TTF_FontHeight. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_SizeUTF8	Calculate the resulting surface size of the UTF8 encoded text rendered using font. No actual rendering is done, however correct kerning is done to get the actual width. The height returned is the same as you can get using TTF_FontHeight. Returns 0 on success, -1 is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_SizeUNICODE	Calculate the resulting surface size of the UNICODE encoded text rendered using font. No actual rendering is done, however correct kerning is done to get the actual width. The height returned in h is the same as you can get using TTF_FontHeight. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.

TTF_RenderText Solid	Render the LATIN1 encoded text using font with color onto a new surface, using the Solid mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderUTF8 Solid	Render the UTF8 encoded text using font with foreground color onto a new surface, using the Solid mode. The caller is responsible for freeing any returned surface. Returns aninstance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderUNICODE Solid	Render the UNICODE encoded text using font with fg color onto a new surface, using the Solid mode. The caller is responsible for freeing any returned surface. Returns aninstance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderGlyph Solid	Render the glyph for the UNICODE using font with foreground color onto a new surface, using the Solid mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors, such as when the glyph not available in the font. NOTE: Passing a nil font into this function will cause a segfault.

TTF_RenderText Shaded	Render the LATIN1 encoded text using font with foreground color onto a new surface filled with the background color, using the Shaded mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderUTF8 Shaded	Render the UTF8 encoded text using font with foreground color onto a new surface filled with the background color, using the Shaded mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderUNICODE Shaded	Render the UNICODE encoded text using font with foreground color onto a new surface filled with the background color, using the Shaded mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderGlyph Shaded	Render the glyph for the UNICODE using font with foreground color onto a new surface filled with the background color, using the Shaded mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors, such as when the glyph not available in the font. NOTE: Passing a nil font into this function will cause a segfault.

TTF_RenderText Blended	Render the LATIN1 encoded text using font with foreground color onto a new surface, using the Blended mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderUTF8 Blended	Render the UTF8 encoded text using font with foreground color onto a new surface, using the Blended mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderUNICODE Blended	Render the UNICODE encoded text using font with foreground color onto a new surface, using the Blended mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors. NOTE: Passing a nil font into this function will cause a segfault. NOTE: Passing a nil text into this function will result in undefined behavior.
TTF_RenderGlyph Blended	Render the glyph for the UNICODE using font with foreground color onto a new surface, using the Blended mode. The caller is responsible for freeing any returned surface. Returns an instance of a new SDL Surface. nil is returned on errors, such as when the glyph not available in the font. NOTE: Passing a nil font into this function will cause a segfault.

SDL_util	
SDL_putenv	The utility function sets the environment variable passed as parameter.
SDL_getenv	The utility function retrieves the environment variable.
bit_or	The utility function performs the bitwise OR operation on the operands.
bit_and	Bitwise AND value
bit_not	Bitwise NOT value
bit_xor	The utility function performs the bitwise XOR operation on the operands.
bit_lshift	Bitwise left shift value
bit_rshift	The utility function does the bitwise right shift operation on the operand.
new_int	The utility function does the memory allocation
delete_int	The utility function deletes the memory allocated for the ary element.
int_getitem	The utility function retrieves the array element at index location.
int_setitem	The utility function sets the array element at index location.

new_Uint8	The utility function does the memory allocation for an array of Uint8 elements.
delete_Uint8	The utility function deletes the memory allocated for the Uint8 ary element.
Uint8_getitem	The utility function retrieves the Uint8 array element at index location.
Uint8_setitem	Set the ‘value’ at index in given array. The array element size is one byte.
new_Uint16	Create an array of size ‘nelements’. The array element size is 2 byte.
delete_Uint16	Delete and free memory for given array.
Uint16_getitem	Return the element at given index in Uint16 array.
Uint16_setitem	Set the ‘value’ at index in given array. The array element is 2 byte long.
new_Uint32	Create an array of size ‘nelements’. The array element size is 4 byte.
delete_Uint32	Delete and free memory for given array.
Uint32_getitem	Return the element at given index in array.
Uint32_setitem	Set the ‘value’ at index in given array. The array element is 2 byte long.
new_bool	Create an array of size ‘nelements’. The array element is boolean.
Delete_bool	Delete and free memory for given array.
Bool_getitem	Element as per the index value (Note : Index should start from 1)
Bool_setitem	Setting the element of array as per corresponding index and value in the array
New_Uint8p	Initialize the unsigned integer reference. Allocate contiguous memory (block of - sizeof(Uint8)).
delete_Uint8p	Delete(Free) the allocated memory
uInt8p_assign	Assign the internation reference with corresponding value.
Uint8p_value	Retrieve the reference / instance

new_intp	Initialize the unsigned integer reference. Allocate contiguous memory.
delete_intp	Delete(Free) the allocated memory.
intp_assign	Assigns the value passed as a second argument to the address pointed by the the first argument(ptr)
intp_value	This function gets the value at specified address
new_Uint16p	This function allocates 16 bit of memory
Uint16p_value	This function returns the value pointed by the 16 bit integer address
Uint16p_assign	This function assigns the value passed as an argument to the address
delete_Uint16p	This function frees the memory of specified address
new_Uint32p	This function allocates a 32 bit memory
Uint32p_value	This function returns the value pointed by the memory address
Uint32p_assign	This function assigns the value passed as an argument to the memory address
delete_Uint32p	This function frees the specified address location
new_chardp	This function creates the double pointer for character array.
delete_chardp	This function frees the memory containing string
chardp_assign	This function assigns the value passed as second parameter to self which is a memory address that stores string.
chardp_value	This function returns the value from a memory address containing a string
new_Uint8dp	This function allocates memory for Uint8dp variable
delete_Uint8dp	This function frees the memory allocated for Uint8dp
Uint8dp_assign	This function assigns the value passed as second parameter to self which is a memory location for Uint8dp
Uint8dp_value	This function returns the value pointed to by the memory address forUint8 passed to it.
new_ptrDouble	This function creates a Double pointer and returns the address of this pointer.
delete_ptrDouble	This function delete the Double pointer pointed to by the parameter passed to this function
SDL_malloc	This function allocates memory of size bytes and returns a pointer to this memory address.

Unit8_to_char	This is a function used to change the type of a variable from Unit8 to char
char_to_Uint8	This is a function used to change the type of a variable from char to Unit8.
Uint32_to_Uint16	This is a function used to change the type of a variable from Uint32 to Uint 16..
Uint16_to_Uint32	This is a function used to change the type of a variable from Uint16 to Uint 32.
Unit8p_to_charp	This is a function used to change the type of a variable from Unit8 to charp.
charp_to_Uint8p	This is a function used to change the type of a variable from charp to Uint8p.
charp_to_voidp	This is a function used to change the type of a variable from charp to voidp.
voidp_to_charp	This is a function used to change the type of a variable from voidp to charp.
voidp_to_Uint8p	This is a function used to change the type of a variable from voidp to Uint8p.
Uint8p_to_voidp	This is a function used to change the type of a variable from Uint8p to voidp.
float_to_int	This is a function used to change the type of a variable from float to int.
intp_to_Uint8p	This is a function used to change the type of a variable from int to Uint8.
Uint8_to_int	This is a function used to change the type of a variable from Uint8 to int.
intp_to_Uint8p	This typecasts an integer pointer to Uint8 pointer
voidp_to_Uint16p	This typecasts a void pointer to a Uint16 pointer
Uint16p_to_voidp	This typecasts a Uint16 pointer to void pointer
voidp_to_Uint32p	This typecasts a void pointer to Uint32 pointer
Uint32p_to_voidp	This typecasts a Uint32 pointer to void pointer
voidp_to_intp	This typecasts a void pointer to integer pointer
intp_to_voidp	This typecasts an integer pointer to void pointer
SDL_RectpAr_getitem	This returns the pointer to SDL_Rect at location index in an array of SDL_Rect pointers
new_SDL_RectAr	This creates a new SDL_Rect array
delete_SDL_RectAr	This deletes the SDL_Rect array

SDL_RectAr_getitem	This returns the SDL_Rect at location index in the SDL_Rect array.
SDL_RectpAr_getitem	This returns the SDL_Rect pointer at location index in the SDL_Rect array.
SDL_free	The free() function shall cause the space pointed to by instance to be deallocated; that is, made available for further allocation.
SDL_calloc	The calloc() function shall allocate unused space for a table of nth elements each of whose size in bytes is elsize. The space shall be initialized to all bits 0.The order and contiguity of storage allocated by successive calls to calloc() is unspecified. The instance returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a instance to any type of object and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a instance to an object disjoint from any other object. The instance returned shall point to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a nil instance shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: the value returned shall be either a nil instance or a unique instance.
SDL_realloc	The *realloc*() function shall change the size of the memory object pointed to by *object* to the size specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new and old sizes. If the new size of the memory object would require movement of the object, the space for the previous instantiation of the object is freed. If the new size is larger, the contents of the newly allocated portion of the object are unspecified. If *size* is 0 and *object* is not a nil instance, the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged. The order and contiguity of storage allocated by successive calls to *realloc*() is unspecified. The instance returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a instance to any type of object and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a instance to an object disjoint from any other object. The instance returned shall point to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a nil pointer shall be returned.Error if insufficient memory is available.

SDL_memset	It is set bytes in memory .The memset() function copies c (converted to an unsigned char) into each of the first n bytes of the object pointed to by s. The memset() function returns s; no return value is reserved to indicate an error.
SDL_memcpy	The memcpy() function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. The memcpy() function shall return s1; no return value is reserved to indicate an error.
SDL_memmove	The memmove() function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. Copying takes place as if the n bytes from the object pointed to by s2 are first copied into a temporary table of n bytes that does not overlap the objects pointed to by s1 and s2, and then the n bytes from the temporary array are copied into the object pointed to by s1. The memmove() function shall return s1; no return value is reserved to indicate an error.
SDL_memcmp	The memcmp() function shall compare the first n bytes (each interpreted as unsigned char) of the object pointed to by s1 to the first n bytes of the object pointed to bys2. The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type unsigned char) that differ in the objects being compared. The memcmp() function shall return an integer greater than, equal to, or less than 0, if the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2, respectively.
SDL_strlen	The strlen() function shall compute the number of bytes in the string to which s points, not including the terminating null byte. The strlen() function shall return the length of s; no return value shall be reserved to indicate an error.
SDL_strlcpy	size-bounded string copying. The strlcpy() function copies up to size - 1 characters from the NUL- terminated string src to dst, nil-terminating the result.
SDL_strlcat	size-bounded string concatenation. The strlcat() function appends the NUL-terminated string src to the end of dst. It will append at most size - strlen(dst) - 1 bytes, nil-terminating the result.
SDL_strdup	The strdup() function shall return an instance to a new string, which is a

	duplicate of the string pointed to by s. The returned pointer can be passed to free(). A null pointer is returned if the new string cannot be created. These functions shall fail if:Storage space available is insufficient.
SDL_strrev	Reverse the string
SDL_strupr	Make the string to upper case.
SDL_strlwr	Make the string to lower case.
SDL_strchr	The strchr() function shall locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string. Upon completion, strchr() shall return a pointer to the byte, or a null pointer if the byte was not found.
SDL_strrchr	The strrchr() function shall locate the last occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string. Upon successful completion, strrchr() shall return a pointer to the byte or a null pointer if c does not occur in the string.
SDL_strstr	The strstr() function shall locate the first occurrence in the string pointed to by s1 of the sequence of bytes (excluding the terminating null byte) in the string pointed to bys2. Upon successful completion, strstr() shall return a pointer to the located string or a null pointer if the string is not found.If s2 points to a string with zero length, the function shall return s1.
SDL_itoa	Value to be converted to a string
SDL_ltoa	The value to convert into a string.
SDL_strcmp	The strcmp() function shall compare the string pointed to by s1 to the string pointed to by s2. The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type unsigned char) that differ in the strings being compared. Upon completion, strcmp() shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2, respectively.
SDL_strncmp	The strncmp() function shall compare not more than n bytes (bytes that follow a null byte are not compared) from the array pointed to by s1 to the array pointed to by s2. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as

	type unsigned char) that differ in the strings being compared. Upon successful completion, strncmp() shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2 respectively.
SDL_strcasecmp	The strcasecmp() function compares, while ignoring differences in case, the string pointed to by s1 to the string pointed to by s2. The strncasecmp() function compares, while ignoring differences in case, not more than n bytes from the string pointed to by s1 to the string pointed to by s2.
SDL_strncasecmp	case-insensitive string comparison.
SDL_memset4	Setting up the memory.

13.2. GLUE Layer API Manual

13.2.1. SDL_Init

Name	SDL_Init
Synopsis	require('SDL') SDL_Init(flag);
Description	Initializes SDL. This should be called before all other SDL functions. The flags parameter specifies what part(s) of SDL to initialize. Refer below table for flag parameter values.
Return Value	Returns -1 on an error or 0 on success.
Parameters	flag [in] : SDL initialization flags

Flag Parameter Values :

SDL_INIT_TIMER	Initializes the timer subsystem.
SDL_INIT_AUDIO	Initializes the audio subsystem
SDL_INIT_VIDEO	Initializes the video subsystem
SDL_INIT_CDROM	Initializes the cdrom subsystem
SDL_INIT_JOYSTICK	Initializes the joystick subsystem
SDL_INIT_EVERYTHING	Initialize all of the above
SDL_INIT_NOPARACHUTE	Prevents SDL from catching fatal signals
SDL_INIT_EVENTTHREAD	Runs the event manager in a separate thread

Example:

```
SDL_Init(SDL_INIT_VIDEO)
```

13.2.2. SDL_InitSubSystem

Name	SDL_InitSubSystem
Synopsis	require('SDL') SDL_InitSubSystem(flag);
Description	Initializes SDL. This should be called before all other SDL functions. The flags parameter specifies what part(s) of SDL to initialize.
Return Value	Returns -1 on an error or 0 on success.
Parameters	flag [in] : SDL initialization flags

Example :

```
SDL_Init(SDL_INIT_VIDEO)
.
SDL_SetVideoMode(640, 480, 16, SDL_ANYFORMAT)
.
-- Do Some Video stuff
.
-- Initialize the timer subsystem
SDL_InitSubSystem(SDL_INIT_TIMER)
-- Do some stuff with video and joystick
.
-- Shut them both down
SDL_Quit()
```

13.2.3. SDL_QuitSubSystem

Name	SDL_QuitSubSystem
Synopsis	require('SDL') SDL_QuitSubSystem(flag);
Description	SDL_QuitSubSystem allows you to shut down a subsystem that has been previously initialized by SDL_Init or SDL_InitSubSystem. The flags tells SDL_QuitSubSystem which subsystems to shut down, it uses the same values that are passed to SDL_Init.
Parameters	flag [in] : SDL initialization flags

Example :

```
SDL_Init(SDL_INIT_VIDEO)
SDL_SetVideoMode(640, 480, 16, SDL_ANYFORMAT)
.
SDL_QuitSubSystem(SDL_INIT_VIDEO)
```

13.2.4. SDL_Quit

Name	SDL_Quit
Synopsis	require('SDL') SDL_Quit();
Description	SDL_Quit shuts down all SDL subsystems and frees the resources allocated to them. This should always be called before you exit.
Parameters	nothing

Example :

```
SDL_Init(SDL_INIT_VIDEO)
SDL_SetVideoMode(640, 480, 16, SDL_ANYFORMAT)
……
SDL_Quit()
```

13.2.5. SDL_WasInit

Name	SDL_WasInit
Synopsis	require('SDL') SDL_WasInit(flags);
Description	SDL_WasInit allows you to see which SDL subsytems have been initialized. flags is a bitwise OR'd combination of the subsystems you wish to check (see SDL_Init for a list of subsystem flags).
Return Value	SDL_WasInit returns a bitwised OR'd combination of the initialized subsystems.
Parameters	flags [in] : SDL initialization flags

Example :

```
SDL_Init(SDL_INIT_TIMER)
if SDL_WasInit(0)== 1 then
        print("Correct subsystem initialized")
else
        print("Wrong subsystem initialized")
end
SDL_Quit()
```

13.2.6. SDL_GetError

Name	SDL_GetError
Synopsis	require('SDL') SDL_GetError(void);
Description	Returns a null terminated string containing information about the last internal SDL error
Return Value	Returns a null terminated string which represents the last error. This string is statically allocated and must not be freed by the user.
Parameters	nothing

Example :

```
screen = SDL_GetVideoSurface()
if screen == nil then
        print("Error in SDL_GetVideoSurface()"..SDL_GetError())
end
```

13.2.7. SDL_SetError

Name	SDL_SetError
Synopsis	require('SDL') SDL_SetError(fmt, …);
Description	Sets the SDL error to a printf style formatted string.
Return Value	nothing
Parameters	Constant string

Example :

```
buffer[BUFSIZ];
SDL_memset(buffer ,'1',BUFSIZ);
Buffer[BUFSIZ] = 0;
SDL_SetError("This is the error "..buffer..1.0);
print("Error 1: "..SDL_GetError());
if (tostring(SDL_GetError()) ~= 0 then
    print("SDL_Error:"..SDL_GetError())

end
```

13.2.8. SDL_Error

Name	SDL_Error
Synopsis	require('SDL') SDL_Error(code);
Description	Sets the SDL error message to the predefined string specified by code.
Return Value	nothing
Parameters	code[in]

SDL Error Codes

SDL_errorcode	The corresponding error string
SDL_ENOMEM	Out of memory
SDL_EFREAD	Error reading from datastream
SDL_EFWRITE	Error writing to datastream
SDL_EFSEEK	Error seeking in datastream
SDL_UNSUPPORTED	Unknown SDL error
SDL_LASTERROR	Unknown SDL error
any other value	Unknown SDL error

Example :

```
local ptr = SDL_malloc (1024);
```

```
if ptr == nil then
        print("Cant able to allocate memory "..SDL_Error(SDL_ENOMEM))
end
```

13.2.9. SDL_ClearError

Name	SDL_Error
Synopsis	require(‘SDL’) SDL_ClearError(void);
Description	Deletes all information about the last internal SDL error. Useful if the error has been handled by the program.
Return Value	nothing
Parameters	nothing

Example :

```
if (tostring(SDL_GetError())) ~= 0 then
        print("SDL_Error "..SDL_GetError())
SDL_ClearError();
end
```

13.2.10. SDL_version

Name	SDL_version
Synopsis	require('SDL') typedef struct { Uint8 major; Uint8 minor; Uint8 patch; } SDL_version;
Description	The `SDL_version` structure is used by the SDL_Linked_Version function and the SDL_VERSION macro.
Return Value	nothing
Parameters	The pointer to the SDL_version structure to fill. It must point to a valid memory location.

13.2.11. SDL_VERSION (macro)

Name	SDL_VERSION
Synopsis	require('SDL') SDL_VERSION(pointer);
Description	The SDL_VERSION macro is a helper macro that fills out a SDL_version structure with the compile-time SDL version.
Return Value	nothing
Parameters	pointer [in] The pointer to the SDL_version structure to fill. It must point to a valid memory location.

13.2.12. SDL_Linked_Version

Name	SDL_Linked_Version
Synopsis	require('SDL') SDL_Linked_Version();
Description	The SDL_Linked_Version function gets the version of the current dynamically linked SDL library and returns a pointer to a SDL_versionstructure containing the version information.
Return Value	Returns a constant pointer to an SDL_version structure containing the version information.
Parameters	nothing

Example :

```
local v = SDL_version_new()
v = SDL_Linked_Version()
print("Version "..v.major..v.minor..v.patch)
```

13.2.13. SDL_GetVideoSurface

Name	SDL_GetVideoSurface
Synopsis	require('SDL') SDL_Surface SDL_GetVideoSurface();
Description	This function returns current display surface. If SDL is doing format conversion on the display surface, this function returns the publicly visible surface, not the real video surface.
Return Value	NULL On error The pointer to the current display surface On success. The caller must not free the returned pointer.
Parameters	nothing

Example :

```
screen = SDL_GetVideoSurface()
if screen == nil then
        print("Error in SDL_GetVideoSurface()"..SDL_GetError())
end
```

13.2.14. SDL_GetVideoInfo

Name	SDL_GetVideoInfo
Synopsis	require('SDL') SDL_VideoInfo SDL_GetVideoInfo();
Description	This function returns a read-only pointer to information about the video hardware.
Return Value	Returns a read-only pointer to the structure containing the information about the current video hardware. Technically, it's a constantSDL_VideoInfo pointer.
Parameters	nothing

Example :

```
info = SDL_GetVideoInfo()
print("Current display: " ..info.current_w .."x" ..info.current_h ..","  .. info.vfmt.BitsPerPixel .."bits-per-pixel")
```

13.2.15. SDL_VideoDriverName

Name	SDL_VideoDriverName
Synopsis	require('SDL') SDL_VideoDriverName(namebuf, maxlen);
Description	The buffer pointed to by namebuf is filled up to a maximum of maxlen characters (include the NIL terminator) with the name of the initialised video driver. The driver name is a simple one word identifier like "x11" or "windib".
Return Value	Returns NIL if video has not been initialised with SDL_Init or a pointer to namebuf otherwise.
Parameters	namebuf [out] : The pointer to the character buffer which is previously allocated by the caller maxlen [in] : The maximum characters to fill in the character buffer namebuf

Example :

```
local driver = ' '
if SDL_VideoDriverName(driver,10) ~= nil then
        print("Video driver: "..driver)
end
```

13.2.16. SDL_ListModes

Name	SDL_ListModes
Synopsis	require('SDL') SDL_Rect SDL_ListModes(format, flags);
Description	Return a pointer to an array of available screen dimensions for the given format and video flags, sorted largest to smallest. Returns NIL if there are no dimensions available for a particular format,or -1 if any dimension is okay for the given format. If format is NIL, the mode list will be for the format returned by SDL_GetVideoInfo()->vfmt.The flag parameter is an OR'd combination of surface flags. The flags are the same as those used SDL_SetVideoMode and they play a strong role in deciding what modes are valid.
Return Value	NULL There is not any mode available for the particular format -1 Any dimension is okay for the given format The pointer to the array of SDL_Rect On success. The pointer to the video dimensions array is managed by SDL itself. The caller must not free the returned pointer.
Parameters	format [in] : SDL pixel format flags [in] : SDL flags

Example :

```
modes = SDL_ListModes(nil, SDL_FULLSCREEN)
i = 1
repeat
	modd = SDL_RectpAr_getitem(modes, i)
	i = i + 1
	if modd ~=nil then
		print("		" ..modd.w .."x" ..modd.h.."x" ..info.vfmt.BitsPerPixel)
	end
until modd == nil
```

13.2.17. SDL_VideoModeOK

Name	SDL_VideoModeOK
Synopsis	require('SDL') SDL_VideoModeOK(width, height, bpp, flags);
Description	SDL_VideoModeOK returns 0 if the requested mode is not supported under any bit depth, or returns the bits-per-pixel of the closest available mode with the given width, height and requested surface flags (see SDL_SetVideoMode). The bits-per-pixel value returned is only a suggested mode. You can usually request and bpp you want when setting the video mode and SDL will emulate that color depth with a shadow video surface.The arguments to SDL_VideoModeOK are the same ones you would pass to SDL_SetVideoMode
Return Value	0 The requested mode is not supported under any bit depth bits per pixel The function returned bits per pixel of the closest available mode with the given parameters. The bits per pixel value returned is only a suggested mode. You can usually request any bpp you want when setting the video mode and SDL will emulate that color depth with a shadow video surface.
Parameters	width, height [in] : The test video resolution in pixels bpp [in] : The test video bits per pixel flags [in] : The test video flags.

Example :

```
local retMode = SDL_VideoModeOK(940,480,32,SDL_SWSURFACE)
print("Value of SDL_VideoModeOK = " ..retMode)
```

```
if retMode == 0 then
        print("Mode Not available.")
end
```

13.2.18. SDL_SetVideoMode

Name	SDL_SetVideoMode
Synopsis	require('SDL') SDL_SetVideoMode(width, height, bpp, flags);
Description	Set up a video mode with the specified width, height and bits-per-pixel. If bpp is 0, it is treated as the current display bits per pixel. The flags parameter is the same as the flags field of the SDL_Surface structure. OR'd combinations of the following values are valid.Refer below table for flag parameter values.
Return Value	The framebuffer surface, or NIL if it fails.
Parameters	width, height [in] : The video resolution in pixels bpp [in] : video bits per pixel flags [in] : video flags.

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32, bit_or(SDL_SWSURFACE,
                                        SDL_ANYFORMAT))

if screen == nil then
                print("Error in SDL_SetVideoMode() API"..SDL_GetError())
end
```

13.2.19. SDL_UpdateRect

Name	SDL_UpdateRect
Synopsis	require('SDL') SDL_UpdateRect(screen, x, y, w, h);
Description	Makes sure the given area is updated on the given screen.If 'x', 'y', 'w' and 'h' are all 0, SDL_UpdateRect will update the entire screen. This function should not be called while 'screen' is locked.
Return Value	nothing
Parameters	screen [in] : SDL surface x, y [in] : Rectangle's coordinates w, h [in] : Rectangle's width & height.

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
                    bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
bmp = SDL_LoadBMP("sample.bmp")
dst = SDL_Rect_new()
dst.x = 0
dst.y = 0
dst.w = bmp.w
dst.h = bmp.h
```

```
src = SDL_Rect_new()
src.x = 0
src.y = 0
src.w = 200
src.h = 540

SDL_BlitSurface(bmp, src, screen, dst)
SDL_UpdateRect(screen,0,0,0,0)
```

13.2.20. SDL_UpdateRects

Name	SDL_UpdateRects
Synopsis	require('SDL') SDL_UpdateRects(screen, num_rects,rects);
Description	Makes sure the given list of rectangles is updated on the given screen. The rectangles must all be confined within the screen boundaries because there's no clipping. Note 1: This function should not be called while screen is locked by SDL_LockSurface Note 2: It is advised to call this function only once per frame, since each call has some processing overhead. This is no restriction since you can pass any number of rectangles each time.
Return value	nothing
Parameters	screen [in] The pointer to the SDL_Surface to update. This SDL surface should not be locked. See SDL_LockSurface numrects [in] The number of rectangles in the rects array rects [in] The array of SDL_Rect which describe the rectangles of screen to update. The rectangles are not automatically merged or checked for overlap. The caller should handle them carefully to avoid overdraw. The rectangles must

	be confined within the screenboundaries otherwise they can cause unexpected behaviors.

Example:

```
local screen
srcRect = SDL_Rect_new()
srcRect.w=200
srcRect.h=200
SDL_UpdateRects(screen,1,srcRect)
```

13.2.21. SDL_DisplayFormat

Name	SDL_DisplayFormat
Synopsis	require('SDL') SDL_DisplayFormat(surface)
Description	This function takes a surface and copies it to a new surface of the pixel format and colors of the video framebuffer, suitable for fast blitting onto the display surface. It calls SDL_ConvertSurface. If you want to take advantage of hardware colorkey or alpha blit acceleration, you should set the colorkey and alpha value before calling this function.
Return Value	This function returns SDL surface if successful, or nil if there was an error.
Parameters	surface [in] : SDL surface

Example:

```
NewSurface = SDL_DisplayFormat(surface);
SDL_FreeSurface(surface);
```

13.2.22. SDL_Flip

Name	SDL_Flip
Synopsis	require('SDL') SDL_Flip(screen);
Description	On hardware that supports double-buffering, this function sets up a flip and returns. The hardware will wait for vertical retrace, and then swap video buffers before the next video surface blit or lock will return. On hardware that doesn't support double-buffering, this is equivalent to calling SDL_UpdateRect(screen, 0, 0, 0, 0) The SDL_DOUBLEBUF flag must have been passed to SDL_SetVideoMode, when setting the video mode for this function to perform hardware flipping.
Return Value	This function returns 0 if successful, or -1 if there was an error.
Parameters	screen [in] : SDL surface

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
                    bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
bmp = SDL_LoadBMP("sample.bmp")
SDL_BlitSurface(bmp, src, screen, dst)
```

```
if SDL_Flip(screen) == -1 then
        print("Flip Error")
end
```

13.2.23. SDL_MapRGB

Name	SDL_MapRGB
Synopsis	require('SDL') SDL_MapRGB(fmt, r, g, b);
Description	Maps the RGB color value to the specified pixel format and returns the pixel value as a 32-bit .If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned. If the specified pixel format has an alpha component it will be returned as all 1 bits (fully opaque).
Return Value	A pixel value best approximating the given RGB color value for a given pixel format. If the pixel format bpp (color depth) is less than 32-bpp then the unused upper bits of the return value can safely be ignored
Parameters	fmt [in] : pixel format r [in] : red g [in] : green b [in] : blue

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32, \
                                bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
local Background = SDL_MapRGB(screen.format, 255, 255, 55)
print("Value of MAPRGB = " ..DEC_HEX(Background))
```

13.2.24. SDL_MapRGBA

Name	SDL_MapRGBA
Synopsis	require('SDL') SDL_MapRGBA(fmt, r, g, b, a);
Description	Maps the RGBA color value to the specified pixel format and returns the pixel value as a 32-bit . If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned. If the specified pixel format has no alpha component the alpha value will be ignored (as it will be in formats with a palette).
Return Value	A pixel value best approximating the given RGBA color value for a given pixel format. If the pixel format bpp (color depth) is less than 32-bpp then the unused upper bits of the return value can safely be ignored
Parameters	fmt [in] : pixel format r [in] : red g [in] : green b [in] : blue a [in] : alpha

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
                                bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
local Background = SDL_MapRGBA(screen.format, 255, 255, 55,120)
print("Value of MAPRGB = "..DEC_HEX(Background))
```

13.2.25. SDL_GetRGB

Name	SDL_GetRGB
Synopsis	require('SDL') SDL_GetRGB(fmt, r, g, b);
Description	Get RGB component values from a pixel stored in the specified pixel format.This function uses the entire 8-bit [0..255] range when converting color components from pixel formats with less than 8-bits per RGB component (e.g., a completely white pixel in 16-bit RGB565 format would return [0xff, 0xff, 0xff] not [0xf8, 0xfc, 0xf8]).
Return Value	nothing
Parameters	pixel [in] : pixel fmt [in] : pixel format r [out] : red g [out] : green b [out] : blue

Example:

```
SDL_MapRGB(screen.format, 255, 255, 55)
--
local r = new_Uint8p()
local g = new_Uint8p()
local b = new_Uint8p()
```

SDL_GetRGB(Background, screen.format, r, g, b)

```
print("Red = ",Uint8p_value(r))
print("Green = ",Uint8p_value(g))
print("Blue = ",Uint8p_value(b))
```

13.2.26. SDL_GetRGBA

Name	SDL_GetRGBA
Synopsis	require('SDL') SDL_GetRGBA(pixel, fmt, r, g, b, a);
Description	Get RGBA component values from a pixel stored in the specified pixel format.This function uses the entire 8-bit [0..255] range when converting color components from pixel formats with less than 8-bits per RGB component (e.g., a completely white pixel in 16-bit RGB565 format would return [0xff, 0xff, 0xff] not [0xf8, 0xfc, 0xf8]). If the surface has no alpha component, the alpha will be returned as 0xff (100% opaque).
Return Value	nothing
Parameters	pixel [in] : pixel fmt [in] : pixel format r [out] : red g [out] : green b [out] : blue a [out] : alpha

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
                                        bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
local Background = SDL_MapRGBA(screen.format, 255, 255, 55,120)
--
--
local r = new_Uint8p()
local g = new_Uint8p()
local b = new_Uint8p()
local a = new_Uint8p()

SDL_GetRGBA(Background, screen.format, r, g, b ,a)

print("Red = ",Uint8p_value(r))
print("Green = ",Uint8p_value(g))
print("Blue = ",Uint8p_value(b))
print("Alpha = ",Uint8p_value(a))
```

13.2.27. SDL_LoadBMP

Name	SDL_LoadBMP
Synopsis	require('SDL') SDL_LoadBMP(file);
Description	Loads a surface from a namedWindows BMP file.
Return Value	Returns the new surface, or NIL if there was an error.
Parameters	file[in]: window BMP file name

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
                                    bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
picture = SDL_LoadBMP("sample.bmp")
if picture == nil then
        print("Couldn't load bmpfile: "..SDL_GetError())
end
```

13.2.28. SDL_SetColorKey

Name	SDL_SetColorKey
Synopsis	require('SDL') SDL_SetColorKey(surface, flag, key);
Description	Sets the color key (transparent pixel) in a blittable surface and enables or disables RLE blit acceleration.RLE acceleration can substantially speed up blitting of images with large horizontal runs of transparent pixels (i.e., pixels that match the key value). The key must be of the same pixel format as the surface, SDL_MapRGB is often useful for obtaining an acceptable value. If flag is SDL_SRCCOLORKEY then key is the transparent pixel value in the source image of a blit. If flag is OR'd with SDL_RLEACCEL then the surface will be draw using RLE acceleration when drawn with SDL_BlitSurface. The surface will actually be encoded for RLE acceleration the first time SDL_BlitSurface or SDL_DisplayFormat is called on the surface. If flag is 0, this function clears any current color key.
Return Value	This function returns 0, or -1 if there was an error.
Parameters	surface [in] : SDL surface flag[in]:Color key Flag key[in]: Color key

Example:

```
bmpcc = SDL_LoadBMP("sample.bmp")
if bmpcc == nil then
        print("Couldn't load sample.bmp: "..SDL_GetError())
        return 0
end

if SDL_SetColorKey(bmpcc, bit_or(SDL_SRCCOLORKEY ,
                        SDL_RLEACCEL), 10 ) == -1 then
        print("Error in ColorKey API:"..SDL_GetError())
end
```

13.2.29. SDL_SetClipRect

Name	SDL_SetClipRect
Synopsis	require('SDL') SDL_SetClipRect(surface, rect);
Description	Sets the clipping rectangle for a surface. When this surface is the destination of a blit, only the area within the clip rectangle will be drawn into.The rectangle pointed to by rect will be clipped to the edges of the surface so that the clip rectangle for a surface can never fall outside the edges of the surface. If rect is NIL the clipping rectangle will be set to the full size of the surface.
Return Value	nothing
Parameters	surface [in] : SDL surface rect[in]: SDL rect

Example :

```
dstrect = SDL_Rect_new()
dst = SDL_Rect_new()
--
--Assign x,y,w,h, values
--
SDL_SetClipRect(screen,dstrect)
```

```
SDL_GetClipRect(screen,dst)
print(dst.x,dst.y,dst.w,dst.h)
```

13.2.30. SDL_GetClipRect

Name	SDL_GetClipRect
Synopsis	require('SDL') SDL_GetClipRect(surface, rect);
Description	Gets the clipping rectangle for a surface. When this surface is the destination of a blit, only the area within the clip rectangle is drawn into. The rectangle pointed to by rect will be filled with the clipping rectangle of the surface.
Parameters	surface [in] : SDL surface rect[out]: SDL rect

Example:

```
-- Example available in SDL_SetClipRect()
```

13.2.31. SDL_BlitSurface

Name	SDL_BlitSurface
Synopsis	require('SDL') SDL_BlitSurface(src, srcrect, dst, dstrect);
Description	This performs a fast blit from the source surface to the destination surface. Only the position is used in the dstrect (the width and height are ignored). If either srcrect or dstrect are NIL, the entire surface (src or dst) is copied. The final blit rectangle is saved in dstrect after all clipping is performed (srcrect is not modified). The blit function should not be called on a locked surface.The results of blitting operations vary greatly depending on whether SDL_SRCAPLHA is set or not. See SDL_SetAlpha for an explaination of how this effects the results.
Return Value	If the blit is successful, it returns 0, otherwise it returns -1. If either of the surfaces were in video memory, and the blit returns -2, the video memory was lost, so it should be reloaded with artwork and re-blitted:
Parameters	src[in]:Source SDL surface srcrect[in]:source SDL rect dst[in]: Destination SDL surface dstrect[in]:destination SDL rect

Example:

```
require(“SDL”)
local screen = SDL_SetVideoMode(960, 540, 32, bit_or(SDL_SWSURFACE,
                                        SDL_ANYFORMAT))
dest = SDL_Rect_new()

-- Set X,Y,W,H values
picture = SDL_LoadBMP("sample.bmp")
if picture == nil then
        print("Couldn't load bmpfile: "..SDL_GetError())
end

if SDL_BlitSurface(picture, nil, screen, dest) < 0 then
        print("Blit failed: ", SDL_GetError())
end
```

13.2.32. SDL_FillRect

Name	SDL_FillRect
Synopsis	require('SDL') SDL_FillRect(dst, dstrect, color);
Description	This function performs a fast fill of the given rectangle with color. If dstrect is NIL, the whole surface will be filled with color.The color should be a pixel of the format used by the surface, and can be generated by the SDL_MapRGB function. If there is a clip rectangle set on the destination (set via SDL_SetClipRect) then this function will clip based on the intersection of the clip rectangle and the dstrect rectangle.
Return Value	This function returns 0 on success, or -1 on error.
Parameters	dst[in]: SDL surface dstrect[in] : SDL rect color[in] : pixel of the format used by the surface

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32, bit_or(SDL_SWSURFACE,
SDL_ANYFORMAT))
dstrect = SDL_Rect_new()
dstrect.x = 50
dstrect.y = 32
```

dstrect.w = 500
dstrect.h = 500

SDL_FillRect(screen, dstrect, SDL_MapRGB(screen.format, r, g, b))

13.2.33. SDL_SaveBMP

Name	SDL_SaveBMP
Synopsis	require('SDL') SDL_SaveBMP(surface, file);
Description	Saves the SDL_Surface surface as a Windows BMP file named file.
Return Value	Returns 0 if successful or -1 if there was an error.
Parameters	surface [in] : SDL surface file[in]: Save window BMP file name

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
                                    bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
SDL_FillRect(screen, nil, SDL_MapRGB(screen.format, 255, 0, 0))
SDL_Flip(screen)
if SDL_SaveBMP(screen,"red.bmp") ~= 0 then
        print("Error saving red bmp"..SDL_GetError())
end
```

13.2.34. SDL_SetPalette

Name	SDL_SetPalette
Synopsis	require('SDL') SDL_SetPalette(surface, flags, colors, firstcolor, ncolors);
Description	Sets a portion of the palette for the given 8-bit surface. Palettized (8-bit) screen surfaces with the SDL_HWPALETTE flag have two palettes, a logical palette that is used for mapping blits to/from the surface and a physical palette (that determines how the hardware will map the colors to the display). SDL_BlitSurface always uses the logical palette when blitting surfaces (if it has to convert between surface pixel formats). This function can modify either the logical or physical palette by specifing SDL_LOGPAL or SDL_PHYSPALthe in the flags parameter. When surface is the surface associated with the current display, the display colormap will be updated with the requested colors. If SDL_HWPALETTE was set in SDL_SetVideoMode flags, SDL_SetPalette will always return 1, and the palette is guaranteed to be set the way you desire,even if the window colormap has to be warped or run under emulation.
Return Value	If surface is not a palettized surface, this function does nothing, returning 0. If all of the colors were set as passed to SDL_SetPalette, it will return 1. If not all the color entries were set exactly as given, it will return 0, and you should look at the surface palette to determine the actual color palette.
Parameters	surface [in] : SDL surface flags [in] : SDL_LOGPAL or SDL_PHYSPAL

	colors [in] : SDL color firstcolor [in] : start point ncolors [in] : number of colors

Example :

```
-- Create a display surface with a grayscale palette
colors    = SDL_Color_new()

-- Fill colors with color information
colors.r=255
colors.g=0
colors.b=0

-- Create display
screen=SDL_SetVideoMode(640, 480, 8, SDL_HWPALETTE)
if    screen == nil then
            print("Couldn’t set video mode: "..SDL_GetError())
end
-- Set palette
SDL_SetPalette(screen, SDL_LOGPAL|SDL_PHYSPAL, colors, 0, 256)
```

13.2.35. SDL_CreateRGBSurface

Name	SDL_CreateRGBSurface
Synopsis	require('SDL') SDL_CreateRGBSurface(flags, width, height, depth, Rmask, Gmask, Bmask, Amask);
Description	Allocate an empty surface (must be called after SDL_SetVideoMode) If depth is 8 bits an empty palette is allocated for the surface, otherwise a 'packed-pixel' SDL_PixelFormat is created using the [RGBA]mask's provided (see SDL_PixelFormat). The flags specifies the type of surface that should be created, it is an OR'd combination of the possible values in the below table :
Return Value	Returns the created surface, or NULL upon error.
Parameters	Flags ,width ,heigh ,depth,Rmask,Gmask,Bmask,Amask

Example:

```
bg = SDL_CreateRGBSurface(bit_or(SDL_SWSURFACE,SDL_ANYFORMAT),
                                    screen.w, screen.h, 8, 0, 0, 0, 0)
if bg == nil then
        print("Error createRGBsurface:"..SDL_GetError())
end
```

13.2.36. SDL_CreateRGBSurfaceFrom

Name	SDL_CreateRGBSurfaceFrom
Synopsis	require('SDL') SDL_CreateRGBSurfaceFrom(pixels, width, height, depth, pitch, Rmask, Gmask, Bmask, Amask);
Description	Creates an SDL_Surface from the provided pixel data. The data stored in pixels is assumed to be of the depth specified in the parameter list. The pixel data is not copied into the SDL_Surface structure so it should no be freed until the surface has been freed with a called to SDL_FreeSurface. pitch is the length of each scanline in bytes.
Return Value	Returns the created surface, or NULL upon error.
Parameters	Pixels ,width,height,depth,pitch,Rmask,Gmask,Bmask,Amask.

Example:

```
local ret_CRGDF
ret_CRGDF    = SDL_CreateRGBSurfaceFrom(screen.pixel,640,480,8,8,0,0,0,0)
if ret_CRGDF == nil then
        print("Error in creating RGB surface from pixel data: "..SDL_GetError())
end
```

13.2.37. SDL_FreeSurface

Name	SDL_FreeSurface
Synopsis	require('SDL') SDL_FreeSurface(surface);
Description	Frees the resources used by a previously created SDL_Surface. If the surface was created using SDL_CreateRGBSurfaceFrom then the pixel data is not freed.
Return Value	nothing
Parameters	Surface [in] : SDL surface

Example:

```
local screen = SDL_SetVideoMode(960, 540, 32,
bit_or(SDL_SWSURFACE, SDL_ANYFORMAT))
--
SDL_FreeSurface(screen)
```

13.2.38. SDL_LockSurface

Name	SDL_LockSurface
Synopsis	require('SDL') SDL_LockSurface(surface);
Description	SDL_LockSurface sets up a surface for directly accessing the pixels. Between calls to SDL_LockSurface and SDL_UnlockSurface, you can write to and read from surface->pixels, using the pixel format stored in surface->format. Once you are done accessing the surface, you should use SDL_UnlockSurface to release it. Not all surfaces require locking. If SDL_MUSTLOCK(surface) evaluates to 0, then you can read and write to the surface at any time, and the pixel format of the surface will not change. No operating system or library calls should be made between lock/unlock pairs, as critical system locks may be held during this time. Note that since SDL 1.1.8 surface locks are recursive , you can lock a surface multiple times, but each lock must have a match unlock.
Return Value	SDL_LockSurface returns 0, or -1 if the surface couldn't be locked.
Parameters	Surface [in] : SDL surface

Example :

```
SDL_LockSurface( surface )
```

.
-- Surface is locked
-- Direct pixel access on surface here
.
SDL_LockSurface(surface)

13.2.39. SDL_UnlockSurface

Name	SDL_UnlockSurface
Synopsis	require('SDL') SDL_UnlockSurface(surface);
Description	Surfaces that were previously locked using SDL_LockSurface must be unlocked with SDL_UnlockSurface. Surfaces should be unlocked as soon as possible.It should be noted that since 1.1.8, surface locks are recursive. See SDL_LockSurface.
Parameters	Surface [in] : SDL surface

Example:

```
SDL_UnlockSurface( surface )
-- Surface is now unlocked
```

13.2.40. SDL_GetGammaRamp

Name	SDL_GetGammaRamp
Synopsis	require('SDL') SDL_GetGammaRamp(redtable, greentable, bluetable);
Description	Gets the gamma translation lookup tables currently used by the display. Each table is an array of 256 Uint16 values. Not all display hardware is able to change gamma.
Return Value	Returns -1 on error.
Parameters	redtable [out] : red greentable [out] : green bluetable [out] : blue

Example:

```
--Check in SDL_SetGammaRamp()
```

13.2.41. SDL_SetGammaRamp

Name	SDL_SetGammaRamp
Synopsis	require('SDL') SDL_SetGammaRamp(redtable, greentable, bluetable);
Description	Sets the gamma lookup tables for the display for each color component. Each table is an array of 256 Uint16 values, representing a mapping between the input and output for that channel. The input is the index into the array, and the output is the 16-bit gamma value at that index, scaled to the output color precision. You may pass NIL to any of the channels to leave them unchanged.This function adjusts the gamma based on lookup tables, you can also have the gamma calculated based on a "gamma function" parameter with SDL_SetGamma.Not all display hardware is able to change gamma.
Return Value	Returns -1 on error (or if gamma adjustment is not supported).
Parameters	redtable [in] : red greentable [in] : green bluetable [in] : blue

Example:

```
ramp = new_Uint16(255)
for i=1, 255 do
        local temp = bit_lshift((i * fade_level/fade_max),8)
```

```
		if not(temp < 0) then
			Uint16_setitem(ramp,i,temp)
		end
end
if SDL_SetGammaRamp(ramp,ramp,ramp) ~= 0 then
		print("Error in GammaRamp:"..SDL_GetError())
end

SDL_GetGammaRamp(ramp1,ramp1,ramp1)
for i = 1, 255 do
		print(i,Uint16_getitem(ramp1,i))
end
```

13.2.42. SDL_SetGamma

Name	SDL_SetGamma
Synopsis	require('SDL') SDL_SetGamma(redgamma, greengamma, bluegamma);
Description	Sets the "gamma function" for the display of each color component. Gamma controls the brightness/contrast of colors displayed on the screen. A gamma value of 1.0 is identity (i.e., no adjustment is made). This function adjusts the gamma based on the "gamma function" parameter, you can directly specify lookup tables for gamma adjustment with SDL_SetGammaRamp.Not all display hardware is able to change gamma.
Return Value	Returns NIL if video has not been initialised with SDL_Init or a pointer to namebuf otherwiseReturns -1 on error (or if gamma adjustment is not supported).
Parameters	redgamma [in] : red greengamma [in] : green bluegamma [in] : blue

Example:

```
-- Set the desired gamma, if any
gamma = 1.0
```

```
if SDL_SetGamma(gamma, gamma, gamma) < 0 then
        print("Unable to set gamma: "..SDL_GetError())
        quit(1)
end
```

13.2.43. SDL_ConvertSurface

Name	SDL_ConvertSurface
Synopsis	require('SDL') SDL_ConvertSurface(src, fmt, flags);
Description	Creates a new surface of the specified format, and then copies and maps the given surface to it. If this function fails, it returns NIL. The flags parameter is passed to SDL_CreateRGBSurface and has those semantics.This function is used internally by SDL_DisplayFormat.
Return Value	Returns either a pointer to the new surface, or NIL on error.
Parameters	src [in] : SDL surface fmt [in] : pixel format flags [in] : SDL flags

Example:

```
Require("SDL")
local screen = SDL_SetVideoMode(640,480,video_bpp,flags)
if screen == nil then
        print("Couldn't set " .. w .." " ..h .. " " .. "video mode:
                                        .. video_bpp .. SDL_GetError() )
end
```

```
surface=SDL_ConvertSurface(screen,screen.format,bit_or(SDL_SWSURFACE,
                                    SDL_ANYFORMAT))
print("Surface width = "..surface.w)
print("Surface height = "..surface.h)
print("Surface flags = "..surface.flags)
print("Surface pitch = "..surface.pitch)
print("Surface refcount = "..surface.refcount)
```

13.2.44. SDL_CreateYUVOverlay

Name	SDL_CreateYUVOverlay
Synopsis	require('SDL') SDL_CreateYUVOverlay(width, height, display);
Description	SDL_CreateYUVOverlay creates a YUV overlay of the specified width, height and format (see SDL_Overlay for a list of available formats), for the provided display. A SDL_Overlay structure is returned. The term 'overlay' is a misnomer since, unless the overlay is created in hardware, the contents for the display surface underneath the area where the overlay is shown will be overwritten when the overlay is displayed.
Return Value	Returns SDL_Overlay struct– YUV video overlay
Parameters	width[in] : display width height[in] : display height format[in] : SDL overlay format display[in] : SDL surface

Example:

```
Require("SDL")
screen = SDL_SetVideoMode(w, h, video_bpp, video_flags)
pic = SDL_LoadBMP(bmpfile)
--
```

```
overlay = SDL_CreateYUVOverlay(pic.w, pic.h, SDL_IYUV_OVERLAY, screen)
if overlay == nil then
        print("Couldn't create overlay: "..SDL_GetError())
end
```

13.2.45. SDL_LockYUVOverlay

Name	SDL_LockYUVOverlay
Synopsis	require('SDL') SDL_LockYUVOverlay(overlay);
Description	Much the same as SDL_LockSurface, SDL_LockYUVOverlay locks the overlay for direct access to pixel data.
Return Value	Returns 0 on success, or -1 on an error.
Parameters	overlay[in] : SDL overlay

Example:

```
Require("SDL")
--
overlay = SDL_CreateYUVOverlay(pic.w, pic.h, SDL_IYUV_OVERLAY, screen)
--
if SDL_LockYUVOverlay(overlay) ~= 0 then
        print("Couldn't lock overlay: "..SDL_GetError())
end
```

13.2.46. SDL_UnlockYUVOverlay

Name	SDL_UnlockYUVOverlay
Synopsis	require('SDL') SDL_UnlockYUVOverlay(overlay);
Description	The opposite to SDL_LockYUVOverlay. Unlocks a previously locked overlay. An overlay must be unlocked before it can be displayed.
Return Value	nothing
Parameters	overlay[in] : SDL overlay

Example:

```
Require("SDL")
--
overlay = SDL_CreateYUVOverlay(pic.w, pic.h, SDL_IYUV_OVERLAY, screen)
--
SDL_UnlockYUVOverlay(overlay)
```

13.2.47. SDL_DisplayYUVOverlay

Name	SDL_DisplayYUVOverlay
Synopsis	require('SDL') SDL_DisplayYUVOverlay(overlay, dstrect);
Description	Blit the overlay to the surface specified when it was created. The SDL_Rect structure, dstrect,specifies the position and size of the destination. If the dstrect is a larger or smaller than the overlay then the overlay will be scaled, this is optimized for 2x scaling.
Return Value	Returns 0 on success
Parameters	overlay[in] : SDL overlay rect[in] : SDL rect on the display

Example:

```
Require("SDL")
--
overlay = SDL_CreateYUVOverlay(pic.w, pic.h, SDL_IYUV_OVERLAY, screen)
--
if SDL_DisplayYUVOverlay(overlay, dstrect) ~=0 then
        print("Error in display YUV overlay: "..SDL_GetError())
end
```

13.2.48. SDL_FreeYUVOverlay

Name	SDL_FreeYUVOverlay
Synopsis	require('SDL') SDL_FreeYUVOverlay(overlay);
Description	Frees and overlay created by SDL_CreateYUVOverlay.
Return Value	nothing
Parameters	overlay[in] : SDL overlay

Example:

```
Require("SDL")
--
overlay = SDL_CreateYUVOverlay(pic.w, pic.h, SDL_IYUV_OVERLAY, screen)
--
SDL_FreeYUVOverlay(overlay)
```

13.2.49. SDL_SetAlpha

Name	SDL_SetAlpha
Synopsis	require('SDL') SDL_SetAlpha(surface, flag, alpha);
Description	SDL_SetAlpha is used for setting the per-surface alpha value and/or enabling and disabling alpha blending. Thesurface parameter specifies which surface whose alpha attributes you wish to adjust. Flags is used to specify whether alpha blending should be used (SDL_SRCALPHA) and whether the surface should use RLE acceleration for blitting (SDL_RLEACCEL). flags can be an OR'd combination of these two options, one of these options or 0. If SDL_SRCALPHA is not passed as a flag then all alpha information is ignored when blitting the surface. The alpha parameter is the per-surface alpha value; a surface need not have an alpha channel to use per-surface alpha and blitting can still be accelerated with SDL_RLEACCEL.
Return Value	This function returns 0, or -1 if there was an error.
Parameters	surface [in] : SDL surface flag[in]: alpha blending flag alpha[in]: per-surface alpha value

13.2.50. SDL_SetColors

Name	SDL_SetColors
Synopsis	require('SDL') SDL_SetColors(surface, colors, firstcolor, ncolors);
Description	Sets a portion of the colormap for the given 8-bit surface. When surface is the surface associated with the current display, the display colormap will be updated with the requested colors. If SDL_HWPALETTE was set in SDL_SetVideoMode flags, SDL_SetColors will always return 1, and the palette is guaranteed to be set the way you desire, even if the window colormap has to be warped or run under emulation. The color components of a SDL_Color structure are 8-bits in size, giving you a total of 2563 =16777216 colors.
Return Value	If surface is not a palettized surface, this function does nothing, returning 0. If all of the colors were set as passed to SDL_SetColors, it will return 1. If not all the color entries were set exactly as given, it will return 0, and you should look at the surface palette to determine the actual color palette.
Parameters	surface [in] : SDL surface colors [in] : SDL color

	firstcolor [in] : start point ncolors [in] : number of colors

Example:

```
colors = SDL_Color_new()
colors.r=255
colors.g=0
colors.b=0

-- Create display
screen=SDL_SetVideoMode(640, 480, 8, SDL_HWPALETTE)
if screen == nil then
        print("Couldn’t set video mode: "..SDL_GetError())
end
-- Set palette
SDL_SetColors(screen, colors, 0, 256)
```

13.2.51. SDL_DisplayFormatAlpha

Name	SDL_DisplayFormatAlpha
Synopsis	require('SDL') SDL_DisplayFormatAlpha (surface);
Description	This function takes a surface and copies it to a new surface of the pixel format and colors of the video framebuffer plus an alpha channel, suitable for fast blitting onto the display surface.
Return value	Returns a copy of surface converted to the display format. This surface must be freed using SDL_FreeSurface. If the conversion fails or runs out of memory, it returns nil.
Parameters	Surface[IN]: SDL surface

13.2.52. SDL_ Overlay

Name	SDL_ Overlay
Synopsis	require(‘SDL’) SDL_ Overlay (None);
Description	A SDL_Overlay is similar to a SDL_Surface except it stores a YUV ove rlay.
Return value	none
Parameters	none

13.2.53. SDL_WaitEvent

Name	SDL_WaitEvent
Synopsis	require('SDL') SDL_WaitEvent(event);
Description	Waits indefinitely for the next available event, returning 1, or 0 if there was an error while waiting for events. If event is not NIL, the next event is removed from the queue and stored in that area.
Return Value	Returning 0 if there was an error while waiting for events, 1 otherwise
Parameters	event[out] : Structure of SDL_Event

Example:

```
event    = SDL_Event_new() -- Event structure
.
-- Check for events
SDL_WaitEvent(event)
if event == SDL_KEYDOWN then--* Handle a KEYDOWN event
        print("Oh! Key press")
end
if event == SDL_MOUSEMOTION then
.
        .
end
SDL_Event_delete(event)
```

13.2.54. SDL_GetModState

Name	SDL_GetModState
Synopsis	require('SDL') SDLMod SDL_GetModState();
Description	Returns the current of the modifier keys (CTRL, ALT, etc.).
Return Value	The return value can be an OR'd combination of the SDLMod enum. SDLMod typedef enum { KMOD_NONE = 0x0000, KMOD_LSHIFT= 0x0001, KMOD_RSHIFT= 0x0002, KMOD_LCTRL = 0x0040, KMOD_RCTRL = 0x0080, KMOD_LALT = 0x0100, KMOD_RALT = 0x0200, KMOD_LMETA = 0x0400, KMOD_RMETA = 0x0800, KMOD_NUM = 0x1000, KMOD_CAPS = 0x2000, KMOD_MODE = 0x4000, } SDLMod; SDL also defines the following symbols for convenience: #define KMOD_CTRL (KMOD_LCTRL|KMOD_RCTRL) #define KMOD_SHIFT (KMOD_LSHIFT|KMOD_RSHIFT)

	#define KMOD_ALT (KMOD_LALT|KMOD_RALT) #define KMOD_META (KMOD_LMETA|KMOD_RMETA)
Parameters	

Example:

```
Require(“SDL”)
print(" modifiers:");

mods = SDL_GetModState();
print("Initial GetModState is:" , mods);

if (mods == 0) then
        print(" (none)");
        return;
end

if (mods == KMOD_LSHIFT) then
        print(" LSHIFT");
end

if (mods == KMOD_RSHIFT) then
        print(" RSHIFT");
end

if(mods == KMOD_LCTRL) then
        print(" LCTRL")
end
if (mods == KMOD_RCTRL) then
        print(" RCTRL")
end
if (mods == KMOD_LALT )then
        print(" LALT")
end
if (mods == KMOD_RALT) then
```

```
        print(" RALT")
end
if (mods == KMOD_LMETA) then
        print(" LMETA")
end
if (mods == KMOD_RMETA) then
        print(" RMETA")
end
if (mods == KMOD_NUM) then
        print(" NUM")
end
if (mods == KMOD_CAPS) then
        print(" CAPS")
end
if (mods == KMOD_MODE) then
        print(" MODE")
end
```

13.2.55. SDL_SetModState

Name	SDL_SetModState
Synopsis	require('SDL') SDL_SetModState(SDLMod modstate);
Description	The inverse of SDL_GetModState, SDL_SetModState allows you to impose modifier key states on your application. Simply pass your desired modifier states into modstate. This value my be a logical OR'd combination of the following: typedef enum { KMOD_NONE = 0x0000, KMOD_LSHIFT= 0x0001, KMOD_RSHIFT= 0x0002, KMOD_LCTRL = 0x0040, KMOD_RCTRL = 0x0080, KMOD_LALT = 0x0100, KMOD_RALT = 0x0200, KMOD_LMETA = 0x0400, KMOD_RMETA = 0x0800, KMOD_NUM = 0x1000, KMOD_CAPS = 0x2000, KMOD_MODE = 0x4000, } SDLMod;
Parameters	Modstate[in]: enum value of SDLMod

Example:

```
Require(“SDL”)
--
mods == KMOD_LSHIFT
SDL_SetModState(mods)
--
```

13.2.56. SDL_EnableUNICODE

Name	SDL_EnableUNICODE
Synopsis	require('SDL') SDL_EnableUNICODE(enable);
Description	Enables/Disables UNICODE keyboard translation. If you wish to translate a keysym to it's printable representation, you need to enable UNICODE translation using this function (enable=0) and then look in the unicode member of the SDL_keysym structure. This value will be zero for keysyms that do not have a printable representation. UNICODE translation is disabled by default as the conversion can cause a slight overhead.
Return Value	Returns the previous translation mode.
Parameters	Flag[in] : Enables (1) /Disables(0) Unicode keyboard translation.

Example:

```
Require("SDL")
if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) then
      print( "Couldn't initialize SDL:",SDL_GetError());
end


if ( SDL_SetVideoMode(640, 480, 0, videoflags) == nil ) then
```

```
        print("Couldn't set 640x480 video mode:" ..SDL_GetError());
        --break;
end
SDL_EnableUNICODE(1);
```

13.2.57. SDL_EnableKeyRepeat

Name	SDL_EnableKeyRepeat
Synopsis	require('SDL') SDL_EnableKeyRepeat(delay, interval);
Description	Enables or disables the keyboard repeat rate. delay specifies how long the key must be pressed before it begins repeating, it then repeats at the speed specified by interval. Both delay and interval are expressed in milliseconds.Setting delay to 0 disables key repeating completely. Good default values are SDL_DEFAULT_REPEAT_DELAY and SDL_DEFAULT_REPEAT_INTERVAL.
Return Value	Delay [in] : time in millisecond for key press to enable repeat. Interval [in]: repeat interval time.
Parameters	delay and interval (milliseconds)

Example:

```
Require("SDL")
if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) then
        print( "Couldn't initialize SDL:"..SDL_GetError());
end
if ( SDL_SetVideoMode(640, 480, 0, videoflags) == nil ) then
            print("Couldn't set 640x480 video mode:" ..SDL_GetError());
            --break;
```

```
end
SDL_EnableKeyRepeat(SDL_DEFAULT_REPEAT_DELAY,
SDL_DEFAULT_REPEAT_INTERVAL);
```

13.2.58. SDL_PollEvent

Name	SDL_PollEvent
Synopsis	require('SDL') SDL_PollEvent(event);
Description	Polls for currently pending events, and returns 1 if there are any pending events, or 0 if there are none available. If event is not NIL, the next event is removed from the queue and stored in that area.
ReturnValue	Returns 1 if there are any pending events, or 0 if there are none available.
Parameters	event[out] : Structure of SDL_Event

Example:

```
Require("SDL")
event    = SDL_Event_new() -- Event structure
.
-- Check for events
While SDL_PollEvent(event) do -- Loop until there are no events left on the queue
        if event == SDL_KEYDOWN then--* Handle a KEYDOWN event
                print("Oh! Key press")
        end
        if event == SDL_MOUSEMOTION then
                .
        end
end
```

SDL_Event_delete(event)

13.2.59. SDL_PushEvent

Name	SDL_PushEvent
Synopsis	require('SDL') SDL_PushEvent(event);
Description	The event queue can actually be used as a two way communication channel. Not only can events be read from the queue, but the user can also push their own events onto it. event is a pointer to the event structure you wish to push onto the queue.Note: Pushing device input events onto the queue doesn't modify the state of the device within SDL.
Return Value	Returns 0 on success or -1 if the event couldn't be pushed.
Parameters	event[in] : Structure of SDL_Event

Example:

```
Require("SDL")
event    = SDL_Event_new();
event.type = SDL_USEREVENT;
ret = SDL_PushEvent(event);
if( ret == -1) then
        print("API is Failed");
end
while    (SDL_PollEvent(event)    ~=    nil) do
```

```
        if ( event.type == SDL_USEREVENT ) then
                print("SDL_PushEvent API :    PASSSSS");
                break;
        end
        if( event.type == SDL_QUIT) then
                print("Quiting Application");
                break;
        end
end    --while end
```

13.2.60. SDL_GetKeyName

Name	SDL_GetKeyName
Synopsis	require('SDL') SDL_GetKeyName(SDLKey key);
Description	Returns the SDL-defined name of the SDLKey key.
Return Value	Returns the SDL-defined name of the SDLKey key
Parameters	Key[in] : SDL key value

Example:

```
Require("SDL")
function PrintKey(sym, pressed)
        if ( sym.sym ~= 0 ) then
                if ( pressed ~= 0)then
                        print("Key    pressed is " .. sym.sym .. "
                                " ..SDL_GetKeyName(sym.sym));
                else
                        print("Key    released is " .. sym.sym .. "
                              " ..SDL_GetKeyName(sym.sym));
                end
        else
```

```
                if ( pressed ~= 0)then
                        print("Unknown Key scancode =    " .. sym.scancode .."pressed");
                else
                        print("Unknown Key scancode =    " .. sym.scancode .."released");
                end
        end
end
```

13.2.61. SDL_PumpEvents

Name	SDL_ PumpEvents
Synopsis	require('SDL') SDL_ PumpEvents (none);
Description	It gathers all the pending input information from devices and places it on the event queue.
Return Value	none
Parameters	none

13.2.62. SDL_ PeepEvents

Name	SDL_ PeepEvents
Synopsis	require('SDL') SDL_ PeepEvents (SDL_Event *events, int numevents, SDL_eventaction action, Uint32 mask);
Description	Checks the event queue for messages and optionally returns them. If action is SDL_ADDEVENT, up to numevents events will be added to the back of the event queue. If action is SDL_PEEKEVENT, up to numevents events at the front of the event queue, matching mask, will be returned and will not be removed from the queue.
Return Value	This function returns the number of events actually stored, or -1 if there was an error.
Parameters	events[IN]: SDL event numevents [IN]: number event action[IN]: SDL event action mask[IN]: masking

13.2.63. SDL_GetEventFilter

Name	SDL_ GetEventFilter
Synopsis	require('SDL') SDL_ GetEventFilter (void);
Description	This function retrieves a instance to the event filter that was previously set using SDL_SetEventFilter.
Return Value	Returns a instance to the event filter or nil if no filter has been set.
Parameters	none

13.2.64. SDL_ SetEventFilter

Name	SDL_ SetEventFilter
Synopsis	require('SDL') SDL_ SetEventFilter (SDL_EventFilter filter);
Description	This function sets up a filter to process all events before they are posted to the event queue. This is a very powerful and flexible feature. If the filter returns 1, then the event will be added to the internal queue.
Return Value	none
Parameters	filter[IN]: SDL event filter

13.2.65. SDL_ EventState

Name	SDL_ EventState
Synopsis	require('SDL') SDL_ EventState (Uint8 type, int state);
Description	This function allows you to set the state of processing certain events. SDL_EventState must be called following the SDL_VideoInfo call. Otherwise the back-end data structure will be erased.
Return Value	Return the current processing state of the specified event type.
Parameters	type[IN]: number State[IN]: number

13.2.66. SDL_ GetKeyState

Name	SDL_ GetKeyState
Synopsis	require('SDL') SDL_ GetKeyState (int *numkeys);
Description	Gets a snapshot of the current keyboard state. The current state is returned as an instance to an table. The size of this array is stored in numkeys. The table is indexed by theSDLK_* symbols (see SDLKey). A value of 1 means that the key is pressed and a value of 0 means that it is not. The instance returned is an instance to an internal SDL table . It will be valid for the whole lifetime of the application and should not be freed by the caller.
Return Value	Returned is a instance to an internal SDL array.
Parameters	numkeys[IN]: number instance

13.2.67. SDL_ GetAppState

Name	SDL_ GetAppState
Synopsis	require('SDL') SDL_ GetAppState ();
Description	Get the state of the application
Return Value	The value returned is a bitwise combination
Parameters	none

AUDIO:

13.2.68. SDL_OpenAudio

Name	SDL_OpenAudio
Synopsis	require('SDL') SDL_OpenAudio(desired, obtained);
Description	Please look below
Return Value	int
Parameters	desired[in] : pointer to SDL AudioSpec obtained[in] : pointer to SDL AudioSpec

Description

This function opens the audio device with the desired parameters, and returns 0 if successful,placing the actual hardware parameters in the structure pointed to by obtained. If obtained is NIL, the audio data passed to the callback function will be guaranteed to be in the requested format, and will be automatically converted to the hardware audio format if necessary. This function returns -1 if it failed to open the audio device, or couldn't set up the audio thread.
To open the audio device a desired SDL_AudioSpec must be created.

```
SDL_AudioSpec *desired;
.
.
desired=(SDL_AudioSpec *)malloc(sizeof(SDL_AudioSpec));
```

You must then fill this structure with your desired audio specifications.
desired->freq
The desired audio frequency in samples-per-second.

desired->format
The desired audio format (see SDL_AudioSpec)

desired->samples
The desired size of the audio buffer in samples. This number should be a power of two, and may be adjusted by the audio driver to a value more suitable for the hardware. Good values seem to range between 512 and 8192 inclusive, depending on the application and CPU speed.

desired->callback
This should be set to a function that will be called when the audio device is ready for more data.It is passed a pointer to the audio buffer, and the length in bytes of the audio buffer. This function usually runs in a separate thread, and so you should protect data structures that it accesses by calling SDL_LockAudio and SDL_UnlockAudio in your code.
The callback prototype is:
callback(*userdata, Uint8 *stream, len);

userdata is the pointer stored in userdata field of the SDL_AudioSpec.

stream is a pointer to the audio buffer you want to fill with information and len is the length of the audio .buffer in bytes.

desired->userdata
This pointer is passed as the first parameter to the callback function.
SDL_OpenAudio reads these fields from the desired SDL_AudioSpec structure pass to the function and attempts to find an audio configuration matching your desired. if the obtained parameter is NIL then SDL with convert from your desired audio settings to the hardware settings as it plays.
If obtained is NIL then the desired SDL_AudioSpec is your working specification, otherwise the obtained SDL_AudioSpec becomes the working specification and the desirec specification can be deleted. The data in the working specification is used when building SDL_AudioCVT's for converting loaded data to the hardware format.

SDL_OpenAudio calculates the size and silence fields for both the desired and obtained specifications. The size field stores the total size of the audio buffer in bytes, while the silence stores the value used to represent silence in the audio buffer The audio device starts out playing silence when it's opened, and should be enabled for playing by calling SDL_PauseAudio(0) when you are ready for your audio callback function to be called. Since the audio driver may modify the

requested size of the audio buffer, you should allocate any local mixing buffers after you open the audio device.

Example

```
Require(“SDL”)

-- Prototype of our callback function
  my_audio_callback( userdata, stream,    len)

-- Open the audio device

-- Allocate a desired SDL_AudioSpec
desired= SDL_AudioSpec_new()

-- Allocate space for the obtained SDL_AudioSpec
obtained= SDL_AudioSpec_new()

-- 22050Hz - FM Radio quality
Desired.freq=22050

-- 16-bit signed audio
Desired.format=AUDIO_S16LSB

-- Large audio buffer reduces risk of dropouts but increases response time
Desired.samples=8192

-- Our callback function
Desired.callback=my_audio_callback
Desired.userdata=nil

-- Open the audio device
if      SDL_OpenAudio(desired, obtained) < 0 then
            print( "Couldn’t open audio: "..SDL_GetError())
exit(-1)
```

```
end

-- desired spec is no longer needed
SDL_AudioSpec_delete(desired)

hardware_spec=obtained
.
-- Prepare callback for playing
.
-- Start playing
SDL_PauseAudio(0)
```

13.2.69. SDL_PauseAudio

Name	SDL_PauseAudio
Synopsis	require('SDL') SDL_PauseAudio(pause_on);
Description	This function pauses and unpauses the audio callback processing. It should be called with pause_on=0 after opening the audio device to start playing sound. This is so you can safely initialize data for your callback function after opening the audio device. Silence will be written to the audio device during the pause.
Return Value	nothing
Parameters	pause_on[in] : value of pause_on

Example:

```
Require("SDL")

-- Pause playing
SDL_PauseAudio(0)
```

13.2.70. SDL_GetAudioStatus

Name	SDL_GetAudioStatus
Synopsis	require('SDL') SDL_GetAudioStatus();
Description	typedef enum{ SDL_AUDIO_STOPPED, SDL_AUDIO_PAUSED, SDL_AUDIO_PLAYING } SDL_audiostatus; Returns either SDL_AUDIO_STOPPED, SDL_AUDIO_PAUSED or SDL_AUDIO_PLAYING depending on the current audio state.
Return value	Returns either SDL_AUDIO_STOPPED, SDL_AUDIO_PLAYING or SDL_AUDIO_PAUSED depending on the current audio state.
Parameters	nothing

Example:

```
Require("SDL")
local temp = SDL_GetAudioStatus()
if temp == SDL_AUDIO_STOPPED then
        print("Audio is stopped.")
end
if temp == SDL_AUDIO_PLAYING then
```

```
        print("Audio is playing.")
end
if temp == SDL_AUDIO_PAUSED then
        print("Audio is Paused.")
end
```

13.2.71. SDL_LoadWAV

Name	SDL_LoadWAV
Synopsis	require('SDL') SDL_LoadWAV(file, spec, audio_buf, audio_len);
Description	SDL_LoadWAV This function loads a WAVE file into memory. If this function succeeds, it returns the given SDL_AudioSpec, filled with the audio data format of the wave data, and sets audio_buf to a malloc'd buffer containing the audio data, and sets audio_len to the length of that audio buffer, in bytes. You need to free the audio buffer with SDL_FreeWAV when you are done with it. Currently raw, MS-ADPCM and IMA-ADPCM WAVE files are supported.
Return value	This function returns NULL and sets the SDL error message if the wave file cannot be opened, uses an unknown data format, or is corrupt.
Parameters	file[in] : WAV file name spec[out] : SDL AudioSpec (Audio data format) audio_len[out] : length of the audio buffer

Example:

```
Require("SDL")
spec = SDL_AudioSpec_new()
soundlen = new_Uint32p()
file_name = "sample.wav";

-- Load the wave file into memory
local sound = SDL_LoadWAV(file_name, spec, soundlen)
```

13.2.72. SDL_FreeWAV

Name	SDL_FreeWAV
Synopsis	require('SDL') SDL_FreeWAV(audio_buf);
Description	After a WAVE file has been opened with SDL_LoadWAV its data can eventually be freed with SDL_FreeWAV. audio_buf is a pointer to the buffer created by SDL_LoadWAV.
Return value	nothing
Parameters	Audio buffer - audio_buf

Example:

```
Require("SDL")

-- load wave file
SDL_FreeWAV(sound)
```

13.2.73. SDL_BuildAudioCVT

Name	SDL_BuildAudioCVT
Synopsis	require('SDL') SDL_BuildAudioCVT(cvt,src_format,src_channels,src_rate, dst_format, dst_channels, dst_rate);
Description	Before an SDL_AudioCVT structure can be used to convert audio data it must be initialized with source and destination information. src_format and dst_format are the source and destination format of the conversion. src_channels and dst_channels are the number of channels in the source and destination formats. Finally, src_rate and dst_rate are the frequency or samples-per-second of the source and destination formats.
Return Value	Returns -1 if the filter could not be built or 1 if it could.
Parameters	cvt[out] : SDL AudioCVT src_format[in] : source format of the conversion src_channels[in]: no. of channel in source format src_rate[in] : frequance of source format dst_format[in] : source format of the conversion dst_channels[in]: no. of channel in source format dst_rate[in] : frequance of source format

Example:

```
Require(“SDL”)

-- load wave file
-- Initialize SDL_AudioCVT structure --
cvt = SDL_AudioCVT_new()
local aud = SDL_BuildAudioCVT(cvt,AUDIO_S16LSB,1,4096,AUDIO_S8,1,256)
print("SDL_BuildAudioCVT Return value:"..aud)
```

13.2.74. SDL_ConvertAudio

Name	SDL_ConvertAudio
Synopsis	require('SDL') SDL_ConvertAudio(cvt);
Description	SDL_ConvertAudio takes one parameter, cvt, which was previously initilized. Initilizing a SDL_AudioCVT is a two step process. First of all, the structure must be passed to SDL_BuildAudioCVT along with source and destination format parameters. Secondly, the cvt->buf and cvt->len fields must be setup. cvt->buf should point to the audio data and cvt->len should be set to the length of the audio data in bytes. Remember, the length of the buffer pointed to by buf show be len*len_mult bytes in length. Once the SDL_AudioCVTstructure is initilized then we can pass it to SDL_ConvertAudio, which will convert the audio data pointer to by cvt->buf. If SDL_ConvertAudio returned 0 then the conversion was completed successfully, otherwise -1 is returned.If the conversion completed successfully then the converted audio data can be read from cvt->buf.The amount of valid, converted, audio data in the buffer is equal to cvt->len*cvt->len_ratio.
Return value	Returned 0 then the conversion was completed successfully, otherwise -1 is returned
Parameters	cvt[in] : pointer to SDL AudioCVT

Example:

```
Require(“SDL”)

-- Converting some WAV data to hardware format
my_audio_callback( userdata, stream,    len)
wav_cvt    = SDL_AudioCVT_new()
wav_len -= new_Uint32p()

-- Allocated audio specs
desired= SDL_AudioSpec_new()
obtained=SDL_AudioSpec_new()
spec=SDL_AudioSpec_new()

-- Set desired format
Desired.freq=22050
Desired.format=AUDIO_S16LSB
Desired.samples=8192
Desired.callback=my_audio_callback
Desired.userdata=nil

-- Open the audio device
if    SDL_OpenAudio(desired, obtained) < 0    then
print( "Couldn’t open audio: "..SDL_GetError())
exit(-1)
end
SDL_AudioSpec_delete(desired)

-- Load the test.wav
spc, sound = SDL_LoadWAV(“test.wav”,spec, wav_len)

-- Build AudioCVT
ret = SDL_BuildAudioCVT(wav_cvt,
spec.format, spec.channels,spec.freq,
obtained.format, obtained.channels, obtained.freq)
```

```
-- Check that the convert was built
If ret==-1then
        print( "Couldn’t build converter!")
        SDL_CloseAudio()
        SDL_AudioSpec_delete(obtained)
        SDL_FreeWAV(sound)
End

-- Setup for conversion
local temp = Uint32p_value(wav_len)
wav_cvt.buf= new_Uint8(wav_len)
wav_cvt.buf = sound
wav_cvt.len = temp

-- We can delete to original WAV data now
SDL_FreeWAV(sound)

-- And now we’re ready to convert
SDL_ConvertAudio(wav_cvt)
```

13.2.75. SDL_MixAudio

Name	SDL_MixAudio
Synopsis	require('SDL') SDL_MixAudio(dst, src, len, volume);
Description	This function takes two audio buffers of len bytes each of the playing audio format and mixes them,performing addition, volume adjustment, and overflow clipping. The volume ranges from 0 to SDL_MIX_MAXVOLUME and should be set to the maximum value for full audio volume. Note this does not change hardware volume. This is provided for convenience -- you can mix your own audio data.
Return value	nothing
Parameters	dst[in]: destination audio buffer src[in] : source audio buffer len[in] : audio buffer lenth volume[in] : volume range

Example:

```
Require("SDL")

-- Open audio device
-- Load wave file
-- Initialize AudioCVT structure
SDL_MixAudio(cvt.buf, sound, 8, SDL_MIX_MAXVOLUME)
```

13.2.76. SDL_LockAudio

Name	SDL_LockAudio
Synopsis	require('SDL') SDL_LockAudio();
Description	The lock manipulated by these functions protects the callback function. During a LockAudio period,you can be guaranteed that the callback function is not running. Do not call these from the callback function or you will cause deadlock.
Return value	nothing
Parameters	nothing

Example:

```
Require("SDL")
-- Open audio device
-- Load wave file
-- Initialize AudioCVT structure
SDL_LockAudio()
```

13.2.77. SDL_UnlockAudio

Name	SDL_UnlockAudio
Synopsis	require('SDL') SDL_UnlockAudio();
Description	Unlocks a previous SDL_LockAudio call.
Return value	nothing
Parameters	nothing

Example:

```
Require("SDL")
-- Open audio device
-- Load wave file
-- Initialize AudioCVT structure
-- Lock the audio callback
SDL_UnlockAudio()
```

13.2.78. SDL_CloseAudio

Name	SDL_CloseAudio
Synopsis	require('SDL') SDL_CloseAudio();
Description	This function shuts down audio processing and closes the audio device.
Return value	nothing
Parameters	nothing

Example:

```
Require("SDL")
-- Open audio device
-- Load wave file
-- Initialize AudioCVT structure
SDL_CloseAudio()
```

13.2.79. SDL_CreateThread

Name	SDL_CreateThread
Synopsis	require(‘SDL’) require(‘SDL_thread’) SDL_CreateThread(file, fn, data);
Description	SDL_CreateThread creates a new thread of execution that shares all of its parent’s global memory, signal handlers, file descriptors, etc, and runs the function fn passing the data. The thread quits when this function returns.
Parameters	file [in] : file fn [in] : function data [in] : user data

13.2.80. SDL_ThreadID

Name	SDL_ThreadID
Synopsis	require('SDL') require('SDL_thread') SDL_ThreadID();
Description	Get the 32-bit thread identifier for the current thread.

13.2.81. SDL_GetThreadID

Name	SDL_GetThreadID
Synopsis	require('SDL') require('SDL_thread') SDL_GetThreadID(thread);
Description	Returns the ID of a SDL_Thread created by SDL_CreateThread.
Return Value	thread [out]: SDL_Thread

13.2.82. SDL_WaitThread

Name	SDL_WaitThread
Synopsis	require('SDL') require('SDL_thread') SDL_WaitThread(thread, status);
Description	Wait for a thread to finish (timeouts are not supported).
Return Value	The return code for the thread function is placed in the area pointed to by status, if status is not NIL.
Parameters	thread [in] : SDL_Thread status [out] : return code of thread fn

13.2.83. SDL_KillThread

Name	SDL_KillThread
Synopsis	require('SDL') require('SDL_thread') SDL_KillThread(thread);
Description	SDL_KillThread gracelessly terminates the thread associated with thread. If possible, you should use some other form of IPC to signal the thread to quit.
Parameters	thread [in] : SDLThread

Example:

```
function mythread(data)
	local	str = voidp_to_charp(data)
	for i=1,10 do
		print("...Thread",str," running.....")
		SDL_Delay(1*1000)
	end
	return 0
end

local function SDL_Thread_Test()
	local str = charp_to_voidp("#1")
	local thread1 = SDL_CreateThread("mythread",str)
	if thread1 == nil then
		print("Couldn't create thread 1:", SDL_GetError())
		return 2
	end
```

```
        str = charp_to_voidp("#2")
        local thread2 = SDL_CreateThread("mythread",str)
        if thread2 == nil then
                print("Couldn't create thread 2:", SDL_GetError())
                return 2
        end

        SDL_WaitThread(thread1, nil)
        SDL_WaitThread(thread2, nil)

        print("Thread ID of First Thread: ",thread1)
        print("Thread ID of Second Thread: ",thread2)

        SDL_KillThread(thread1)
        SDL_KillThread(thread2)
end
```

13.2.84. SDL_CreateMutex

Name	SDL_CreateMutex
Synopsis	require(‘SDL’) require(‘SDL_thread’) SDL_CreateMutex();
Description	Create a new, unlocked mutex.
Parameters	

13.2.85. SDL_DestroyMutex

Name	SDL_DestroyMutex
Synopsis	require('SDL') require('SDL_thread') SDL_DestroyMutex(mutex);
Description	Destroy a previously created mutex.
Parameters	mutex [in] : SDL Mutex

13.2.86. SDL_mutexP

Name	SDL_mutexP
Synopsis	require('SDL') require('SDL_thread') SDL_mutexP(mutex);
Description	Locks the mutex, which was previously created with SDL_CreateMutex. If the mutex is already locked then SDL_mutexP will not return until it is unlocked. Returns 0 on success, or -1 on an error. SDL also defines a macro #define SDL_LockMutex(m) SDL_mutexP(m).
Parameters	mutex [in] : SDL Mutex

13.2.87. SDL_mutexV

Name	SDL_mutexV
Synopsis	require('SDL') require('SDL_thread') SDL_mutexV(mutex);
Description	Unlocks the mutex, which was previously created with SDL_CreateMutex. Returns 0 on success,or -1 on an error. SDL also defines a macro #define SDL_UnlockMutex(m) SDL_mutexV(m).
Parameters	mutex [in] : SDL Mutex

Example:

```
function mymutex(data)
        local   str = voidp_to_charp(data)
        for i = 1,30 do
                if SDL_mutexP(mut)==-1 then
                        print("SDL_mutexP(mut) error", SDL_GetError())
                        return 2
                end

                sum = sum + 1
                print("Thread=",str,"Thread ID=",SDL_ThreadID(),"sum=",sum)

                if SDL_mutexV(mut)==-1 then
                        print("SDL_mutexV(mut) error", SDL_GetError())
                        return 2
```

```
                end
                SDL_Delay(1000)
        end
        return 0
end

function SDL_Mutex_Test()
        local str = charp_to_voidp("#1")
        local thread1 = SDL_CreateThread("mymutex",str)
        if thread1 == nil then
                print("Couldn't create thread 1:", SDL_GetError())
                return 2
        end

        str = charp_to_voidp("#2")
        local thread2 =SDL_CreateThread("mymutex",str)
        if thread2 == nil then
                print("Couldn't create thread 2:", SDL_GetError())
                return 2
        end

        SDL_WaitThread(thread1, nil)
        SDL_WaitThread(thread2, nil)


        SDL_DestroyMutex(mut)
        SDL_KillThread(thread1)
        SDL_KillThread(thread2)
end
```

13.2.88. SDL_CreateSemaphore

Name	SDL_CreateSemaphore
Synopsis	require('SDL') require('SDL_thread') SDL_CreateSemaphore(initial_value);
Description	SDL_CreateSemaphore() creates a new semaphore and initializes it with the value initial_value. Each locking operation on the semaphore by SDL_SemWait, SDL_SemTryWait or SDL_SemWaitTimeout will atomically decrement the semaphore value. The locking operation will be blocked if the semaphore value is not positive (greater than zero). Each unlock operation by SDL_SemPost will atomically increment the semaphore value.
Return Value	Returns a an initialized semaphore or NIL if there was an error.
Parameters	initial_value [in] : initial value

Example:

```
my_sem = SDL_CreateSemaphore(INITIAL_SEM_VALUE)

if my_sem == NULL then
        return CREATE_SEM_FAILED;
end
```

13.2.89. SDL_DestroySemaphore

Name	SDL_DestroySemaphore
Synopsis	require('SDL') require('SDL_thread') SDL_DestroySemaphore(sem);
Description	SDL_DestroySemaphore destroys the semaphore pointed to by sem that was created by SDL_CreateSemaphore. It is not safe to destroy a semaphore if there are threads currently blocked waiting on it.
Parameters	sem [in] : SDL sem

Example:

```
if my_sem ~= nil then
        SDL_DestroySemaphore(my_sem)
        my_sem = nil
end
```

13.2.90. SDL_SemWait

Name	SDL_SemWait
Synopsis	require('SDL') require('SDL_thread') SDL_SemWait(sem);
Description	SDL_SemWait() suspends the calling thread until either the semaphore pointed to by sem has a positive value, the call is interrupted by a signal or error. If the call is successful it will atomically decrement the semaphore value.After SDL_SemWait() is successful, the semaphore can be released and its count atomically incremented by a successful call to SDL_SemPost.
Return Value	Returns 0 if successful or -1 if there was an error (leaving the semaphore unchanged).
Parameters	sem [in] : SDL sem

Example:

```
if SDL_SemWait(my_sem) == -1 then
        return WAIT_FAILED
end
...
SDL_SemPost(my_sem);
```

13.2.91. SDL_SemTryWait

Name	SDL_SemTryWait
Synopsis	require('SDL') require('SDL_thread') SDL_SemTryWait(sem);
Description	SDL_SemTryWait is a non-blocking varient of SDL_SemWait. If the value of the semaphore pointed to by sem is positive it will atomically decrement the semaphore value and return 0, otherwise it will return SDL_MUTEX_TIMEOUT instead of suspending the thread.After SDL_SemTryWait is successful, the semaphore can be released and its count atomically incremented by a successful call to SDL_SemPost.
Return Value	Returns 0 if the semaphore was successfully locked or either SDL_MUTEX_TIMEOUT or -1 if the thread would have suspended or there was an error, respectivly.If the semaphore was not successfully locked, the semaphore will be unchanged.
Parameters	sem [in] : SDL sem

Example:

```
res = SDL_SemTryWait(my_sem)
```

```
if res == SDL_MUTEX_TIMEOUT then
        return TRY_AGAIN;
end
if res == -1 then
        return WAIT_ERROR;
end
...
SDL_SemPost(my_sem)
```

13.2.92. SDL_SemWaitTimeout

Name	SDL_SemWaitTimeout
Synopsis	require('SDL') require('SDL_thread') SDL_SemWaitTimeout(sem, timeout);
Description	SDL_SemWaitTimeout() is a varient of SDL_SemWait with a maximum timeout value. If the value of the semaphore pointed to by sem is positive (greater than zero) it will atomically decrement the semaphore value and return 0, otherwise it will wait up to timeout milliseconds trying to lock the semaphore. This function is to be avoided if possible since on some platforms it is implemented by polling the semaphore every millisecond in a busy loop.After SDL_SemWaitTimeout() is successful, the semaphore can be released and its count atomically incremented by a successful call to SDL_SemPost.
Return Value	Returns 0 if the semaphore was successfully locked or either SDL_MUTEX_TIMEOUT or -1 if the timeout period was exceeded or there was an error, respectivly.If the semaphore was not successfully locked, the semaphore will be unchanged.
Parameters	sem [in] : SDL sem ms [in] : time

Example:

```
res = SDL_SemWaitTimeout(my_sem, WAIT_TIMEOUT_MILLISEC)

if res == SDL_MUTEX_TIMEOUT then
        return TRY_AGAIN
end
if res == -1 then
        return WAIT_ERROR
end
...
SDL_SemPost(my_sem)
```

13.2.93. SDL_SemPost

Name	SDL_SemPost
Synopsis	require('SDL') require('SDL_thread') SDL_SemPost(sem);
Description	SDL_SemPost unlocks the semaphore pointed to by sem and atomically increments the semaphores value. Threads that were blocking on the semaphore may be scheduled after this call succeeds. SDL_SemPost should be called after a semaphore is locked by a successful call to SDL_SemWait,SDL_SemTryWait or SDL_SemWaitTimeout.
Return Value	Returns 0 if successful or -1 if there was an error (leaving the semaphore unchanged).
Parameters	sem [in] : SDL sem

Example:

```
SDL_SemPost(my_sem)
```

13.2.94. SDL_SemValue

Name	SDL_SemValue
Synopsis	require('SDL') require('SDL/SDL_thread') SDL_SemValue(sem);
Description	SDL_SemValue() returns the current semaphore value from the semaphore pointed to by sem.
Return Value	Returns current value of the semaphore.
Parameters	sem [in] : SDL sem

Example:

```
sem_value = SDL_SemValue(my_sem);
```

13.2.95. SDL_CreateCond

Name	SDL_CreateCond
Synopsis	require('SDL') require('SDL_thread') SDL_CreateCond();
Description	Creates a condition variable.
Return Value	SDL_cond instance

Example:

```
cond=SDL_CreateCond();
.
--Do stuff
.
SDL_DestroyCond(cond);
```

13.2.96. SDL_DestroyCond

Name	SDL_DestroyCond
Synopsis	require('SDL') require('SDL_thread') SDL_DestroyCond(cond);
Description	Destroys a condition variable.
Parameters	cond [in] : SDL_cond

13.2.97. SDL_CondSignal

Name	SDL_CondSignal
Synopsis	require('SDL') require('SDL_thread') SDL_CondSignal(cond);
Description	Restart one of the threads that are waiting on the condition variable, cond. Returns 0 on success of -1 on an error.
Parameters	cond [in] : SDL_cond

Example:

```
mutc=SDL_CreateMutex()
cond = SDL_CreateCond()
if SDL_CondSignal(cond) == -1 then
        print("SDL_CondSignal Error:", SDL_GetError())
        return 2
end
```

13.2.98. SDL_CondBroadcast

Name	SDL_CondBroadcast
Synopsis	require('SDL') require('SDL_thread') SDL_CondBroadcast(cond);
Description	Restarts all threads that are waiting on the condition variable, cond. Returns 0 on success, or -1 on an error.
Parameters	cond [in] : SDL_cond

Example:

```
mutc=SDL_CreateMutex()
cond = SDL_CreateCond()

if SDL_CondBroadcast(cond, mutc, 1000) == -1 then
        print("SDL_CondBroadcast Error:", SDL_GetError())
        return 2
end
```

13.2.99. SDL_CondWait

Name	SDL_CondWait
Synopsis	require('SDL') require('SDL_thread') SDL_CondWait(cond, mut);
Description	Wait on the condition variable cond and unlock the provided mutex. The mutex must the locked before entering this function. Returns 0 when it is signalled, or -1 on an error.
Parameters	cond [in] : SDL_cond mut [in] :SDL mutex

Example:

```
mutc=SDL_CreateMutex()
cond = SDL_CreateCond()
if SDL_CondWait(cond, mutc) == -1 then
        print("SDL_CondWait Error:", SDL_GetError())
        return 2
end
```

13.2.100. SDL_CondWaitTimeout

Name	SDL_CondWaitTimeout
Synopsis	require('SDL') require('SDL_thread') SDL_CondWaitTimeout(cond, mutex, ms);
Description	Wait on the condition variable cond for, at most, ms milliseconds. mut is unlocked so it must be locked when the function is called. Returns SDL_MUTEX_TIMEDOUT if the condition is not signaled in the allotted time, 0 if it was signalled or -1 on an error.
Parameters	cond [in] : SDL_cond mutex [in] :SDL mutex ms [in] : miliseconds

Example:

```
mutc=SDL_CreateMutex()
cond = SDL_CreateCond()
if SDL_CondWaitTimeout(cond, mutc, 1000) == -1 then
        print("SDL_CondWaitTimeout Error:", SDL_GetError())
        return 2
end
SDL_DestroyCond(cond)
```

Time

13.2.101. SDL_GetTicks

Name	SDL_GetTicks
Synopsis	require('SDL') SDL_GetTicks();
Description	Get the number of milliseconds since the SDL library initialization. Note that this value wraps if the program runs for more than ~49 days.
Parameters	

Example:

```
Require("SDL")
thenn = SDL_GetTicks()
frames = 0
for i = 0, 256 do
        r = i
        g = 0
        b = 0
        SDL_FillRect(screen, dstrect, SDL_MapRGB(screen.format, r, g, b))
        SDL_Flip(screen)
        frames = frames + 1
end

now = SDL_GetTicks()
seconds = (now - thenn) / 1000
if seconds > 0.0 then
```

```
        print( frames .." fills and flips in " ..seconds .." seconds, " ..frames/seconds .." FPS" )
else
        print(frames .." fills and flips in zero seconds!n")
end
```

13.2.102. SDL_Delay

Name	SDL_Delay
Synopsis	require('SDL') SDL_Delay(ms);
Description	Wait a specified number of milliseconds before returning. SDL_Delay will wait at least the specified time, but possible longer due to OS scheduling. Note: Count on a delay granularity of at least 10 ms. Some platforms have shorter clock ticks but this is the most common.
Parameters	ms [in]: time delay (in millisecond) to return from this function.

Example:

```
Require("SDL")
.
print("Waiting 10 seconds\n");
SDL_Delay(10*1000);
.
.
```

13.2.103. SDL_AddTimer

Name	SDL_AddTimer
Synopsis	require('SDL') SDL_AddTimer(interval, callback, param);
Description	Adds a callback function to be run after the specified number of milliseconds has elapsed. The callback function is passed the current timer interval and the user supplied parameter from the SDL_AddTimer call and returns the next timer interval. If the returned value from the callback is the same as the one passed in, the periodic alarm continues, otherwise a new alarm is scheduled. To cancel a currently running timer call SDL_RemoveTimer with the timer ID returned from SDL_AddTimer.The timer callback function may run in a different thread than your main program, and so shouldn't call any functions from within itself. You may always call SDL_PushEvent, however.
Return Value	Returns an ID value(SDL_TimerID) for the added timer or NIL if there was an error.
Example	
Parameters	Interval[in]: time interval value to call callback at this interval value. Callback[in] : SDL_NewTimerCallback callback function Param[in]: input parameters value of callback.
Callback	/* type definition for the "new" timer callback function */ typedef Uint32 (*SDL_NewTimerCallback)(Uint32 interval, *param);

Example:

```
Require(“SDL”)
function    callback( interval, param)
   print("Timer :" ..interval .. " param = " .. param);
   return interval;
end

function main()
          print("Testing multiple timers...\n");
           t1 = SDL_AddTimer(100, callback, (void*)1);
           if(t1 == 0) then
                  print("Could not create timer 1:".. SDL_GetError());
          end
          t2 = SDL_AddTimer(50, callback, (void*)2);
           if(t2 == 0) then
                    print("Could not create timer 2 :" .. SDL_GetError());
          end
           t3 = SDL_AddTimer(233, callback, (void*)3);
           if(t3 == 0) then
               print("Could not create timer 3:" .. SDL_GetError());
          end
end
```

13.2.104. SDL_RemoveTimer

Name	SDL_RemoveTimer
Synopsis	require('SDL') SDL_RemoveTimer(id);
Description	Removes a timer callback previously added with SDL_AddTimer.
Return Value	Returns a boolean value (SDL_bool) indicating success.
Parameters	Id[in] : SDL_TimerID structure value

Example:

```
Require("SDL")
.
-- Add t1,t2,t3 timers
.

print("Removing timer 1 and waiting 5 more seconds\n");
SDL_RemoveTimer(t1);
SDL_Delay(5*1000);
SDL_RemoveTimer(t2);
SDL_RemoveTimer(t3);
```

13.2.105. SDL_SetTimer

Name	SDL_SetTimer
Synopsis	require('SDL') SDL_SetTimer(interval, callback);
Description	Set a callback to run after the specified number of milliseconds has elapsed. The callback function is passed the current timer interval and returns the next timer interval. If the returned value is the same as the one passed in, the periodic alarm continues, otherwise a new alarm is scheduled. To cancel a currently running timer, call SDL_SetTimer(0, NIL); The timer callback function may run in a different thread than your main constant, and so shouldn't call any functions from within itself. If you use this function, you need to pass SDL_INIT_TIMER to SDL_Init().
Parameters	Interval [in]: Time to run callback at this time interval. Callback[in] : SDL_TimerCallback callback function
Callback	/* Function prototype for the timer callback function */ typedef Uint32 (*SDL_TimerCallback)(Uint32 interval);

Example:

```
Require("SDL")
ticks =0
if ( SDL_Init(SDL_INIT_TIMER) < 0 ) then
        print( "Couldn't initialize SDL:" .. SDL_GetError());
end
local desired =0;
```

```
argv = MakeGetEvents()

if ( argv ) then
        desired = argv
end
if ( desired == 0 ) then
        desired = DEFAULT_RESOLUTION;
end
.
SDL_SetTimer(desired, ticktock);
```

Files

13.2.106. SDL_RWFromFile

Name	DL_RWFromFile
Synopsis	require('SDL') SDL_RWFromFile (file, mode)
Description	Opens a file.
Return Value	Returns the newly created RWops structure.
Parameters	file [in] : file name mode [in] : Mode to open a file

Example:

```
local rwops = SDL_RWFromFile(chResPath.."test.txt", "rb")
if rwops == nil then
        print("Couldn't open test.txt")
        return 2
end

local buf = SDL_malloc(256)
SDL_RWread(rwops,buf,16,256/16)
pstr = voidp_to_charp(buf)
print("Read from file :",pstr)
SDL_RWclose(rwops)
```

13.2.107. SDL_RWFromMem

Name	SDL_RWFrom
Synopsis	require('SDL') SDL_RWFrom(mem, size)
Description	Prepares a memory area for use with SDL_RWops
Return Value	Returns the newly created RWops structure
Parameters	Mem [in] : memory size [in] : size of memory

Example:

```
local bitmap = SDL_malloc(310000)
local rwops = SDL_RWFromMem(bitmap, 310000)
if rwops == nil then
        print("Couldn't open test.txt")
        return 1
end
SDL_RWclose(rwops)
```

13.2.108. SDL_RWFromConstMem

Name	SDL_RWFromConstMem
Synopsis	require('SDL') SDL_RWFromConstMem (mem, size);
Description	Prepares a constant memory area for use with SDL_RWops.
Return Value	Returns the newly created RWops structure
Parameters	Mem [in] : memory size [in] : size of memory

Example:

```
local bitmap_d = {66, 77, 86, 2, 0, 0, 0, 0}

bitmap = new_Uint8(8)
for i = 1, 8 do
	Uint8_setitem(bitmap, 1, bitmap_d[i])
end

rwops = SDL_RWFromConstMem(bitmap, 8)
if rwops == nil then
	print("Couldn't open test.txt")
	return 2
end
SDL_RWclose(rwops)
```

13.2.109. SDL_AllocRW

Name	SDL_AllocRW
Synopsis	require('SDL') SDL_AllocRW (void);
Description	Returns the newly created RWops structure
Parameters	none

13.2.110. SDL_FreeRW

Name	SDL_FreeRW
Synopsis	require('SDL') SDL_FreeRW (context);
Description	Free the memory
Parameters	Context[in] : SDL_RWops

Example:

```
rwops=SDL_AllocRW()
if rwops==nil then
	print("Couldn't allocate SDL_RWops structure")
	return 2
end
SDL_FreeRW(rwops)
print("SDL_FreeRW() Passed!!!")
--
SDL_FreeRW(rwops) -- negative test case, It should show error.
```

13.2.111. SDL_ RWops

Name	SDL_ RWops
Synopsis	require('SDL') SDL_ RWops - This is the structure of SDL_RWops
Description	These functions are used to provide an SDL_RWops structure from various sources. The names of the functions should describe what they do (FP is a file instance). Then of course, don't forget to free the structure when you've finished with it using SDL_RWclose() or SDL_FreeRW().
Return Value	none
Parameters	none

13.2.112. SDL_RWseek

Name	SDL_RWseek
Synopsis	require('SDL') SDL_RWseek(ctx, offset, whence)
Description	Return final offset in the data source
Parameters	ctx [in] : SDL_RWops structure offset [in] : An offset in bytes whence [in] : Flag to seek the position

Example:

```
local rwops = SDL_RWFromFile(chResPath.."test.txt","r")
if rwops == nil then
	print("Couldn't open test.txt")
	return 2
end

--	Seek to 0 bytes from the end of the file -- i.e. the exact end of file */
local offset1 = SDL_RWseek(rwops,0,RW_SEEK_SET)
local offset2 = SDL_RWseek(rwops,0,RW_SEEK_CUR)
local offset3 = SDL_RWseek(rwops,0,RW_SEEK_END)
SDL_RWclose(rwops)
if offset1<0 then
	print("Could not seek inside test.txt")
	return 2
end
if offset2<0 then
	print("Could not seek inside test.txt")
	return 2
end
```

```
if offset3<0 then
        print("Could not seek inside test.txt")
        return 2
end
print("SEEK_SET: ",offset1,"SEEK_CUR: ",offset2,"SEEK_END :",offset3)
return 0
```

13.2.113. SDL_RWtell

Name	SDL_RWseek
Synopsis	require('SDL') SDL_RWtell(ctx)
Description	Gets current position within a data source
Parameters	ctx [in] : SDL_RWops structure

Example:

```
local rwops = SDL_RWFromFile(chResPath.."test.txt","r")
if rwops == nil then
        print("Couldn't open test.txt")
        return 1
end

SDL_RWseek(rwops,0,RW_SEEK_SET)
print("Beginning position in test.txt: ",SDL_RWtell(rwops))
SDL_RWseek(rwops,0,RW_SEEK_CUR)
print("Current position in test.txt: ",SDL_RWtell(rwops))
SDL_RWseek(rwops,0,RW_SEEK_END)
print("Final position in test.txt: ",SDL_RWtell(rwops))
SDL_RWclose(rwops)
return 0
```

13.2.114. SDL_RWread

Name	SDL_RWread
Synopsis	require('SDL') SDL_RWread(ctx, ptr, size, n)
Description	Gets number of memory blocks read
Parameters	ctx [in] : SDL_RWops structure ptr [in] : area of memory to read data into size [in] : The size of the memory blocks to write n [in] : The exact number of memory blocks to write

Example:

```
local buf = SDL_malloc(256)
local rwops = SDL_RWFromFile(chResPath.."test.txt","r")

blocks=SDL_RWread(rwops,buf,16,256/16)
print("Number of blocks read :",blocks)
pstr = voidp_to_charp(buf)
print("Read from file :",pstr)
SDL_RWclose(rwops)

if blocks == -1 then
        print("Couldn't read from test.txt")
        return 2
else
        print("Read ",blocks ",16-byte blocks")
end
```

13.2.115. SDL_RWwrite

Name	SDL_RWread
Synopsis	require('SDL') SDL_RWwrite(ctx, ptr, size, n)
Description	Writes to a data source. Gets number of memory blocks read.
Parameters	ctx [in] : SDL_RWops structure ptr [in] : area in memory to read data from size [in] : The size of the memory blocks to write n [in] : The exact number of memory blocks to write

Example:

```
rwops = SDL_RWFromFile(chResPath.."test.txt","wb")
local written=SDL_RWwrite(rwops,charp_to_voidp("Maisnam Danny"),1,13)

if written == -1 then
	print("Couldn't write to test.txt")

else
	print("Wrote",written,"1-byte blocks")
end

SDL_RWclose(rwops)
```

13.2.116. SDL_RWclose

Name	SDL_RWread
Synopsis	require('SDL') SDL_RWclose()
Description	Closes a data source.
Return Value	Return 0 on success.
Parameters	ctx[in] : SDL_RWops structure

Example:

```
local rwops = SDL_RWFromFile(chResPath.."test.txt","r")
local crwops = SDL_RWclose(rwops)
if crwops == -1 then
	print("SDL_RWclose fail!!")
	return 1
end
if crwops == 0 then
	print("SDL_RWclose() passed!!")
end
```

SDL Extension Libraries Reference

SDL Image

13.2.117. IMG_Load

Name	IMG_Load
Synopsis	require(‘SDL_image’) IMG_Load(file)
Description	Load from a file
Return Value	New SDL surface.
Parameters	file[in] : image file name

Example :

```
local filename = “icon.bmp”
local SDL_Surface_BMP = IMG_Load(filename)
if(SDL_Surface_BMP ~= nil) then
        print("Pass!! Successfully loaded the Image")
else
        print ("Fail!! fail to load    the image")
end
```

13.2.118. IMG_Load_RW

Name	IMG_Load_RW
Synopsis	require('SDL_image') IMG_Load_RW(src, freesrc)
Description	Load using a SDL_RWop
Return Value	New SDL surface.
Parameters	src[in] : source instance of SDL RWops freesrc[in] : value

Example :

```
local fileName = "icon.bmp"
local imageSurface = IMG_Load_RW(SDL_RWFromFile(fileName,"rb"),1)
if(imageSurface ~= nil) then
        print("Pass, IMG_Load_RW!! Successfully loaded the Image")
else
        print ("Fail, IMG_Load_RW!! fail to load    the image")
end
```

13.2.119. IMG_LoadTyped_RW

Name	IMG_LoadTyped_RW
Synopsis	require(‘SDL_image’) IMG_LoadTyped_RW(src, freesrc, type)
Description	Load more format specifically with a SDL_RWop
Return Value	New SDL surface.
Parameters	src[in] : source instance of SDL RWops freesrc[in] : value type[in]: format type

Example :

```
local fileName = "icon.bmp"
local imageSurface = IMG_LoadTyped_RW(SDL_RWFromFile(fileName,"rb"), 1, “BMP”)
if(imageSurface ~= nil) then
        print("Pass, IMG_LoadTyped_RW!! Successfully loaded the Image")
else
        print ("Fail, IMG_LoadTyped_RW!! fail to load    the image")
end
```

13.2.120. IMG_LoadBMP_RW

Name	IMG_LoadBMP_RW
Synopsis	require('SDL_image') IMG_LoadBMP_RW(src)
Description	Load BMP using a SDL_RWop
Return Value	New SDL surface.
Parameters	src[in] : source instance of SDL RWops

Example :

```
local fileName = "icon.bmp"
local rwops_BMP = SDL_RWFromFile(filename, "rb")
local ret_BMP = IMG_LoadBMP_RW(rwops_BMP)
if(ret_BMP ~= nil) then
        print("Pass, IMG_LoadBMP_RW!! Successfully loaded the Image")
else
        print ("Fail, IMG_LoadBMP_RW!! fail to load    the image")
end
```

13.2.121. IMG_LoadGIF_RW

Name	IMG_LoadGIF_RW
Synopsis	require('SDL_image') IMG_LoadGIF_RW(src)
Description	Load GIF using a SDL_RWop
Return Value	New SDL surface.
Parameters	src[in] : source instance of SDL RWops

Example :

```
local fileName = " icon.gif"
local rwops_Gif = SDL_RWFromFile(filename, "rb")
local ret_Gif = IMG_LoadGIF_RW(rwops_Gif)
if(ret_Gif ~= nil) then
        print("Pass, IMG_LoadGIF_RW!! Successfully loaded the Image")
else
        print ("Fail, IMG_LoadGIF_RW!! fail to load    the image")
end
```

13.2.122. IMG_LoadJPG_RW

Name	IMG_LoadJPG_RW
Synopsis	require('SDL_image') IMG_LoadJPG_RW(src)
Description	Load JPG using a SDL_RWop
Return Value	New SDL surface.
Parameters	src[in] : source instance of SDL RWops

Example :

```
local fileName = " icon. jpg"
local rwops_JPG = SDL_RWFromFile(filename, "rb")
local ret_JPG = IMG_LoadJPG_RW(rwops_JPG)
if(ret_JPG ~= nil) then
        print("Pass, IMG_LoadJPG_RW!! Successfully loaded the Image")
else
        print ("Fail, IMG_LoadJPG_RW!! fail to load    the image")
end
```

13.2.123. IMG_LoadPNG_RW

Name	IMG_LoadPNG_RW
Synopsis	require('SDL_image') IMG_LoadPNG_RW(src)
Description	Load PNG using a SDL_RWop
Return Value	New SDL surface.
Parameters	src[in] : source instance of SDL RWops

Example :

```
local fileName = " icon.PNG"
local rwops_PNG = SDL_RWFromFile(filename, "rb")
local ret_PNG = IMG_LoadPNG_RW(rwops_PNG)
if(ret_PNG ~= nil) then
        print("Pass, IMG_LoadPNG_RW!! Successfully loaded the Image")
else
        print ("Fail, IMG_LoadPNG_RW!! fail to load    the image")
end
```

13.2.124. IMG_isBMP

Name	IMG_isBMP
Synopsis	require('SDL_image') IMG_isBMP(src)
Description	Test for valid if supported BMP file
Return Value	1 if the image is a BMP, 0 is returned otherwise
Parameters	src[in] : source instance of SDL RWops

Example :

```
local fileName = "icon.bmp"
local rwops_BMP = SDL_RWFromFile(filename, "rb")
local errRet = IMG_isBMP(rwops_BMP)
if(errRet == 1) then
	print("Pass, IMG_isBMP!!")
else
	print ("Fail, IMG_isBMP!!")
end
```

13.2.125. IMG_isGIF

Name	IMG_isGIF
Synopsis	require('SDL_image') IMG_isGIF(src)
Description	Test for valid if supported GIF file
Return Value	1 if the image is a GIF, 0 is returned otherwise
Parameters	src[in] : source pointer to SDL RWops

Example :

```
local fileName = "icon.gif"
local rwops_GIF = SDL_RWFromFile(filename, "rb")
local errRet = IMG_isGIF(rwops_GIF)
if(errRet == 1) then
        print("Pass, IMG_isGIF!!")
else
        print ("Fail, IMG_isGIF!!")
end
```

13.2.126. IMG_isJPG

Name	IMG_isJPG
Synopsis	require('SDL_image') IMG_isJPG(src)
Description	Test for valid if supported JPG file
Return Value	1 if the image is a JPG, 0 is returned otherwise
Parameters	src[in] : source pointer to SDL RWops

Example :

```
local fileName = "icon.jpg"
local rwops_JPG = SDL_RWFromFile(filename, "rb")
local errRet = IMG_isJPG(rwops_JPG)
if(errRet == 1) then
        print("Pass, IMG_isJPG!!")
else
        print ("Fail, IMG_isJPG!!")
end
```

13.2.127. IMG_isPNG

Name	IMG_isPNG
Synopsis	require('SDL_image') IMG_isPNG(src)
Description	Test for valid if supported PNG file
Return Value	1 if the image is a PNG, 0 is returned otherwise
Parameters	src[in] : source pointer to SDL RWops

Example :

```
local fileName = "icon.PNG"
local rwops_PNG = SDL_RWFromFile(filename, "rb")
local errRet = IMG_isPNG(rwops_PNG)
if(errRet == 1) then
        print("Pass, IMG_isPNG!!")
else
        print ("Fail, IMG_isPNG!!")
end
```

13.2.128. IMG_Linked_Version

Name	IMG_Linked_Version
Synopsis	require('SDL_image') IMG_Linked_Version (compile_version)
Description	This works similar to SDL_Linked_Version and SDL_VERSION. Using these you can compare the runtime version to the version that you compiled with.These functions/macros do not require any library initialization calls before using them.
Return Value	none
Parameters	compile_version[IN]: SDL_version

13.2.129. IMG_ SetError

Name	IMG_ SetError
Synopsis	require('SDL_image') IMG_ SetError (str,I)
Description	This is the same as SDL_SetError, which sets the error string which may be fetched with IMG_GetError (or SDL_GetError).
Return Value	none
Parameters	String[IN]: string to be set for error I[IN]:number

Example :

```
int myimagefunc(int i)
{
    IMG_SetError("myimagefunc is not implemented! %d was passed in.",i);
    return(-1);
}
```

13.2.130. IMG_ GetError

Name	IMG_ GetError
Synopsis	require('SDL_image') IMG_ GetError ()
Description	This is the same as SDL_GetError, which returns the last error set as a string which you may use to tell the user what happened when an error status has been returned from an SDL_image function call.
Return Value	a char instance (string) containing a humam readble version or the reason for the last error that occured.
Parameters	none

Example :

```
print("Oh My Goodness, an error : %s", IMG_GetError());
```

SDL Mixer

13.2.131. Mix_OpenAudio

Name	Mix_OpenAudio
Synopsis	require('SDL_mixer') Mix_OpenAudio(frequency,format,channels,chunksize)
Description	Open sound mixer
Return Value	0 on success, -1 on error
Parameters	frequency[in] : sample frequency format[in] : audio format channels[in] : no. of channels chunksize[in] : audio chunksize

Example :

```
local audio_rate = 22050            --Frequency of audio playback
local audio_format = AUDIO_S16          --Format of the audio we're playing
local audio_channels = 2                --2 channels = stereo
local audio_buffers = 4096          --Size of the audio buffers in memory


if Mix_OpenAudio(audio_rate, audio_format, audio_channels,
audio_buffers) ~= 0 then
        print("Unable to initialize audio: ", Mix_GetError())
        return 0
end
```

13.2.132. Mix_CloseAudio

Name	Mix_CloseAudio
Synopsis	require(‘SDL_mixer’) Mix_CloseAudio()
Description	Close sound mixer
Return Value	
Parameters	

Example :

```
local audio_rate = 22050            --Frequency of audio playback
local audio_format = AUDIO_S16          --Format of the audio we're playing
local audio_channels = 2                --2 channels = stereo
local audio_buffers = 4096          --Size of the audio buffers in memory


if Mix_OpenAudio(audio_rate, audio_format, audio_channels,
audio_buffers) ~= 0 then
        print("Unable to initialize audio: ", Mix_GetError())
        return 0
end
--
Mix_CloseAudio()
```

13.2.133. Mix_QuerySpec

Name	Mix_QuerySpec
Synopsis	require('SDL_mixer') Mix_QuerySpec(frequency, format, channels)
Description	Get output format
Return Value	0 on error, no. of times device was opened on success
Parameters	frequency[out] : pointer to frequency format[out] : pointer to format channels[out] : pointer to channels

Example :

```
local frequency = new_intp()
local format = new_Uint16p()
local channels = new_intp()
if Mix_QuerySpec(frequency,format,channels)== 0 then
        print("Error in getting Mixer Parameter : ", Mix_GetError())
end
print(" Actual Mixer Parameter: ")
print("   Frequency : ", intp_value(frequency))
print("   Format : ", Uint16p_value(format))
print("   Channels : ", intp_value(channels))
```

13.2.134. Mix_LoadWAV

Name	Mix_LoadWAV
Synopsis	require(‘SDL_mixer’) Mix_LoadWAV(file)
Description	Load From a file
Return Value	Whole sample on the success, nil on error
Parameters	file[in] : WAV file name

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"

--Load our WAV file from disk
local file = chResPath.."sample.wav"
sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError());
end
```

13.2.135. Mix_LoadWAV_RW

Name	Mix_LoadWAV_RW
Synopsis	require('SDL_mixer') Mix_LoadWAV_RW(src, freesrc)
Description	Load From a file Using RWops
Return Value	Whole sample on the success, nil on error
Parameters	src[in] : instance of SDL RWops freesrc[in] : flag to free

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

rwfp = SDL_RWFromFile(file, "rb")
sound = Mix_LoadWAV_RW(rwfp,1)
if sample == nil then
    print("Mix_LoadWAV_RW: ", Mix_GetError())
end
```

13.2.136. Mix_VolumeChunk

Name	Mix_VolumeChunk
Synopsis	require(‘SDL_mixer’) Mix_VolumeChunk(chunk, volume)
Description	Set mix volume
Return Value	Previous volume of the chunk
Parameters	chunk[in] : Instance of Mix Chunk

Example :

```
-- set the sample's volume to ½
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end

local previous_volume
previous_volume = Mix_VolumeChunk(sample, MIX_MAX_VOLUME/2)
print("previous_volume: ", previous_volume)
```

13.2.137. Mix_FreeChunk

Name	Mix_FreeChunk
Synopsis	require('SDL_mixer') Mix_FreeChunk(chunk)
Description	Set mix volume
Return Value	
Parameters	chunk[in]: Instance of Mix Chunk

Example :

```
-- set the sample's volume to ½
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end

local previous_volume
previous_volume = Mix_VolumeChunk(sample, MIX_MAX_VOLUME/2)
print("previous_volume: ", previous_volume)
--
Mix_FreeChunk(sound)
```

13.2.138. Mix_AllocateChannels

Name	Mix_AllocateChannels
Synopsis	require(‘SDL_mixer’) Mix_AllocateChannels(numchans)
Description	Set the number of channels to mix
Return Value	Number of channels it was able to allocate
Parameters	numchans[in] : number of channels

Example :

```
local allocate = Mix_AllocateChannels(2)
print("No. of allocated channel = ", allocate)
```

13.2.139. Mix_Volume

Name	Mix_Volume
Synopsis	require('SDL_mixer') Mix_Volume(channel, volume)
Description	Set the mix volume of a channel
Return Value	Number of channels it was able to allocate
Parameters	channel[in] : no. of channel volume[in] : volume to be set

Example :

```
local cur_volume = Mix_Volume(1,MIX_MAX_VOLUME)
print("Current volume of channel: ", cur_volume)
```

13.2.140. Mix_PlayChannel

Name	Mix_PlayChannel
Synopsis	require('SDL_mixer') Mix_PlayChannel(channel, chunks, loops)
Description	Play loop
Return Value	No of Channel on success, -1 on error
Parameters	channel[in] : no of channels chunk[in] : Instance of Mix Chunk loops[in] : no. of loops

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
if Mix_Playing(-1) ~= 0 then
        Mix_HaltChannel(-1)
end
channel = Mix_PlayChannel(-1, sound, 1)
print("[PLAY]Sample is played on channel : " ,channel)
if channel == -1 then
        print("Unable to play WAV file: ", Mix_GetError())
end
```

13.2.141. Mix_PlayChannelTimed

Name	Mix_PlayChannelTimed
Synopsis	require('SDL_mixer') Mix_PlayChannelTimed(channel, chunks, loops, ticks)
Description	Play loop and limit by time
Return Value	No of Channel on success, -1 on error
Parameters	channel[in] : no of channels chunk[in] : instance of Mix Chunk loops[in] : no. of loops ticks[in] : value in milisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
channel = Mix_PlayChannelTimed(-1, sound, -1,1000)
print("[PLAY_TIMED]Sample is played on channel : " ,channel)
if channel == -1 then
        print("Unable to play WAV file: ", Mix_GetError())
end
```

13.2.142. Mix_FadeInChannel

Name	Mix_FadeInChannel
Synopsis	require('SDL_mixer') Mix_FadeInChannel(channel, chunks, loops, ms)
Description	Play loop with fade in
Return Value	No of Channel on success, -1 on error
Parameters	channel[in] : no of channels chunk[in] : instance of Mix Chunk loops[in] : no. of loops ms[in] : value in millisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
--
local fChannel = Mix_FadeInChannel(1, sound, -1,2000)
print("[FADEIN]Sample is played on channel : ",fChannel)
if fChannel == -1 then
        print("Unable to FadeIn WAV file: ", Mix_GetError())
end
```

13.2.143. Mix_FadeInChannelTimed

Name	Mix_FadeInChannelTimed
Synopsis	require('SDL_mixer') Mix_FadeInChannelTimed* channel, chunks, loops, ms, ticks)
Description	Play loop with fade in and limit by time
Return Value	No of Channel on success, -1 on error
Parameters	channel[in] : no of channels chunk[in] : instance of Mix Chunk loops[in] : no. of loops ms[in] : value in millisecond ticks[in] : value in milisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
local fChannel = Mix_FadeInChannelTimed(1, sound, 1,2000,fade_timed)
print("[FADE_TIMED]Sample is played on channel : ",fChannel)
if fChannel == -1 then
        print("Unable to FadeIn WAV file in " ..fade_timed .." : ",
Mix_GetError())
end
```

13.2.144. Mix_Pause

Name	Mix_Pause
Synopsis	require('SDL_mixer') Mix_Pause(channel)
Description	Pause a channel
Return Value	
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
--
channel = Mix_PlayChannel(-1, sound, 1)
Mix_Pause(channel)
```

13.2.145. Mix_Resume

Name	Mix_Resume
Synopsis	require('SDL_mixer') Mix_Resume(channel)
Description	Resume a paused channel
Return Value	
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
channel = Mix_PlayChannel(-1, sound, 1)
Mix_Pause(channel)
if Mix_Paused(channel) == 0 then
        Mix_Pause(channel)
else
        Mix_Resume(channel)
end
```

13.2.146. Mix_HaltChannel

Name	Mix_HaltChannel
Synopsis	require('SDL_mixer') Mix_HaltChannel(channel)
Description	Stop playing on a channel
Return Value	0 all the time
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
channel = Mix_PlayChannel(-1, sound, 1)
if Mix_Playing(-1) ~= 0 then
        Mix_HaltChannel(-1)
end
```

13.2.147. Mix_ExpireChannel

Name	Mix_ExpireChannel
Synopsis	require('SDL_mixer') Mix_ExpireChannel(channel, ticks)
Description	Change the timed stoppage of a channel
Return Value	Number of channel stopped
Parameters	channel[in] : no. of channel ticks[in] : value in millisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
--
channel = Mix_PlayChannel(-1, sound, 1)
print("------    Expire channel after 5 seconds -----")
Mix_ExpireChannel(-1,5000)
```

13.2.148. Mix_FadeOutChannel

Name	Mix_FadeOutChannel
Synopsis	require('SDL_mixer') Mix_FadeOutChannel(channel, ms)
Description	Stop playing channel after timed fade out
Return Value	Number of channel faded out
Parameters	channel[in] : no. of channel ms[in] : value in millisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
fadeout_timed = 5000
local fChannel = Mix_FadeOutChannel(-1, fadeout_timed)
print("[FADE_OUT]No. of fadded out channel: ", fChannel)
if fChannel == -1 then
        print("Unable to FadeOut WAV file in " ..fadeout_timed .." : ",
Mix_GetError())
end
```

13.2.149. Mix_ChannelFinished

Name	Mix_ChannelFinished
Synopsis	require('SDL_mixer') Mix_ChannelFinished(channel_finished)
Description	Set callback for when channel finishes playing
Return Value	
Parameters	channel_finished[in] : Function

Example :

```
function channelDone(channel)
	print("channel finished playback.",channel)
end

chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil	then
		print("Unable to load WAV file: ", Mix_GetError())
		return 0
end
--
--
channel = Mix_PlayChannel(-1, sound, 1)

Mix_ChannelFinished("channelDone")
```

13.2.150. Mix_Playing

Name	Mix_Playing
Synopsis	require('SDL_mixer') Mix_Playing(channel)
Description	Get the active playing status of a channel
Return Value	0 if the channel is not playing, or no. of channels playing
Parameters	channel[in] : no. of channel

Example :

```
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
--
channel = Mix_PlayChannel(-1, sound, 1)
if Mix_Playing(-1) ~= 0 then
        print("Channel is playing")
end
```

13.2.151. Mix_Paused

Name	Mix_Paused
Synopsis	require('SDL_mixer') Mix_Paused(channel)
Description	Get the pause status of a channel
Return Value	0 if the channel is not paused, or no. of channels paused
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
--
channel = Mix_PlayChannel(-1, sound, 1)
Mix_Pause(channel)
if Mix_Paused(channel) == 0 then
        Mix_Pause(channel)
else
        Mix_Resume(channel)
end
```

13.2.152. Mix_FadingChannel

Name	Mix_FadingChannel
Synopsis	require('SDL_mixer') Mix_FadingChannel(channel)
Description	Get the pause status of a channel
Return Value	Fading status
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"
sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
channel = Mix_PlayChannel(-1, sound, 1)
local status = Mix_FadingChannel(channel)
if status == MIX_NO_FADING then
        print("Not fading.")
end
if status == MIX_FADING_OUT then
        print("Fading out.")
end
if status == MIX_FADING_IN then
        print("Fading in.")
end
```

13.2.153. Mix_GetChunk

Name	Mix_GetChunk
Synopsis	require('SDL_mixer') Mix_GetChunk(channel)
Description	Get the sample playing on a channel
Return Value	Pointer to Mix Chunk, nil on error
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
channel = Mix_PlayChannel(-1, sound, 1)
local pMixChunk = Mix_GetChunk(channel)
if pMixChunk == nil then
        print("Channel is not allocated or channel has not played any
samples yet.", Mix_GetError())
else
print("Mix_Chunk* last in use on channel 0 was: " , pMixChunk.allocated)
end
```

13.2.154. Mix_ReserveChannels

Name	Mix_ReserveChannels
Synopsis	require('SDL_mixer') Mix_ReserveChannels(channel)
Description	Prevent channels from being used in default group
Return Value	No. of channels reserved
Parameters	channel[in] : no. of channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."sample.wav"

sound = Mix_LoadWAV(file)
if sound == nil    then
        print("Unable to load WAV file: ", Mix_GetError())
        return 0
end
--
--
channel = Mix_PlayChannel(-1, sound, 1)
local reserve = Mix_ReserveChannels(1)
print("Total number of Reserve channel : ", reserve)
```

13.2.155. Mix_GroupCount

Name	Mix_GroupCount
Synopsis	require('SDL_mixer') Mix_GroupCount(tag)
Description	Get number of channels in group
Return Value	No of channels found in the group
Parameters	tag[in] : group number

Example :

```
local nChannel = Mix_GroupCount(-1)
print(" Number of channel in group 1 : ", nChannel)
```

13.2.156. Mix_GroupAvailable

Name	Mix_GroupAvailable
Synopsis	require('SDL_mixer') Mix_GroupAvailable(tag)
Description	Get first inactive channel in group
Return Value	No of channels found in the group, or -1 if no channel found
Parameters	tag[in] : group number

Example :

```
local avaChannel = Mix_GroupAvailable(-1)
print(" First available (not playing) channel : ", avaChannel)
```

13.2.157. Mix_GroupOldest

Name	Mix_GroupOldest
Synopsis	require('SDL_mixer') Mix_GroupOldest(tag)
Description	Get oldest busy channel in group
Return Value	No of channels found in the group, or -1 if no channel found
Parameters	tag[in] : group number

Example :

```
local oChannel = Mix_GroupOldest(-1)
print(" Oldest actively playing channel : ", oChannel)
```

13.2.158. Mix_GroupNewer

Name	Mix_GroupNewer
Synopsis	require('SDL_mixer') Mix_GroupNewer(tag)
Description	Get youngest busy channel in group
Return Value	No of channels found in the group, or -1 if no channel found
Parameters	tag[in] : group number

Example :

```
local newChannel = Mix_GroupNewer(-1)
print(" Newest actively playing channel : ", newChannel)
```

13.2.159. Mix_FadeOutGroup

Name	Mix_FadeOutGroup
Synopsis	require('SDL_mixer') Mix_FadeOutGroup(tag, ms)
Description	Fade out a group over time
Return Value	No of channels faded out in group
Parameters	tag[in] : group number ms[in] : value in millisecond

Example :

```
local fChannel = Mix_FadeOutGroup(1,1000)
print(" No. of channel set to fade out : ", fChannel)
```

13.2.160. Mix_HaltGroup

Name	Mix_HaltGroup
Synopsis	require(‘SDL_mixer’) Mix_HaltGroup(tag)
Description	Stop a group
Return Value	Always returns 0
Parameters	tag[in] : group number

Example :

```
local hGroup = Mix_HaltGroup(1)
print("Halt Group :", hGroup)
```

13.2.161. Mix_LoadMUS

Name	Mix_LoadMUS
Synopsis	require('SDL_mixer') Mix_LoadMUS(file)
Description	Load a music file
Return Value	Pointer to Mix Music, nil on error
Parameters	file[in] : music file name

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
```

13.2.162. Mix_FreeMusic

Name	Mix_FreeMusic
Synopsis	require('SDL_mixer') Mix_FreeMusic(music)
Description	Free music
Return Value	
Parameters	music[in] : pointer to Mix Music

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_FreeMusic(music)
```

13.2.163. Mix_PlayMusic

Name	Mix_PlayMusic
Synopsis	require('SDL_mixer') Mix_PlayMusic(music, loops)
Description	Play music, with looping
Return Value	0 on success, -1 on error
Parameters	music[in] : pointer to Mix Music loops[in]: no of loops

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil  then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
if Mix_PlayMusic(music, 1) == -1 then
        print("Mix_PlayMusic : Fail",Mix_GetError())
else
        print("Mix_PlayMusic : Pass")
end
```

13.2.164. Mix_FadeInMusic

Name	Mix_FadeInMusic
Synopsis	require('SDL_mixer') Mix_FadeInMusic(music, loops, ms), position)
Description	Play music, with looping, and fade in
Return Value	0 on success, -1 on error
Parameters	music[in] : pointer to Mix Music loops[in]: no of loops ms[in] : value in millisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
if Mix_FadeInMusic(music, 1, 5000) == -1 then
        print("Mix_FadeInMusic : Fail",Mix_GetError())
else
        print("Mix_FadeInMusic : Pass")
end
```

13.2.165. Mix_FadeInMusicPos

Name	Mix_FadeInMusicPos
Synopsis	require('SDL_mixer') Mix_FadeInMusicPos(music, loops, ms, position)
Description	Play music from a start point, with looping, and fade in
Return Value	0 on success, -1 on error
Parameters	music[in] : pointer to Mix Music loops[in]: no of loops ms[in] : value in millisecond position[in] : position value

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
if Mix_FadeInMusicPos(music, 1, 5000, 20) == -1 then
        print("Mix_FadeInMusicPos : Fail",Mix_GetError())
else
        print("Mix_FadeInMusicPos : Pass")
end
```

13.2.166. Mix_HookMusic

Name	Mix_HookMusic
Synopsis	require('SDL_mixer') Mix_HookMusic(mix_func, arg)
Description	Hook for a custom music player
Return Value	
Parameters	mix_func[in] : function arg[out]: music position

Example :

```
-- make a music play function
-- it expects udata to be a pointer to an int
function myMusicPlayer(udata, stream, len)
        -- do something
end
...
-- use myMusicPlayer for playing...uh...music
Mix_HookMusic(myMusicPlayer, music_pos)
```

13.2.167. Mix_VolumeMusic

Name	Mix_VolumeMusic
Synopsis	require(‘SDL_mixer’) Mix_VolumeMusic(volume)
Description	Set music volume
Return Value	Previous volume setting
Parameters	volume[in] : volume

Example :

```
local vol_level = Mix_VolumeMusic(100)
print("Previous volume level: ", vol_level)
```

13.2.168. Mix_PauseMusic

Name	Mix_PauseMusic
Synopsis	require(‘SDL_mixer’) Mix_PauseMusic()
Description	Pause music
Return Value	
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_PlayMusic(music, 1)
Mix_PauseMusic()
```

13.2.169. Mix_ResumeMusic

Name	Mix_ResumeMusic
Synopsis	require(‘SDL_mixer’) Mix_ResumeMusic()
Description	Resume paused music
Return Value	
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_PlayMusic(music, 1)
Mix_PauseMusic()
if Mix_PausedMusic() == 0 then
        Mix_PauseMusic()
else
        Mix_ResumeMusic()
end
```

13.2.170. Mix_RewindMusic

Name	Mix_RewindMusic
Synopsis	require('SDL_mixer') Mix_RewindMusic()
Description	Rewind music to beginning
Return Value	
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_PlayMusic(music, 1)
print("Rewind Music")
Mix_RewindMusic()
```

13.2.171. Mix_SetMusicPosition

Name	Mix_SetMusicPosition
Synopsis	require('SDL_mixer') Mix_SetMusicPosition(position)
Description	Set position of playback in stream
Return Value	0 on success, -1 on error
Parameters	position[in]: position to play from

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
print("Setting music position to 50")
if Mix_PlayingMusic() ~= 0 then
        Mix_HaltMusic()
end
Mix_SetMusicPosition(150)
Mix_PlayMusic(music, 1)
```

13.2.172. Mix_SetMusicCMD

Name	Mix_SetMusicCMD
Synopsis	require('SDL_mixer') Mix_SetMusicCMD(command)
Description	Use external program for music playback
Return Value	0 on success, -1 on error
Parameters	Command[in]: system command to play the music

Example :

```
SDL_putenv("name = 10")
local env = SDL_getenv("name")
print(env)
if Mix_SetMusicCMD(env) == -1 then
        print("Mix_SetMusicCMD", Mix_GetError())
end
```

13.2.173. Mix_HaltMusic

Name	Mix_HaltMusic
Synopsis	require(‘SDL_mixer’) Mix_HaltMusic()
Description	Stop music playback
Return Value	Always returns 0
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_PlayMusic(music, 1)
if Mix_PlayingMusic() ~= 0 then
        Mix_HaltMusic()
end
```

13.2.174. Mix_FadeOutMusic

Name	Mix_FadeOutMusic
Synopsis	require('SDL_mixer') Mix_FadeOutMusic(ms)
Description	Stop music playback
Return Value	1 on success, 0 on failure.
Parameters	ms[in] : value in millisecond

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_PlayMusic(music, 1)
if Mix_FadeOutMusic(5000) == 0 then
        print("Mix_FadeOutMusic : Fail",Mix_GetError())
else
        print("Mix_FadeOutMusic : Pass")
end
```

13.2.175. Mix_HookMusicFinished

Name	Mix_HookMusicFinished
Synopsis	require('SDL_mixer') Mix_HookMusicFinished(music_finished)
Description	Set a callback for when music stops
Return Value	
Parameters	music_finished[in] : Function

Example :

```
-- make a music finished function
function musicFinished()
    print("Music stopped.\n")
end
...
-- use musicFinished for when music stops
Mix_HookMusicFinished(musicFinished)
```

13.2.176. Mix_GetMusicType

Name	Mix_GetMusicType
Synopsis	require('SDL_mixer') Mix_GetMusicType(music)
Description	Get the music encoding type
Return Value	Type of music
Parameters	music[in] : music to get the type of

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
--
Mix_PlayMusic(music, 1)

if Mix_GetMusicType(nil) == MUS_NONE then
        print("Command based music is playing.")
end
if Mix_GetMusicType(nil) == MUS_CMD then
        print("Command based music is playing.")
end
if Mix_GetMusicType(nil) == MUS_WAV then
```

```
        print("WAVE/RIFF music is playing.")
end
if Mix_GetMusicType(nil) == MUS_MOD then
        print("MOD music is playing.")
end
if Mix_GetMusicType(nil) == MUS_MID then
        print("MID music is playing.")
end
if Mix_GetMusicType(nil) == MUS_OGG then
        print("OGG music is playing.")
end
if Mix_GetMusicType(nil) == MUS_MP3 then
        print("MP3 music is playing.")
end
```

13.2.177. Mix_PlayingMusic

Name	Mix_PlayingMusic
Synopsis	require('SDL_mixer') Mix_PlayingMusic()
Description	Test whether music is playing
Return Value	0 if the music is not playing, or 1 if it is playing.
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
if Mix_PlayingMusic() ~= 0 then
        print("Music is Playing")
end
```

13.2.178. Mix_PausedMusic

Name	Mix_PausedMusic
Synopsis	require('SDL_mixer') Mix_PausedMusic()
Description	Test whether music is paused
Return Value	0 if music is not paused. 1 if it is paused
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
--
Mix_PlayMusic(music, 1)
Mix_PauseMusic()
if Mix_PausedMusic() == 0 then
        Mix_PauseMusic()
else
        Mix_ResumeMusic()
end
```

13.2.179. Mix_FadingMusic

Name	Mix_FadingMusic
Synopsis	require('SDL_mixer') Mix_FadingMusic()
Description	Get status of current music fade activity
Return Value	Fading status
Parameters	

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
--
Mix_PlayMusic(music, 1)
if Mix_FadeOutMusic(5000) == 0 then
        print("Mix_FadeOutMusic : Fail",Mix_GetError())
else
        print("Mix_FadeOutMusic : Pass")
end
local status = Mix_FadingMusic()
if status == MIX_NO_FADING then
        print("Not fading music.")
```

```
end
if status == MIX_FADING_OUT then
        print("Fading out music.")
end
if status == MIX_FADING_IN then
        print("Fading in music.")
end
```

13.2.180. Mix_GetMusicHookData

Name	Mix_GetMusicHookData
Synopsis	require(‘SDL_mixer’) Mix_GetMusicHookData()
Description	Retrieve the arg
Return Value	
Parameters	

Example :

```
-- retrieve the music hook data pointer
data=Mix_GetMusicHookData()
```

13.2.181. Mix_RegisterEffect

Name	Mix_RegisterEffect
Synopsis	require('SDL_mixer') Mix_RegisterEffect(channel, Effectfunc, EffectDone, arg)
Description	Hook a processor to a channel
Return Value	0 on errors
Parameters	channel[in] : channel number Effectfunc[in] : Function EffectDone[in]: Function Arg[in]: pointer to data

Example :

```
function noEffect(chan, stream, len,udata)
      print("Lenth and Channel number",len,chan)
end

print("------    Testing of Register effect    -----")
if Mix_RegisterEffect(MIX_CHANNEL_POST, "noEffect", "", nil) == 0 then
          print("Mix_RegisterEffect: ", Mix_GetError())
else
          print("Mix_RegisterEffect: PASS")
end
```

13.2.182. Mix_UnregisterEffect

Name	Mix_UnregisterEffect
Synopsis	require('SDL_mixer') Mix_UnregisterEffect(channel, EffectFunc)
Description	Unhook a processor from a channel
Return Value	0 on errors
Parameters	channel[in] : channel number Effectfunc[in] : Function

Example :

```
function noEffect(chan, stream, len,udata)
      print("Lenth and Channel number",len,chan)
end

print("------    Testing of Unregister effect    -----")
if Mix_UnregisterEffect(-2, "noEffect") == 0 then
          print("Mix_UnregisterEffect: ", Mix_GetError())
else
          print("Mix_UnregisterEffect: PASS")
end
```

13.2.183. Mix_UnregisterAllEffects

Name	Mix_UnregisterAllEffects
Synopsis	require('SDL_mixer') Mix_UnregisterAllEffects(channel)
Description	Unhook all processors from a channel
Return Value	0 on errors
Parameters	channel[in] : channel number

Example :

```
print("------    Testing of Unregister effect    -----")
if Mix_UnregisterAllEffects(-2) == 0 then
        print("Mix_UnregisterAllEffects: ", Mix_GetError())
else
        print("Mix_UnregisterAllEffects: PASS")
end
```

13.2.184. Mix_SetPostMix

Name	Mix_SetPostMix
Synopsis	require('SDL_mixer') Mix_SetPostMix(mix_func, args)
Description	Hook in a postmix processor
Return Value	
Parameters	mix_func[in] : Function args[in] : pointer to data

Example :

```
-- make a pass thru processor function that does nothing...
function noEffect(udata, stream, len)
	-- you could work with stream here...
end
...
-- register noEffect as a postmix processor
Mix_SetPostMix(noEffect, nil)
```

13.2.185. Mix_SetPanning

Name	Mix_SetPanning
Synopsis	require('SDL_mixer') Mix_SetPanning(channel, left, right)
Description	Stereo panning
Return Value	0 on errors
Parameters	channel[in] : channel number left[in] : volume for the left channel right[in] : volume for the right channel

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
Mix_PlayMusic(music, 1)
if Mix_SetPanning(MIX_DEFAULT_CHANNELS, 0, 255) == 0 then
        print("Mix_SetPanning:Fail",Mix_GetError())
else
        print("Mix_SetPanning:Pass")
end
```

13.2.186. Mix_SetDistance

Name	Mix_SetDistance
Synopsis	require('SDL_mixer') Mix_SetDistance(*channel, distance)*
Description	Distance attenuation (volume)
Return Value	0 on errors
Parameters	channel[in] : channel number distance[in] : distance from the listener

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
--
Mix_PlayMusic(music, 1)
if Mix_SetDistance(MIX_CHANNEL_POST, 220) == 0 then
        print("Mix_SetDistance : Fail",Mix_GetError())
else
        print("Mix_SetDistance : Pass")
end
```

13.2.187. Mix_SetPosition

Name	Mix_SetPosition
Synopsis	require('SDL_mixer') Mix_SetPosition(channel, angle , distance)
Description	Panning(angular) and distance
Return Value	0 on errors
Parameters	channel[in] : channel number angle[in] : Direction distance[in] : distance from the listener

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
--
Mix_PlayMusic(music, 1)
if Mix_SetPosition(MIX_CHANNEL_POST,125,100) == 0 then
        print("Mix_SetPosition : Fail ",Mix_GetError())
else
        print("Mix_SetPosition : Pass ")
end
```

13.2.188. Mix_Linked_Version

Name	Mix_Linked_Version – This works similar to SDL_Linked_Version and SDL_VERSION.
Synopsis	require('SDL_image') Mix_Linked_Version(&compile_version)
Description	compare the runtime version to the version that you compiled
Return Value	Return the current version
Parameters	&compile_version[IN]: SDL_version

Example :

```
compile_version  -- SDL_version
link_version=Mix_Linked_Version()
SDL_MIXER_VERSION(compile_version);
print("compiled with SDL_mixer version: ",
        compile_version.major,
        compile_version.minor,
        compile_version.patch);
print("running with SDL_mixer version: ",
        link_version.major,
        link_version.minor,
        link_version.patch);
```

13.2.189. Mix_ SetError

Name	Mix_SetError – This is the same as SDL_SetError.
Synopsis	require('SDL_image') Mix_SetError (str,i)
Description	This is the same as SDL_SetError, which sets the error string which may be fetched with Mix_GetError (or SDL_GetError).
Return Value	none
Parameters	str[IN]: string for error message i [IN]: number

Example :

```
int mymixfunc(int i)
 {
    Mix_SetError("mymixfunc is not implemented! %d was passed in.",i);
    return(-1);
}
```

13.2.190. Mix_ GetError

Name	Mix_ GetError – This is the same as SDL_GetError.
Synopsis	require('SDL_image') Mix_ GetError ()
Description	This is the same as SDL_GetError, which returns the last error set as a string which you may use to tell the user what happened when an error status has been returned from an SDL_mixer function call.
Return Value	Return the string of error message
Parameters	none

Example :

```
printf("Oh My Goodness, an error : %s", Mix_GetError());
```

13.2.191. Mix_SetReverseStereo

Name	Mix_SetReverseStereo
Synopsis	require('SDL_mixer') Mix_SetReverseStereo(channel, flip)
Description	Swap stereo left and right
Return Value	0 on errors
Parameters	channel[in] : channel number flip[in]: effect to register or unregister

Example :

```
chResPath = "/mtd_down/widgets/user/Test/Res/"
local file = chResPath.."SDD_Intro.mp3"
music = Mix_LoadMUS(file)

if music == nil    then
        print("Unable to load music file: ", Mix_GetError())
        return 0
end
--
--
Mix_PlayMusic(music, 1)

if Mix_SetReverseStereo(MIX_CHANNEL_POST,1) == 0 then
        print("Mix_SetReverseStereo : Fail ",Mix_GetError())
else
        print("Mix_SetReverseStereo : Pass ")
end
```

SDL Net

13.2.192. SDLNet Init

Name	SDLNet Init
Synopsis	require(‘SDL_net’) SDLNet_Init()
Description	Initialize the network API
Return Value	-1 on error 0 on success
Parameters	

Example :

```
if SDLNet_Init()==-1 then
print("SDLNet_Init: ",SDLNet_GetError())
        return err
end
```

13.2.193. SDLNet_Quit

Name	SDLNet_Quit
Synopsis	require(‘SDL_net’) SDLNet_Quit()
Description	Shutdown and cleanup the network API
Return Value	
Parameters	

Example :

```
if SDLNet_Init()==-1 then
        print("SDLNet_Init: ",SDLNet_GetError())
        return err
end

-- shutdown SDL_net
SDLNet_Quit()
```

13.2.194. SDLNet ResolveHost

Name	SDLNet ResolveHost
Synopsis	require('SDL_net') SDLNet_ResolveHost(address, host, port)
Description	Resolve the string host, and fill in the IPaddress pointed to by address with the resolved IP and the port number passed in through port
Return Value	Status
Parameters	address [out] : IP address host [in] : string, port [in] : port number

Example :

```
--For a server listening on all interfaces, on port 1234
-- create a server type IPaddress on port 1234
ipaddress = new_IPaddress()
SDLNet_ResolveHost(ipaddress, nil, 1234)
```

13.2.195. SDLNet ResolveIP

Name	SDLNet ResolveIP
Synopsis	require('SDL_net') SDLNet_ResolveIP(ip)
Description	Resolve the IPv4 numeric address in address->host, and return the hostname as a string
Return Value	Host name buffer
Parameters	ip [in] : IP address

Example :

```
-- resolve the host name of the address in ipaddress
Ipaddress = new_IPaddress()
If host=SDLNet_ResolveIP(ipaddress) then
    print("SDLNet_ResolveIP: ", SDLNet_GetError())
    return err
end
```

13.2.196. SDLNet_TCP_Open

Name	SDLNet_TCP_Open
Synopsis	require('SDL_net') SDLNet_TCP_Open(ip)
Description	Connect to the host and port contained in ip using a TCP connection
Return Value	TCP socket
Parameters	ip [in] : IP address

Example :

```
ip = IPaddress_new()
port = 3366
ip_address = "107.108.89.131"
-- Resolve the argument into an IPaddress type
if SDLNet_ResolveHost(ip,ip_address,port)==-1 then
	print("Error in SDLNet_ResolveHost: ",SDLNet_GetError())
end

-- open the server socket
sock = SDLNet_TCP_Open(ip)

if sock == nil then
	print("Error in SDLNet_TCP_Open: ",SDLNet_GetError())
end
```

13.2.197. SDLNet_TCP_Close

Name	SDLNet_TCP_Close
Synopsis	require('SDL_net') SDLNet_TCP_Close(sock)
Description	This shutsdown, disconnects, and closes the TCPsocket sock
Return Value	
Parameters	sock [in] : TCPsocket

Example :

```
ip = IPaddress_new()
port = 3366
ip_address = "107.108.89.131"
-- Resolve the argument into an IPaddress type
if SDLNet_ResolveHost(ip,ip_address,port)==-1 then
	print("Error in SDLNet_ResolveHost: ",SDLNet_GetError())
end

-- open the server socket
sock = SDLNet_TCP_Open(ip)

if sock == nil then
	print("Error in SDLNet_TCP_Open: ",SDLNet_GetError())
end
SDLNet_TCP_Close(sock)
```

13.2.198. SDLNet_TCP_Accept

Name	SDLNet_TCP_Accept
Synopsis	require('SDL_net') SDLNet_TCP_Accept(server)
Description	Accept an incoming connection on the server TCPsocket
Return Value	TCP socket
Parameters	server [in] : TCPsocket

Example :

```
ip = IPaddress_new()
port = 3366
ip_address = "107.108.89.131"
-- Resolve the argument into an IPaddress type
if SDLNet_ResolveHost(ip,ip_address,port)==-1 then
        print("Error in SDLNet_ResolveHost: ",SDLNet_GetError())
end

-- open the server socket
sock = SDLNet_TCP_Open(ip)

if sock == nil then
        print("Error in SDLNet_TCP_Open: ",SDLNet_GetError())
end

client=SDLNet_TCP_Accept(sock)
```

13.2.199. SDLNet_TCP_GetPeerAddress

Name	SDLNet_TCP_GetPeerAddress
Synopsis	require('SDL_net') SDLNet_TCP_GetPeerAddress(sock)
Description	Get the Peer's (the other side of the connection, the remote side, not the local side) IP address and port number.
Return Value	IP address
Parameters	sock [in] : TCPsocket

Example :

```
client=SDLNet_TCP_Accept(sock)
remoteip= SDLNet_TCP_GetPeerAddress(client)
print("remoteip = ",remoteip)
```

13.2.200. SDLNet_TCP_Send

Name	SDLNet_TCP_Send
Synopsis	require('SDL_net') SDLNet_TCP_Send(sock, data, len)
Description	Send data of length len over the socket sock
Return Value	Size of data sent
Parameters	sock [in] : TCPsocket data [in] : data buffer len [in] : length of data

Example :

```
sock = SDLNet_TCP_Open(ip)
..
print("Enter Message, or Q to make the server quit:")
message = io.stdin:read()
len = string.len(message)
msg = charp_to_voidp(message)
…
result = SDLNet_TCP_Send(sock,msg,len)
if result < len then
        print("SDLNet_TCP_Send: ",SDLNet_GetError())
end
```

13.2.201. SDLNet_TCP_Recv

Name	SDLNet_TCP_Recv
Synopsis	require('SDL_net') SDLNet_TCP_Recv(sock, data, maxlen)
Description	Receive data of exactly length maxlen bytes from the socket sock, into the memory pointed to by data
Return Value	Size of data received
Parameters	sock [in] : TCPsocket data [in] : data buffer maxlen [in] : length

Example :

```
server=SDLNet_TCP_Open(ip)
client=SDLNet_TCP_Accept(server)
…
local msg = tovoidp_char(message)
len=SDLNet_TCP_Recv(client,msg,1024)
if(len ~= 0) then
        m = voidp_to_charp(msg)
        print("Received: ",len.."message = ", m)
end
```

13.2.202. SDLNet_UDP_Open

Name	SDLNet_UDP_Open
Synopsis	require(‘SDL_net’) SDLNet_UDP_Open(port)
Description	Open a socket to be used for UDP packet sending and/or receiving
Return Value	UDP socket
Parameters	Port [in] : port number

Example:

```
port = 1234

-- open udp server socket
sock = SDLNet_UDP_Open(port)
if sock == nil then
	print("Couldn't Open UDP socket: ",SDLNet_GetError())
end
```

13.2.203. SDLNet_UDP_Close

Name	SDLNet_UDP_Close
Synopsis	require('SDL_net') SDLNet_UDP_Close(sock)
Description	Shutdown, close, and free a UDPsocket
Return Value	
Parameters	sock [in] : UDP socket

Example :

```
port = 1234

-- open udp server socket
sock = SDLNet_UDP_Open(port)
if sock == nil then
        print("Couldn't Open UDP socket: ",SDLNet_GetError())
end
-- close the socket
SDLNet_UDP_Close(sock)
```

13.2.204. SDLNet_UDP_Bind

Name	SDLNet_UDP_Bind
Synopsis	require('SDL_net') SDLNet_UDP_Bind(sock, channel, address)
Description	Bind an address to a channel on a socket.
Return Value	Channel, -1 on error
Parameters	sock [in] : UDP socket channel [in] : channel address [in] : IP address

Example :

```
ip = IPaddress_new()
…
sock = SDLNet_UDP_Open(port)
host=SDLNet_ResolveIP(ip)
…
if SDLNet_UDP_Bind(sock, 0, ip)==-1 then
        print("Couldn't Bind UDP socket: ",SDLNet_GetError())
else
        print("SDLNet_UDP_Bind: PASS")
end
```

13.2.205. SDLNet_UDP_Unbind

Name	SDLNet_UDP_Unbind
Synopsis	require('SDL_net') SDLNet_UDP_Unbind(sock, channel)
Description	This removes all previously assigned (bound) addresses from a socket channel
Return Value	
Parameters	sock [in] : UDP socket channel [in] : channel

Example :

```
ip = IPaddress_new()
…
sock = SDLNet_UDP_Open(port)
host=SDLNet_ResolveIP(ip)
…
if SDLNet_UDP_Bind(sock, 0, ip)==-1 then
        print("Couldn't Bind UDP socket: ",SDLNet_GetError())
else
        print("SDLNet_UDP_Bind: PASS")
end
…
SDLNet_UDP_Unbind(sock, 0)
```

13.2.206. SDLNet_UDP_GetPeerAddress

Name	SDLNet_UDP_GetPeerAddress
Synopsis	require('SDL_net') SDLNet_UDP_GetPeerAddress(sock, channel)
Description	Get the primary address assigned to this channel
Return Value	IP address
Parameters	sock [in] : UDP socket channel [in] : channel

Example :

```
port = 1234
sock = SDLNet_UDP_Open(port)

myip = SDLNet_UDP_GetPeerAddress(sock, -1)
if myip == nil then
	print("SDLNet_UDP_GetPeerAddress: ", SDLNet_GetError())
else
	print("SDLNet_UDP_GetPeerAddress: PASS")
end
```

13.2.207. SDLNet_UDP_Send

Name	SDLNet_UDP_Send
Synopsis	require('SDL_net') SDLNet_UDP_Send(sock, channel, packet)
Description	Send packet using the specified socket sock, use ing the specified channel or else the packet's address
Return Value	1 on success 0 on error
Parameters	sock [in] : UDP socket channel [in] : channel packet [in] : UDP packet

Example :

```
port = 1234
sock = SDLNet_UDP_Open(port)
…
input = SDLNet_AllocPacket(65535)
if input == nil then
        print("Couldn't Allocate packet: ",SDLNet_GetError())
end
…
if SDLNet_UDP_Send(sock, -1, input) == 0 then
        print("Could send UDP data:",SDLNet_GetError())
else
        print("SDLNet_UDP_Send : PASS")
end
```

13.2.208. SDLNet_UDP_Recv

Name	SDLNet_UDP_Recv
Synopsis	require('SDL_net') SDLNet_UDP_Recv(sock, packet)
Description	Receive a packet on the specified sock socket.
Return Value	Number of packets, -1 on error
Parameters	sock [in] : UDP socket packet [out] : UDP packet

Example :

```
port = 1234
sock = SDLNet_UDP_Open(port)
…
input = SDLNet_AllocPacket(65535)
if input == nil then
        print("Couldn't Allocate packet: ",SDLNet_GetError())
end
…
while SDLNet_UDP_Recv(sock, input) ==nil do
        SDL_Delay(100)          -- /*1/10th of a second */
end
```

13.2.209. SDLNet_UDP_SendV

Name	SDLNet_UDP_SendV
Synopsis	require('SDL_net') SDLNet_UDP_SendV(sock, packets, npackets)
Description	Send npackets of packetV using the specified sock socket
Return Value	Number of packets sent, -1 on error
Parameters	sock [in] : UDP socket packets [in] : UDP packets npackets [in] : number

Example :

```
port = 1234
local len = 1400
sock = SDLNet_UDP_Open(port)
…
packets = SDLNet_AllocPacketV(32,len)
if packets == nil then
        print("Couldn't Allocate PacketV ",SDLNet_GetError())
else
        print("SDLNet_AllocPacketV : PASS")
end
…
if SDLNet_UDP_SendV(sock, packets, 32) == 0 then
        print("Couldn't send PacketV:",SDLNet_GetError())
else
        print("SDLNet_UDP_SendV :PASS")
end
```

13.2.210. SDLNet_UDP_RecvV

Name	SDLNet_UDP_RecvV
Synopsis	require('SDL_net') SDLNet_UDP_RecvV(sock, packets)
Description	Receive into a packet vector on the specified socket sock
Return Value	Number of packets rx, -1 on error
Parameters	sock [in] : UDP socket packets [out] : UDP packets

Example :

```
port = 1234
local len = 1400
sock = SDLNet_UDP_Open(port)
…
local packetV = nil
ret ,packetV = SDLNet_UDP_RecvV(sock)
if ret == -1 then
        print("SDLNet_UDP_RecvV: ",SDLNet_GetError())
else
        print("SDLNet_UDP_RecvV: PASS")
end
```

13.2.211. SDLNet_AllocPacket

Name	SDLNet_AllocPacket
Synopsis	require(‘SDL_net’) SDLNet_AllocPacket(size)
Description	Create (via malloc) a new UDPpacket with a data buffer of size bytes
Return Value	UDP packet
Parameters	size [in] : bytes

Example :

```
out = SDLNet_AllocPacket(65535)
if out == nil then
	print("Couldn't Allocate packet: ",SDLNet_GetError())
else
	print("SDLNet_AllocPacket : PASS")
end
```

13.2.212. SDLNet_ResizePacket

Name	SDLNet_ResizePacket
Synopsis	require('SDL_net') SDLNet_ResizePacket(packet, newsize)
Description	Resize a UDPpackets data buffer to size bytes
Return Value	New allocated size
Parameters	packet [in] : UDP packet newsize [in] : size

Example :

```
input = SDLNet_AllocPacket(65535)
if input == nil then
	print("Couldn't Allocate packet: ",SDLNet_GetError())
end

newsize=SDLNet_ResizePacket(input, 2048)
if newsize<2048 then
	print("SDLNet_ResizePacket: ", SDLNet_GetError())
else
	print("SDLNet_ResizePacket: PASS")
end
```

13.2.213. SDLNet_FreePacket

Name	SDLNet_FreePacket
Synopsis	require('SDL_net') SDLNet_FreePacket(packet)
Description	Free a UDPpacket from memory
Return Value	
Parameters	packet [in] : UDP packet

Example :

```
out = SDLNet_AllocPacket(65535)
if out == nil then
        print("Couldn't Allocate packet: ",SDLNet_GetError())
else
        print("SDLNet_AllocPacket : PASS")
end
…
SDLNet_FreePacket(out)
```

13.2.214. SDLNet_AllocPacketV

Name	SDLNet_AllocPacketV
Synopsis	require('SDL_net') SDLNet_AllocPacketV(howmany, size)
Description	Create (via malloc) a vector of new UDPpackets, each with data buffers of size bytes
Return Value	UDP packet
Parameters	howmany [in] : number of packet size [in] : size of each

Example :

```
local len = 1400
packets = SDLNet_AllocPacketV(32,len)
if packets == nil then
	print("Couldn't Allocate PacketV ",SDLNet_GetError())
else
	print("SDLNet_AllocPacketV : PASS")
end
```

13.2.215. SDLNet_FreePacketV

Name	SDLNet_FreePacketV
Synopsis	require('SDL_net') SDLNet_FreePacketV(packetV)
Description	Free a UDPpacket vector from memory
Return Value	
Parameters	packetV [in] : UDP packet

Example :

```
local len = 1400
packets = SDLNet_AllocPacketV(32,len)
if packets == nil then
        print("Couldn't Allocate PacketV ",SDLNet_GetError())
else
        print("SDLNet_AllocPacketV : PASS")
end
…
SDLNet_FreePacketV(packets)
```

13.2.216. SDLNet_AllocSocketSet

Name	SDLNet_AllocSocketSet
Synopsis	require('SDL_net') SDLNet_AllocSocketSet(maxsockets)
Description	Create a socket set that will be able to watch up to maxsockets number of sockets
Return Value	SDLNet SocketSet
Parameters	maxsockets [in] : number of sockets

Example :

```
set=SDLNet_AllocSocketSet(16)
if set== nil then
    printf("SDLNet_AllocSocketSet: ", SDLNet_GetError())
end
```

13.2.217. SDLNet_FreeSocketSet

Name	SDLNet_FreeSocketSet
Synopsis	require('SDL_net') SDLNet_FreeSocketSet(set)
Description	Free the socket set from memory
Return Value	
Parameters	set [in] : SDLNet SocketSet

Example :

```
set=SDLNet_AllocSocketSet(16)
if set== nil then
    printf("SDLNet_AllocSocketSet: ", SDLNet_GetError())
end
…
SDLNet_FreeSocketSet(set)
```

13.2.218. SDLNet_AddSocket

Name	SDLNet_AddSocket
Synopsis	require('SDL_net') SDLNet_AddSocket(set, sock)
Description	Add a socket to a socket set that will be watched
Return Value	Number of sockets
Parameters	set [in] : SDLNet SocketSet sock [in] : Generic socket

Example :

```
port= 1234
sock = SDLNet_UDP_Open(port)
…
set=SDLNet_AllocSocketSet(16)
if set== nil then
	printf("SDLNet_AllocSocketSet: ", SDLNet_GetError())
end
…
numused=SDLNet_UDP_AddSocket(set,sock)
if numused==-1 then
	printf("SDLNet_AddSocket: ", SDLNet_GetError())
end
```

13.2.219. SDLNet_DelSocket

Name	SDLNet_DelSocket
Synopsis	require('SDL_net') SDLNet_DelSocket(set, sock)
Description	Remove a socket from a socket set
Return Value	Number of sockets
Parameters	set [in] : SDLNet SocketSet sock [in] : Generic socket

Example :

```
port= 1234
sock = SDLNet_UDP_Open(port)
…
set=SDLNet_AllocSocketSet(16)
if set== nil then
        printf("SDLNet_AllocSocketSet: ", SDLNet_GetError())
end
…
numused=SDLNet_UDP_AddSocket(set,sock)
if numused==-1 then
        printf("SDLNet_AddSocket: ", SDLNet_GetError())
end
…
if SDLNet_UDP_DelSocket(set, sock) == -1 then
        print("SDLNet_UDP_DelSocket :",SDLNet_GetError())
else
        print("SDLNet_UDP_DelSocket: PASS")
end
```

13.2.220. SDLNet_CheckSockets

Name	SDLNet_CheckSockets
Synopsis	require('SDL_net') SDLNet_CheckSockets(set, timeout)
Description	Check all sockets in the socket set for activity
Return Value	Number of sockets ready for reading, -1 on error
Parameters	set [in] : SDLNet SocketSet timeout [in] : time value

Example :

```
set=SDLNet_AllocSocketSet(16)
if set== nil then
        printf("SDLNet_AllocSocketSet: ", SDLNet_GetError())
end
…
numready=SDLNet_CheckSockets(set, 1000)
if numready==-1 then
        print("SDLNet_CheckSockets: ", SDLNet_GetError())
elseif numready ~= 0 then
        print("sockets with activity!\n",numready)
end
```

13.2.221. SDLNet_ SocketReady

Name	SDLNet_SocketReady
Synopsis	require('SDL_net') SDLNet_UDP_SocketReady(sock)
Description	Check all the active sockets .
Return Value	non-zero for activity. zero is returned for no activity
Parameters	sock [in] : SDL_net UDP socket

Example :

```
-- Check for, and handle UDP data
local numready, numpkts;

numready=SDLNet_CheckSockets(set, 0);
if(numready==-1)then
	print("SDLNet_CheckSockets: "..SDLNet_GetError());
	--most of the time this is a system error, where perror might help you.
else if(numready)then
	print("There are sockets with activity"..tostring(numready));
	--check all sockets with SDLNet_SocketReady and handle the active ones.
	if(SDLNet_SocketReady(udpsock))then
		numpkts=SDLNet_UDP_Recv(udpsock,packet);
		if(numpkts)then
			-- process the packet.
		end
	end
end
```

13.2.222. SDLNet_ GetError

Name	SDLNet_ GetError
Synopsis	require('SDL_net') SDLNet_GetError()
Description	This is the same as SDL_GetError, which returns the last error set as a string which is use to tell the user what happened when an error status has been returned from an SDLNet_function. It returns a char instace(string) containing a humam readble version or the reason for the last error that occured.
Return Value	a string containing a humam readble version or the reason for the last error that occured.
Parameters	none

Example :

```
print("Oh My Goodness, an error : "..SDLNet_GetError());
```

SDL ttf

13.2.223. TTF_Init

Name	TTF_Init
Synopsis	require(‘SDL_ttf’) TTF_Init()
Description	Initialize
Return Value	0 on success -1 on error

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init: ", TTF_GetError())
end
```

13.2.224. TTF_WasInit

Name	TTF_WasInit
Synopsis	require(‘SDL_ttf’) TTF_WasInit()
Description	Query for Init
Return Value	Non zero if already initialized

Example :

```
TTF_Init()
if TTF_WasInit()== 0 then
        print("\nTTF_Init: ", TTF_GetError())
end
```

13.2.225. TTF_Quit

Name	TTF_Quit
Synopsis	require(‘SDL_ttf’) TTF_Quit()
Description	Shutdown and cleanup the truetype font API
Return Value	

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init: ", TTF_GetError())
end
TTF_Quit()
```

13.2.226. TTF_OpenFont

Name	TTF_OpenFont
Synopsis	require('SDL_ttf') TTF_OpenFont(file, ptsize)
Description	Load font
Return Value	TTF font
Parameters	file [in] : font file ptsize [in] : point size

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16);
if font == nil then
        print("\nTTF_OpenFont: Fail!!, ", TTF_GetError())
        return 2
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.227. TTF_OpenFontRW

Name	TTF_OpenFontRW
Synopsis	require('SDL_ttf') TTF_OpenFontRW(src, freesrc, ptsize)
Description	font is loaded from location
Return Value	TTF font
Parameters	src [in] : location of file freesrc [in] : non-zero value mean is will automatically close/free the *src*. ptsize [in] : point size

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

if TTF_OpenFontRW(SDL_RWFromFile(<path>.."verdana.ttf","r"), 1, 16)==nil then
        print("\nTTF_OpenFontRW: Fail!!, ", TTF_GetError())
        return 2
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.228. TTF_OpenFontIndex

Name	TTF_OpenFontIndex
Synopsis	require('SDL_ttf') TTF_OpenFontIndex(file, ptsize, index)
Description	Indexing of font
Return Value	TTF font
Parameters	file [in] : font file ptsize [in] : point size index [in] : font face

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end
if TTF_OpenFontIndex(chResPath.."verdana.ttf", 16, 0)==nil then
	print("\nTTF_OpenFontIndex: Fail!!, ", TTF_GetError())
	return 2
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.229. TTF_OpenFontIndexRW

Name	TTF_OpenFontIndexRW
Synopsis	require('SDL_ttf') TTF_OpenFontIndexRW(src, freesrc, ptsize, index)
Description	Indexing of font from location
Return Value	TTF font
Parameters	src [in] : location of file freesrc [in] : non-zero value mean is will automatically close/free the *src*. ptsize [in] : point size index [in] : font face

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end

if TTF_OpenFontRW(SDL_RWFromFile(chResPath.."verdana.ttf","r"), 1, 16, 0)==nil then
	print("\nTTF_OpenFontIndexRW: Fail!!, ", TTF_GetError())
	return 2
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.230. TTF_CloseFont

Name	TTF_CloseFont
Synopsis	require('SDL_ttf') TTF_CloseFont(font)
Description	Close font
Return Value	
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

if TTF_OpenFont(chResPath.."verdana.ttf", 16)== nil then
        print("\nTTF_OpenFont: Fail!!, ", TTF_GetError())
        return 2
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.231. TTF_ByteSwappedUNICODE

Name	TTF_ByteSwappedUNICODE
Synopsis	require('SDL_ttf') TTF_ByteSwappedUNICODE(swapped)
Description	Swapped byte
Return Value	
Parameters	swapped [in]

13.2.232. TTF_GetFontStyle

Name	TTF_GetFontStyle
Synopsis	require('SDL_ttf') TTF_GetFontStyle(font)
Description	Get the rendering style of the loaded font
Return Value	Font style
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(chResPath.."verdana.ttf", 16)
if font == nil
        print("\nTTF_OpenFont: Fail!!, ", TTF_GetError())
        return 2
end

local style=TTF_GetFontStyle(font)
print("\nThe font style is:")
if style==TTF_STYLE_NORMAL then
        print("\nNORMAL")
elseif style==TTF_STYLE_BOLD then
        print("\nBOLD")
elseif style==TTF_STYLE_ITALIC then
        print("\nITALIC")
```

```
elseif style==TTF_STYLE_UNDERLINE then
        print("\nUNDERLINE")
elseif style==TTF_STYLE_STRIKETHROUGH then
        print("\nSTRIKETHROUGH")
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.233. TTF_SetFontStyle

Name	TTF_SetFontStyle
Synopsis	require('SDL_ttf') TTF_SetFontStyle(font, style)
Description	Set the rendering style of the loaded font
Return Value	
Parameters	font [in] : TTF font style [in] : font style

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end
local type= {TTF_STYLE_NORMAL, TTF_STYLE_BOLD, TTF_STYLE_ITALIC, TTF_STYLE_UNDERLINE}

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
for i=1,4 do
	TTF_SetFontStyle(font, type[i])
	local style=TTF_GetFontStyle(font)
	print("\nThe font style is:")
	if style==TTF_STYLE_NORMAL then
		print("\nNORMAL")
	elseif style==TTF_STYLE_BOLD then
		print("\nBOLD")
	elseif style==TTF_STYLE_ITALIC then
		print("\nITALIC")
	elseif style==TTF_STYLE_UNDERLINE then
```

```
                print("\nUNDERLINE")
        elseif style==TTF_STYLE_STRIKETHROUGH then
                print("\nSTRIKETHROUGH")
        end
end

TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.234. TTF_FontHeight

Name	TTF_FontHeight
Synopsis	require(‘SDL_ttf’) TTF_FontHeight(font)
Description	Get the maximum pixel height of all glyphs of the loaded font
Return Value	Font height
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
                print("\nTTF_Init Error: ", TTF_GetError())
                return 2
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
print("\nThe font max height is: ", TTF_FontHeight(font))
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.235. TTF_FontAscent

Name	TTF_FontAscent
Synopsis	require(‘SDL_ttf’) TTF_FontAscent(font)
Description	Get the maximum pixel ascent of all glyphs of the loaded font
Return Value	Offset number
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
print("\nThe font ascent is: ", TTF_FontAscent(font))
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.236. TTF_FontDescent

Name	TTF_FontDescent
Synopsis	require(‘SDL_ttf’) TTF_FontDescent(font)
Description	Get the maximum pixel descent of all glyphs of the loaded font
Return Value	Offset number
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
print("\nThe font descent is: ", TTF_FontDescent(font))
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.237. TTF_FontLineSkip

Name	TTF_FontLineSkip
Synopsis	require('SDL_ttf') TTF_FontLineSkip(font)
Description	Get the recommended pixel height of a rendered line of text of the loaded font
Return Value	Spacing between lines of text
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
print("\nThe font line skip is: ", TTF_FontLineSkip(font));
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.238. TTF_FontFaces

Name	TTF_FontFaces
Synopsis	require(‘SDL_ttf’) TTF_FontFaces(font)
Description	Get the number of faces ("sub-fonts") available in the loaded font
Return Value	Number of faces
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont((<path>.."verdana.ttf", 16)
print("\nThe number of faces in the font is: ", TTF_FontFaces(font));
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.239. TTF_FontFaceIsFixedWidth

Name	TTF_FontFaceIsFixedWidth
Synopsis	require(‘SDL_ttf’) TTF_FontFaceIsFixedWidth(font)
Description	Test if the current font face of the loaded font is a fixed width font
Return Value	Number value
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end
local font=TTF_OpenFont((<path>.."verdana.ttf", 16)
if TTF_FontFaceIsFixedWidth(font) then
        print("\nThe font is fixed width.")
else
        print("\nThe font is not fixed width.\n")
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.240. TTF_FontFaceFamilyName

Name	TTF_FontFaceFamilyName
Synopsis	require(‘SDL_ttf’) TTF_FontFaceFamilyName(font)
Description	Get the current font face family name from the loaded font
Return Value	String
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local familyname=TTF_FontFaceFamilyName(font)
if(familyname) then
        print("\nThe family name of the face in the font is: ", familyname)
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.241. TTF_FontFaceStyleName

Name	TTF_FontFaceStyleName
Synopsis	require('SDL_ttf') TTF_FontFaceStyleName(font)
Description	Get the current font face style name from the loaded font
Return Value	String
Parameters	font [in] : TTF font

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local stylename=TTF_FontFaceStyleName(font)
if(stylename) then
        print("\nThe name of the face in the font is: ", stylename)
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.242. TTF_SizeText

Name	TTF_SizeText
Synopsis	require('SDL_ttf') TTF_SizeText(font, text, w, h)
Description	Calculate the resulting surface size of the LATIN1 encoded text rendered using font
Return Value	Size
Parameters	font [in] : TTF font text [in] : text string w [in] : width h [in] : height

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end
local w = new_intp()
local h = new_intp()

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
if TTF_SizeText(font,"Hello World!",w,h) == -1 then
        print("\nTTF_SizeText Error:",TTF_GetError())
        return 2
else
        print("\nwidth=",intp_value(w)," height=",intp_value(h))
end

TTF_CloseFont(font)
font=nil
```

delete_intp(w)
delete_intp(h)
TTF_Quit()

13.2.243. TTF_SizeUTF8

Name	TTF_SizeUTF8
Synopsis	require('SDL_ttf') TTF_SizeUTF8(font, text, w, h)
Description	Calculate the resulting surface size of the UTF8 encoded text rendered using font
Return Value	Size
Parameters	font [in] : TTF font text [in] : text string w [in] : width h [in] : height

Example :

```
if TTF_Init()==-1 then
                print("\nTTF_Init Error: ", TTF_GetError())
                return 2
end

local w = new_intp()
local h = new_intp()

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
if TTF_SizeUTF8(font,"Hello World!",w,h) == -1 then
        print("\nTTF_SizeText Error:",TTF_GetError())
else
        print("\nwidth=",intp_value(w)," height=",intp_value(h))
end
TTF_CloseFont(font)
font=nil
delete_intp(w)
```

delete_intp(h)
TTF_Quit()

13.2.244. TTF_SizeUNICODE

Name	TTF_SizeUNICODE
Synopsis	require('SDL_ttf') TTF_SizeUNICODE(font, text, w, h)
Description	Calculate the resulting surface size of the UNICODE encoded text rendered using font
Return Value	Size
Parameters	font [in] : TTF font text [in] : text string w [in] : width h [in] : height

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end

local w = new_intp()
local h = new_intp()
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
if font == nil then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end
local str = "Hello World!"
local voidp_str = charp_to_voidp(str)
local ttxt = voidp_to_Uint16p(voidp_str)

if(TTF_SizeUNICODE(font,ttxt,w,h)) == -1 then
```

```
        print("\nTTF_SizeText Error:", TTF_GetError())
else
        print("\nwidth=",intp_value(w)," height=",intp_value(h))
end
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.245. TTF_RenderText_Solid

Name	TTF_RenderText_Solid
Synopsis	require('SDL_ttf') TTF_RenderText_Solid(font, text, fg)
Description	Render the LATIN1 encoded text using font with fg color onto a new surface, using the Solid mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
color.r = 150
color.g = 150
color.b = 150
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
text_surface=TTF_RenderText_Solid(font,"TTF_RenderText_Solid_Test TEST!!",color)
if text_surface==nil then
	print("\nTTF_Error:",TTF_GetError())
else
	SDL_BlitSurface(text_surface,nil,screen,nil)
	SDL_Flip(screen)
	SDL_Delay(2000)
```

```
end
SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.246. TTF_RenderUTF8_Solid

Name	TTF_RenderUTF8_Solid
Synopsis	require('SDL_ttf') TTF_RenderUTF8_Solid(font, text, fg)
Description	Render the UTF8 encoded text using font with fg color onto a new surface, using the Solid mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
local text_surface = TTF_RenderUTF8_Solid(font,"TTF_RenderUTF8_Solid_Test TEST!!",color)
if text_surface==nil then
        print("\nTTF_Error:",TTF_GetError())
else
        SDL_BlitSurface(text_surface,nil,screen,nil)
        SDL_Flip(screen)
        SDL_Delay(2000)
```

```
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.247. TTF_RenderUNICODE_Solid

Name	TTF_RenderUNICODE_Solid
Synopsis	require('SDL_ttf') TTF_RenderUNICODE_Solid(font, text, fg)
Description	Render the UNICODE encoded text using font with fg color onto a new surface, using the Solid mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init: ", TTF_GetError())
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
local str = "Hello World!"
local voidp_str = charp_to_voidp(str)
local ttxt = voidp_to_Uint16p(voidp_str)

text_surface=TTF_RenderUNICODE_Solid(font,ttxt,color)
if text_surface==nil then
	print("\nTTF_Init error: ", TTF_GetError())
else
	SDL_BlitSurface(text_surface,nil,screen,nil)
```

```
        SDL_Flip(screen)
        SDL_Delay(2000)
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.248. TTF_RenderGlyph_Solid

Name	TTF_RenderGlyph_Solid
Synopsis	require('SDL_ttf') TTF_RenderGlyph_Solid(font, ch, fg)
Description	Render the glyph for the UNICODE ch using font with fg color onto a new surface, using the Solid mode
Return Value	SDL surface
Parameters	font [in] : TTF font ch [in] : glyph fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init: ", TTF_GetError())
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)

for ch=100,128 do
	text_surface = TTF_RenderGlyph_Solid(font,ch,color);
	if text_surface==nil then
		print("\nTTF_Init error: ", TTF_GetError())
	else
		SDL_BlitSurface(text_surface,nil,screen,nil)
		SDL_Flip(screen)
```

```
                SDL_Delay(2000)
        end
        SDL_FillRect(screen, nil, SDL_MapRGB(screen.format,0,0,0))
        SDL_Flip(screen)
        SDL_Delay(2000)
end
SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.249. TTF_RenderText_Shaded

Name	TTF_RenderText_Shaded
Synopsis	require('SDL_ttf') TTF_RenderText_Shaded(font, text, fg, bg)
Description	Render the LATIN1 encoded text using font with fg color onto a new surface filled with the bg color, using the Shaded mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color bg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
local bgcolor = SDL_Color_new()
color.r = 0
color.g = 0
color.b = 0
bgcolor.r = 255
bgcolor.g = 255
bgcolor.b = 255
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
text_surface=TTF_RenderText_Shaded(font,"TTF_RenderText_Shaded_Test
        TEST!!",color,bgcolor)
```

```
if(text_surface == nil) then
        print("\nTTF_Error:",TTF_GetError())
else
        SDL_BlitSurface(text_surface,nil,screen,nil);
        SDL_Flip(screen)
        SDL_Delay(2000)
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.250. TTF_RenderUTF8_Shaded

Name	TTF_RenderUTF8_Shaded
Synopsis	require('SDL_ttf') TTF_RenderUTF8_Shaded(font, text, fg, bg)
Description	Render the UTF8 encoded text using font with fg color onto a new surface filled with the bg color, using the Shaded mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color bg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
local bgcolor = SDL_Color_new()
color.r = 0
color.g = 0
color.b = 0
bgcolor.r = 255
bgcolor.g = 255
bgcolor.b = 255
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
text_surface=TTF_RenderUTF8_Shaded(font,"TTF_RenderUTF8_Shaded_Test
        TEST!!",color,bgcolor)
if(text_surface == nil) then
```

```
        print("\nTTF_Error:",TTF_GetError())
else
        SDL_BlitSurface(text_surface,nil,screen,nil);
        SDL_Flip(screen)
        SDL_Delay(2000)
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.251. TTF_RenderUNICODE_Shaded

Name	TTF_RenderUNICODE_Shaded
Synopsis	require('SDL_ttf') TTF_RenderUNICODE_Shaded(font, text, fg, bg)
Description	Render the UNICODE encoded text using font with fg color onto a new surface filled with the bg color, using the Shaded mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color bg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init: ", TTF_GetError())
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
local bgcolor = SDL_Color_new()
color.r = 0
color.g = 0
color.b = 0
bgcolor.r = 255
bgcolor.g = 255
bgcolor.b = 255

local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
local str = "Hello World!"
local voidp_str = charp_to_voidp(str)
local ttxt = voidp_to_Uint16p(voidp_str)
```

```
text_surface=TTF_RenderUNICODE_Shaded(font,ttxt,color,bgcolor)
if text_surface==nil then
	print("\TTF_RenderUNICODE_Shaded Error: ", TTF_GetError())
else
	SDL_BlitSurface(text_surface,nil,screen,nil)
	SDL_Flip(screen)
	SDL_Delay(2000)
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.252. TTF_RenderGlyph_Shaded

Name	TTF_RenderGlyph_Shaded
Synopsis	require('SDL_ttf') TTF_RenderGlyph_Shaded(font, ch, fg, bg)
Description	Render the glyph for the UNICODE ch using font with fg color onto a new surface filled with the bg color, using the Shaded mode
Return Value	SDL surface
Parameters	font [in] : TTF font ch [in] : glyph fg [in] : SDL color bg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init: ", TTF_GetError())
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
local bgcolor = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
bgcolor.r = 100
bgcolor.g = 100
bgcolor.b = 100
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)

for ch=100,128 do
        text_surface=TTF_RenderGlyph_Shaded(font,ch,color,bgcolor)
```

```
        if text_surface==nil then
                print("\nTTF_Init error: ", TTF_GetError())
        else
                SDL_BlitSurface(text_surface,nil,screen,nil)
                SDL_Flip(screen)
                SDL_Delay(2000)
                print("Test Pass!!")
        end
        SDL_FillRect(screen, nil, SDL_MapRGB(screen.format,0,0,0))
        SDL_Flip(screen)
        SDL_Delay(2000)
end
SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.253. TTF_RenderText_Blended

Name	TTF_RenderText_Blended
Synopsis	require('SDL_ttf') TTF_RenderText_Blended(font, text, fg)
Description	Render the LATIN1 encoded text using font with fg color onto a new surface, using the Blended mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init Error: ", TTF_GetError())
	return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
text_surface=TTF_RenderText_Blended(font,"TTF_RenderText_Blended_Test!!!",color)
if text_surface==nil then
	print("\nTTF_Error:",TTF_GetError())
else
	SDL_BlitSurface(text_surface,nil,screen,nil);
	SDL_Flip(screen)
	SDL_Delay(2000)
```

```
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.254. TTF_RenderUTF8_Blended

Name	TTF_RenderUTF8_Blended
Synopsis	require('SDL_ttf') TTF_RenderUTF8_Blended(font, text, fg)
Description	Render the UTF8 encoded text using font with fg color onto a new surface, using the Blended mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init Error: ", TTF_GetError())
        return 2
end

local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
text_surface=TTF_RenderUTF8_Blended(font,"TTF_RenderUTF8_Blended_Test!!!",color)
if text_surface==nil then
        print("\nTTF_Error:",TTF_GetError())
else
        SDL_BlitSurface(text_surface,nil,screen,nil);
        SDL_Flip(screen)
        SDL_Delay(2000)
```

```
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.255. TTF_RenderUNICODE_Blended

Name	TTF_RenderUNICODE_Blended
Synopsis	require('SDL_ttf') TTF_RenderUNICODE_Blended(font, text, fg)
Description	Render the UNICODE encoded text using font with fg color onto a new surface, using the Blended mode
Return Value	SDL surface
Parameters	font [in] : TTF font text [in] : text fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
        print("\nTTF_Init: ", TTF_GetError())
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
local bgcolor = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200

local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)
local str = "Hello World!"
local voidp_str = charp_to_voidp(str)
local ttxt = voidp_to_Uint16p(voidp_str)

text_surface=TTF_RenderUNICODE_Blended(font,ttxt,color)
if text_surface==nil then
        print("\TTF_RenderUNICODE_Blended Error: ", TTF_GetError())
```

```
else
        SDL_BlitSurface(text_surface,nil,screen,nil)
        SDL_Flip(screen)
        SDL_Delay(2000)
        print("Test Pass!!")
end

SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.256. TTF_RenderGlyph_Blended

Name	TTF_RenderGlyph_Blended
Synopsis	require('SDL_ttf') TTF_RenderGlyph_Blended(font, ch, fg)
Description	Render the glyph for the UNICODE ch using font with fg color onto a new surface, using the Blended mode
Return Value	SDL surface
Parameters	font [in] : TTF font ch [in] : glyph fg [in] : SDL color

Example :

```
if TTF_Init()==-1 then
	print("\nTTF_Init: ", TTF_GetError())
end
local font=TTF_OpenFont(<path>.."verdana.ttf", 16)
local color = SDL_Color_new()
local bgcolor = SDL_Color_new()
color.r = 200
color.g = 200
color.b = 200
local screen = SDL_SetVideoMode(320,200,8,SDL_ANYFORMAT)

for ch=100,128 do
	text_surface = TTF_RenderGlyph_Blended(font,ch,color)
	if text_surface==nil then
		print("\nTTF_Init error: ", TTF_GetError())
		return 2
	else
		SDL_BlitSurface(text_surface,nil,screen,nil)
```

```
                SDL_Flip(screen)
                SDL_Delay(2000)
        end
        SDL_FillRect(screen, nil, SDL_MapRGB(screen.format,0,0,0))
        SDL_Flip(screen)
end
SDL_FreeSurface(text_surface)
SDL_Color_delete(color)
TTF_CloseFont(font)
font=nil
TTF_Quit()
```

13.2.257. TTF_GlyphMetrics

Name	TTF_ GlyphMetrics
Synopsis	require('SDL_ttf') TTF_GlyphMetrics(font, ch, minx, maxx, miny, maxy, advance)
Description	Get desired glyph metrics of the UNICODE chargiven in ch from the loaded font. NOTE: Passing a nil font into this function will cause a segfault.
Return Value	Returns 0 on success, with all non-nil parameters set to the glyph metric as appropriate. -1 on errors, such as when the glyph does not exist in the font.
Parameters	font [in] : TTF font The loaded font from which to get the glyph metrics of ch. ch [in] : The UNICODE char to get the glyph metrics minx [in] : Instance to int to store the returned minimum X offset into, or nil when no return value desired. maxxy [in] : Instance to int to store the returned maximum X offset into, or nil when no return value desired. miny [in] : Instance to int to store the returned minimum Y offset into, or nilwhen no return value desired. maxy [in] : Instance to int to store the returned maximum Y offset into, or nil when no return value desired. advance [in] : Instance to int to store the returned advance offset into, or nil when no return value desired.

Example :

```
--Get the glyph metric for the letter 'g' in a loaded font
--TTF_Font *font;
local minx,maxx,miny,maxy,advance;
if(TTF_GlyphMetrics(font,'g',minx,maxx,miny,maxy,advance)==-1)then
	print(TTF_GetError());
else
	print("minx	: ".. tostring(minx));
	print("maxx	: ".. tostring(maxx));
```

```
    print("miny       : ".. tostring(miny));
    print("maxy       : ".. tostring(maxy));
    print("advance : ".. tostring(advance));
end
```

13.2.258. TTF_ GetError

Name	TTF_ GetError
Synopsis	require('SDL_ttf') TTF_ GetError ()
Description	This is a defined macro for SDL_GetError. It returns the last error set by TTF_SetError (or SDL_SetError) as a string. Used to tell the user what happened when an error status has been returned from an SDL_ttf function call.
Return Value	a string containing a human readable version or the reason for the last error that occured.
Parameters	none

Example :

```
function myfunc()
{
	print("Oh My Goodness, an error : ".. TTF_GetError());
}
```

13.2.259. TTF_ SetError

Name	TTF_ SetError
Synopsis	require(‘SDL_ttf’) TTF_ SetError (fmt)
Description	This is a defined macro for SDL_SetError, which sets the error string which may be fetched with TTF_GetError (or SDL_GetError). This functions acts like printf, except that it is limited to SDL_ERRBUFIZE(1024) chars in length.
Return Value	none
Parameters	fmt [in] : a string

Example :

```
function myfunc(i)
{
    TTF_SetError("myfunc is not implemented! was passed in.".. tostring(i));
}
```

13.2.260. SDL_putenv

Name	SDL_putenv
Synopsis	require('SDL_util') SDL_putenv(variable)
Description	The utility function sets the environment variable passed as parameter.
Return Value	None
Parameters	variable [in] : string

13.2.261. SDL_getenv

Name	SDL_getenv
Synopsis	require('SDL_util') SDL_getenv(name)
Description	The utility function retrieves the environment variable.
Return Value	Environment variable value
Parameters	name [in] : string

13.2.262. bit_or

Name	bit_or
Synopsis	require('SDL_util') bit_or(a, b)
Description	The utility function performs the bitwise OR operation on the operands.
Return Value	Bitwise OR value
Parameters	a [in] : number b [in] : number

13.2.263. bit_and

Name	bit_and
Synopsis	require('SDL_util') bit_and(a, b)
Description	The utility function performs the bitwise AND operation on the operands.
Return Value	Bitwise AND value
Parameters	a [in] : number b [in] : number

13.2.264. bit_not

Name	bit_not
Synopsis	require('SDL_util') bit_not(a, b)
Description	The utility function performs the bitwise NOT operation on the operands.
Return Value	Bitwise NOT value
Parameters	a [in] : number b [in] : number

13.2.265. bit_xor

Name	bit_xor
Synopsis	require('SDL_util') bit_xor(a, b)
Description	The utility function performs the bitwise XOR operation on the operands.
Return Value	Bitwise XOR value
Parameters	a [in] : number b [in] : number

13.2.266. bit_lshift

Name	bit_lshift
Synopsis	require('SDL_util') bit_lshift(a, shift)
Description	The utility function does the bitwise left shift operation on the operand.
Return Value	Bitwise left shift value
Parameters	a [in] : number to be left shift shift [in] : the number of bits to be shifted on left

13.2.267. bit_rshift

Name	bit_rshift
Synopsis	require('SDL_util') bit_rshift(a, shift)
Description	The utility function does the bitwise right shift operation on the operand.
Return Value	Bitwise right shift value
Parameters	a [in] : number to be right shift shift [in] : the number of bits to be shifted on right

13.2.268. new_int

Name	new_int
Synopsis	require('SDL_util') new_int(nelements)
Description	The utility function does the memory allocation
Return Value	Allocated memory location
Parameters	nelements[in] : number of integer elements to be allocated

13.2.269. delete_int

Name	delete_int
Synopsis	require('SDL_util') delete_int(ary)
Description	The utility function deletes the memory allocated for the ary element.
Return Value	None
Parameters	Ary [in] : Array which needs to be deallocated

13.2.270. int_getitem

Name	int_getitem
Synopsis	require('SDL_util') int_getitem(ary, index)
Description	The utility function retrieves the array element at index location.
Return Value	Array element value at the index position
Parameters	ary [in] : Array whose element needs to be accessed index [in] : location in array

13.2.271. int_setitem

Name	int_setitem
Synopsis	require('SDL_util') int_setitem(ary, index, value)
Description	The utility function sets the array element at index location.
Return Value	None
Parameters	ary [in] : Array whose element needs to be set index [in] : Location in array value [in] : Value which needs to be set for the array element

13.2.272. new_Uint8

Name	new_Uint8
Synopsis	require('SDL_util') new_Uint8(nelements)
Description	The utility function does the memory allocation for an array of Uint8 elements.
Return Value	Allocated memory location
Parameters	nelements[in] : number of elements to be allocated to store Uint8 elements

13.2.273. delete_Uint8

Name	delete_Uint8
Synopsis	require('SDL_util') delete_Uint8(ary)
Description	The utility function deletes the memory allocated for the Uint8 ary element.
Return Value	None
Parameters	ary [in] : Uint8 array which needs to be deallocated

13.2.274. Uint8_getitem

Name	Uint8_getitem
Synopsis	require('SDL_util') Uint8_getitem(ary, index)
Description	The utility function retrieves the Uint8 array element at index location.
Return Value	Array element value at the index position
Parameters	ary [in] : Uint8 Array whose element needs to be accessed index [in] : location in array

13.2.275. Uint8_setitem

Name	Uint8_setitem
Synopsis	require(‘ SDL_util’) Uint8_setitem(ary, index, value);
Description	Set the ‘value’ at index in given array. The array element size is one byte.
Return Value	none
Parameters	ary [in] : Array. index [in] : Target index . Index range start from 0. value [in] : Target value

Example :

```
local bitmap_d = {66, 77, 86, 2, 0, 0, 0, 0}
local bitmap = new_Uint8(8)
for i = 0, 7 do
        Uint8_setitem(bitmap, i, bitmap_d[i+1])
end
```

13.2.276. new_Uint16

Name	new_Uint16
Synopsis	require(‘ SDL_util’) new_Uint16 (nelements)
Description	Create an array of size ‘nelements’. The array element size is 2 byte.
Return Value	New created array.
Parameters	nelements [in] : Number of elements

Example :

```
local ramp = new_Uint16(255)
```

13.2.277. delete_Uint16

Name	delete_Uint16
Synopsis	require(‘ SDL_util’) delete_Uint16(ary)
Description	Delete and free memory for given array.
Return Value	none
Parameters	ary [in] : Array.

Example :

delete_Uint16(ramp)

13.2.278. Uint16_getitem

Name	Uint16_getitem
Synopsis	require(‘ SDL_util’) Uint16_getitem(ary, index)
Description	Return the element at given index in Uint16 array.
Return Value	Array element value at the index position for Uint16 array.
Parameters	ary [in] : Array. index [in] : Target index . Index range start from 0.

Example :

```
for i = 1, 255 do
        print(i," ",Uint16_getitem(ramp,i),"\n")
end
```

13.2.279. Uint16_setitem

Name	Uint16_setitem
Synopsis	require(‘ SDL_util’) Uint16_setitem(ary, index, value)
Description	Set the ‘value’ at index in given array. The array element is 2 byte long.
Return Value	none
Parameters	ary [in] : Array. index [in] : Target index . Index range start from 0. value [in] : Target value

Example :

```
for i=1, 255 do
        if not(temp < 0) then
                Uint16_setitem(ramp,i,temp)
        end
end
```

13.2.280. new_Uint32

Name	new_Uint32
Synopsis	require(‘ SDL_util’) new_Uint32(nelements)
Description	Create an array of size ‘nelements’. The array element size is 4 byte.
Return Value	New created array.
Parameters	nelements [in] : Number of elements

Example :

```
local ramp = new_Uint32(255)
```

13.2.281. delete_Uint32

Name	delete_Uint32
Synopsis	require(‘ SDL_util’) delete_Uint32(ary)
Description	Delete and free memory for given array.
Return Value	none
Parameters	ary [in] : Array.

Example :

delete_Uint32(ramp)

13.2.282. Uint32_getitem

Name	Uint32_getitem
Synopsis	require(‘ SDL_util’) Uint32_getitem(ary, index)
Description	Return the element at given index in array.
Return Value	Array element value at the index position for Uint32 array.
Parameters	ary [in] : Array. index [in] : Target index . Index range start from 0.

Example :

```
for i = 1, 255 do
        print(i," ",Uint32_getitem(ramp,i),"\n")
end
```

13.2.283. Uint32_setitem

Name	Uint32_setitem
Synopsis	require(‘ SDL_util’) Uint32_setitem(ary, index, value)
Description	Set the ‘value’ at index in given array. The array element is 2 byte long.
Return Value	none
Parameters	ary [in] : Array. index [in] : Target index . Index range start from 0. value [in] : Target value

Example :

```
for i=1, 255 do
        if not(temp < 0) then
                Uint32_setitem(ramp,i,temp)
        end
end
```

13.2.284. new_bool

Name	new_bool
Synopsis	require(‘ SDL_util’) new_bool(nelements)
Description	Create an array of size ‘nelements’. The array element is boolean.
Return Value	New created array.
Parameters	nelements [in] : Number of elements

Example :

```
local boolanArray = new_bool(255)
```

13.2.285. delete_bool

Name	delete_bool
Synopsis	require(‘ SDL_util’) delete_bool(ary)
Description	Delete and free memory for given array.
Return Value	none
Parameters	ary [in] : Array.

Example :

delete_Uint32(boolanArray)

13.2.286. bool_getitem

Name	bool_getitem
Synopsis	require('SDL_util') bool_getitem(ary, index)
Description	Getting the element from the array as per its corresponding index value.
Return Value	Element as per the index value (Note : Index should start from 1)
Parameters	ary [in] : Array(table) index[in]: index number

Example :

```
function GetItem(ary)
  ItemTable = {0,0,0,0,0}
     for i=0, 5 do
          ItemTable[i+1] = bool_getitem(ary, i)
     end
end
```

13.2.287. bool_setitem

Name	bool_setitem
Synopsis	require('SDL_util') bool_setitem(ary, index, value)
Description	Setting the element of array as per corresponding index and value in the array
Return Value	None
Parameters	ary [in] : array(table) index[in]: index number value[in]: value

Example :

```
function SetItem(Itemary, index, 10)
        bool_setitem(Itemary, index, 10)
end
```

13.2.288. new_Uint8p

Name	new_Uint8p
Synopsis	require('SDL_util') new_Uint8p()
Description	Initialize the unsigned integer reference. Allocate contiguous memory (block of - sizeof(Uint8)).
Return Value	Uint8p allocated memory reference
Parameters	none

Example :

```
function Init_U8p()
    InitU8p = new_Uint8p()
end
```

13.2.289. delete_Uint8p

Name	delete_Uint8p
Synopsis	require('SDL_util') delete_Uint8p(self)
Description	Delete(Free) the allocated memory
Return Value	none
Parameters	self [in] : own Instance

Example :

```
function Delete_U8p(self)
    delete_Uint8p(self)
end
```

13.2.290. Uint8p_assign

Name	Uint8p_assign
Synopsis	require('SDL_util') Uint8p_assign(self, value)
Description	Assign the internation reference with corresponding value.
Return Value	self = value
Parameters	self [in] : own Instance value[in]: value (a number)

Example :

```
function U8p_Assign(self, 11)
    Uint8p_assign(self, 11)
End
```

13.2.291. Uint8p_value

Name	Uint8p_value
Synopsis	require('SDL_util') Uint8p_value(self)
Description	Retrieve the reference / instance
Return Value	self
Parameters	self [in] : own Instance

Example :

```
function U8p_Instance(self)
    Uint8p_value(self)
end
```

13.2.292. new_intp

Name	new_intp
Synopsis	require('SDL_util') new_intp()
Description	Initialize the unsigned integer reference. Allocate contiguous memory.
Return Value	self – after successful memory allocation
Parameters	none

Example :

```
function Init_INTp()
    pInt = new_intp()
end
```

13.2.293. delete_intp

Name	delete_intp
Synopsis	require('SDL_util') delete_intp(self)
Description	Delete(Free) the allocated memory.
Return Value	Free(self) –return nil
Parameters	self [in] : own instance

Example :

```
function Delete_Intp(self)
    delete_intp(self)
end
```

13.2.294. intp_assign

Name	intp_assign
Synopsis	require('SDL_util') intp_assign(ptr,value)
Description	Assigns the value passed as a second argument to the address pointed by the the first argument(ptr)
Return Value	None
Parameters	Ptr [in] : Address Value[in] : Value to be assigned

Example:

```
local intp = new_intp()
local value = 10
local ret = intp_assign(intp, value)
```

13.2.295. intp_value

Name	intp_value
Synopsis	require('SDL_util') intp_value(ptr)
Description	This function gets the value at specified address
Return Value	Returns a integer value
Parameters	ptr[in] : memory address

Example:

```
local intp = new_intp()
local value = 10
intp_assign(intp, value)
ret = intp_value(intp)
```

13.2.296. new_Uint16p

Name	new_Uint16p
Synopsis	require('SDL_util') new_Uint16p()
Description	This function allocates 16 bit of memory
Return Value	Allocated memory
Parameters	none

Example:

```
local intp = new_Uint16p()
local value = 65535
intp =    copy_Uint16p(value)
ret = Uint16p_value(intp)
```

13.2.297. Uint16p_value

Name	Uint16p_value(ptr)
Synopsis	require('SDL_util') Uint16p_value(ptr)
Description	This function returns the value pointed by the 16 bit integer address
Return Value	Returns a 16 bit integer value
Parameters	Ptr[in] : Address

Example:

```
local intp = new_Uint16p()
local value = 65535
intp = copy_Uint16p(value)
ret = Uint16p_value(intp)
```

13.2.298. Uint16p_assign

Name	Uint16p_assign
Synopsis	require('SDL_util') Uint16p_assign(ptr, value)
Description	This function assigns the value passed as an argument to the address
Return Value	none
Parameters	Ptr[in] : Address Value[in] : 16 bit integer value

Example :

```
local intp = new_Uint16p()
Uint16p_assign(intp, 65530)
ret = Uint16p_value(intp)
```

13.2.299. delete_Uint16p

Name	delete_Uint16p
Synopsis	require('SDL_util') delete_Uint16p(self)
Description	This function frees the memory of specified address
Return Value	None
Parameters	Ptr[in] : Address

Example :

```
local intp = new_Uint16p()
Uint16p_assign(intp, 65530)
ret = Uint16p_value(intp)
delete_Uint16p(intp)
ret = Uint16p_value(intp)
```

13.2.300. new_Uint32p

Name	new_Uint32p
Synopsis	require(‘SDL_util’) new_Uint32p()
Description	This function allocates a 32 bit memory
Return Value	Address
Parameters	None

Example:

```
local intp = new_Uint32p()
local value = 0xFFFFFFFF
intp =    copy_Uint32p(value)
ret = Uint32p_value(intp)
```

13.2.301. Uint32p_value

Name	Uint32p_value
Synopsis	require('SDL_util') Uint32p_value(ptr)
Description	This function returns the value pointed by the memory address
Return Value	Returns a 32 bit integer value
Parameters	Ptr[in] : Memory Address

Example:

```
local intp = new_Uint32p()
local value = 0xFFFFFFFF
intp =    copy_Uint32p(value)
ret = Uint32p_value(intp)
```

13.2.302. Uint32p_assign

Name	Uint32p_assign
Synopsis	require('SDL_util') Uint32p_assign(ptr, value)
Description	This function assigns the value passed as an argument to the memory address
Return Value	none
Parameters	Ptr[in] : Memory Address Value[in] : 32 bit integer value

Example:

```
local intp = new_Uint32p()
Uint32p_assign(intp, 0xffffffff)
ret = Uint32p_value(intp)
```

13.2.303. delete_Uint32p

Name	delete_Uint32p
Synopsis	require('SDL_util') delete_Uint32p(self)
Description	This function frees the specified address location
Return Value	none
Parameters	Ptr[in] : Memory Address

Example :

```
local intp = new_Uint32p()
Uint32p_assign(intp, 0xffffffff)
ret = Uint32p_value(intp)
delete_Uint32p(intp)
ret = Uint32p_value(intp)
```

13.2.304. new_chardp

Name	new_chardp
Synopsis	require('SDL_util') new_chardp()
Description	This function creates the double pointer for character array.
Return Value	Returns a memory address that stores character array.
Parameters	None

Example:

```
local ptr = new_chardp()
```

13.2.305. delete_chardp

Name	delete_chardp
Synopsis	require('SDL_util') delete_chardp(self)
Description	This function frees the memory containing string
Return Value	None
Parameters	Ptr[in] : Memory Address

Example:

```
local ptr = new_chardp()
local value = "Lua SDK Tutorial"
ptr =copy_chardp(value)
delete_chardp(ptr)
```

13.2.306. chardp_assign

Name	chardp_assign
Synopsis	require('SDL_util') chardp_assign(self, value)
Description	This function assigns the value passed as second parameter to self which is a memory address that stores string.
Return Value	None
Parameters	self [in] : Memory Address value[in]: value to be assigned to self

13.2.307. chardp_value

Name	chardp_value
Synopsis	require('SDL_util') chardp_value (self)
Description	This function returns the value from a memory address containing a string
Return Value	String
Parameters	self[in]: Memory Address

13.2.308. new_Uint8dp

Name	new_Uint8dp
Synopsis	require('SDL_util') new_Uint8dp()
Description	This function allocates memory for Uint8dp variable
Return Value	Allocated Memory
Parameters	None

13.2.309. copy_Uint8dp

Name	copy_Uint8dp
Synopsis	require('SDL_util') copy_Uint8dp(value)
Description	This function copies the Uint8 value to a variable and returns the address of that variable.
Return Value	Address of the variable where value has been copied
Parameters	value[in]: value to be copied

13.2.310. delete_Uint8dp

Name	delete_Uint8dp
Synopsis	require('SDL_util') delete_Uint8dp(self)
Description	This function frees the memory allocated for Uint8dp
Return Value	None
Parameters	self[in]: Memory Address

13.2.311. Uint8dp_assign

Name	Uint8dp_assign
Synopsis	require('SDL_util') Uint8dp_assign(self, value)
Description	This function assigns the value passed as second parameter to self which is a memory location for Uint8dp
Return Value	None
Parameters	self[in]: Memory location for Uint8 value[in]: Uint8 value

13.2.312. Uint8dp_value

Name	Uint8dp_value
Synopsis	require('SDL_util') Uint8dp_value(self)
Description	This function returns the value pointed to by the memory address forUint8 passed to it.
Return Value	Memory Address
Parameters	self[in]: Uint8 double pointer

13.2.313. new_ptrDouble

Name	new_ptrDouble
Synopsis	require('SDL_util') new_ptrDouble()
Description	This function creates a Double pointer and returns the address of this pointer.
Return Value	Double **
Parameters	none
Parameters	value[in]: value of type Double

13.2.314. delete_ptrDouble

Name	delete_ptrDouble
Synopsis	require('SDL_util') delete_ptrDouble(self)
Description	This function delete the Double pointer pointed to by the parameter passed to this function
Return Value	none
Parameters	self[in]:pointer of type Double

13.2.315. SDL_malloc

Name	SDL_malloc
Synopsis	require('SDL_util') SDL_malloc(size)
Description	This function allocates memory of size bytes and returns a pointer to this memory address.
Return Value	Pointer to memory assigned
Parameters	size[in]: number of bytes to be allocated

Example :

```
function myfunc()
        SDL_malloc(10)
end
```

13.2.316. Unit8_to_char

Name	Unit8_to_char
Synopsis	require('SDL_util') Unit8_to_char(x)
Description	This is a function used to change the type of a variable from Unit8 to char
Return Value	x :char type
Parameters	x [in]: Unit8 type

Example :

```
function myfunc(i)
    print("char value of x is".. Unit8_to_char(x) )
end
```

13.2.317. char_to_Uint8

Name	char_to_Uint8
Synopsis	require('SDL_util') char_to_Uint8(x)
Description	This is a function used to change the type of a variable from char to Unit8.
Return Value	x : Uint8 type
Parameters	x [in]: char type

Example :

```
function myfunc(i)
    print("Uint8 value of x is"..char_to_Uint8(x) )
end
```

13.2.318. Uint32_to_Uint16

Name	Uint32_to_Uint16
Synopsis	require('SDL_util') Uint32_to_Uint16(x)
Description	This is a function used to change the type of a variable from Uint32 to Uint 16..
Return Value	x : Uint16 type
Parameters	x [in]: Uint32 type

Example :

```
function myfunc(i)
    print("Uint16 value of x is"..Uint32_to_Uint16(x) )
end
```

13.2.319. Uint16_to_Uint32

Name	Uint16_to_Uint32
Synopsis	require('SDL_util') Uint16_to_Uint32(x)
Description	This is a function used to change the type of a variable from Uint16 to Uint 32.
Return Value	x : Uint32 type
Parameters	x [in]: Uint16 type

Example :

```
function myfunc(i)
    print("Uint32 value of x is"..Uint16_to_Uint32(x) )
end
```

13.2.320. Unit8p_to_charp

Name	Unit8p_to_charp
Synopsis	require('SDL_util') Unit8p_to_charp(x)
Description	This is a function used to change the type of a variable from Unit8 to charp.
Return Value	x : charp
Parameters	x [in]: Unit8 type

Example :

```
function myfunc(i)
    print("charp value of x is"..Unit8_to_charp(x) )
end
```

13.2.321. charp_to_Uint8p

Name	charp_to_Uint8p
Synopsis	require('SDL_util') charp_to_Uint8p(x)
Description	This is a function used to change the type of a variable from charp to Uint8p.
Return Value	x : Uint8p
Parameters	x [in]: charp type

Example :

```
function myfunc(i)
    print("Uint8p value of x is"..charp_to_Uint8p(x) )
end
```

13.2.322. charp_to_voidp

Name	charp_to_voidp
Synopsis	require('SDL_util') charp_to_voidp(x)
Description	This is a function used to change the type of a variable from charp to voidp.
Return Value	x : voidp
Parameters	x [in]: charp type

Example :

```
function myfunc(i)
    print("voidp value of x is"..charp_to_voidp(x) )
end
```

13.2.323. voidp_to_charp

Name	voidp_to_charp
Synopsis	require('SDL_util') voidp_to_charp(x)
Description	This is a function used to change the type of a variable from voidp to charp.
Return Value	x : charp
Parameters	x [in]: voidp type

Example :

```
function myfunc(i)
    print("charp value of x is"..voidp_to_charp(x) )
end
```

13.2.324. voidp_to_Uint8p

Name	voidp_to_Uint8p
Synopsis	require('SDL_util') voidp_to_Uint8p(x)
Description	This is a function used to change the type of a variable from voidp to Uint8p.
Return Value	x : Uint8p
Parameters	x [in]: voidp type

Example :

```
function myfunc(i)
    print("Uint8p value of x is"..voidp_to_Uint8p(x) )
end
```

13.2.325. Uint8p_to_voidp

Name	Uint8p_to_voidp
Synopsis	require('SDL_util') Uint8p_to_voidp(x)
Description	This is a function used to change the type of a variable from Uint8p to voidp.
Return Value	x : voidp
Parameters	x [in]: Uint8p type

Example :

```
function myfunc(i)
    print("voidp value of x is"..Uint8p_to_voidp(x) )
end
```

13.2.326. float_to_int

Name	float_to_int
Synopsis	require('SDL_util') float_to_int(x)
Description	This is a function used to change the type of a variable from float to int.
Return Value	x : int
Parameters	x [in]: float type

Example :

```
function myfunc(i)
    print("int value of x is"..float_to_int(x) )
end
```

13.2.327. int_to_Uint8

Name	int_to_Uint8
Synopsis	require('SDL_util') int_to_Uint8(x)
Description	This is a function used to change the type of a variable from int to Uint8.
Return Value	x : Uint8
Parameters	x [in]: int type

Example :

```
function myfunc(i)
    print("Uint8 value of x is"..int_to_Uint8(x) )
end
```

13.2.328. Uint8_to_int

Name	Uint8_to_int
Synopsis	require('SDL_util') Uint8_to_int(x)
Description	This is a function used to change the type of a variable from Uint8 to int.
Return Value	x : int
Parameters	x [in]: Uint8 type

Example :

```
function myfunc(i)
    print("Uint8 value of x is"..Uint8_to_int(x) )
end
```

13.2.329. intp_to_Uint8p

Name	intp_to_Uint8p
Synopsis	Require(SDL_util.lua) intp_to_Uint8p(x)
Description	This typecasts an integer pointer to Uint8 pointer
Return Value	Uint8 pointer
Parameters	x [in] : integer pointer

Example :

```
myiptr = new_intp()
myUiptr = intp_to_Uint8p(myiptr)
```

13.2.330. voidp_to_Uint16p

Name	voidp_to_Uint16p
Synopsis	require('SDL_util') voidp_to_Uint16p(x)
Description	This typecasts a void pointer to a Uint16 pointer
Return Value	Uint16 pointer
Parameters	x [in] : void pointer

Example :

```
myiptr = new_intp()
myvptr = intp_to_voidp(myiptr)
myUiptr = voidp_to_Uint16p(myvptr)
```

13.2.331. Uint16p_to_voidp

Name	Uint16p_to_voidp
Synopsis	require('SDL_util') Uint16p_to_voidp(x)
Description	This typecasts a Uint16 pointer to void pointer
Return Value	void pointer
Parameters	x [in] : Uint16 pointer

Example :

```
myUiptr = new_Uint16p()
myvptr = Uint16p_to_voidp(myUiptr)
```

13.2.332. voidp_to_Uint32p

Name	voidp_to_Uint32p
Synopsis	require('SDL_util') voidp_to_Uint32p(x)
Description	This typecasts a void pointer to Uint32 pointer
Return Value	Uint32 pointer
Parameters	x [in] : void pointer

Example :

```
myiptr = new_intp()
myvptr = intp_to_voidp(myiptr)
myUiptr = voidp_to_Uint32p(myvptr)
```

13.2.333. Uint32p_to_voidp

Name	Uint32p_to_voidp
Synopsis	require('SDL_util') Uint32p_to_voidp(x)
Description	This typecasts a Uint32 pointer to void pointer
Return Value	void pointer
Parameters	x [in] : Uint32 pointer

Example :

```
myUiptr = new_Uint32p()
myvptr = Uint32p_to_voidp(myUiptr)
```

13.2.334. voidp_to_intp

Name	voidp_to_intp
Synopsis	require('SDL_util') voidp_to_intp(x)
Description	This typecasts a void pointer to integer pointer
Return Value	integer pointer
Parameters	x [in] : void pointer

Example :

```
myUiptr = new_Uint32p()
myvptr = Uint32p_to_voidp(myUiptr)
myiptr = voidp_to_intp(myvptr)
```

13.2.335. intp_to_voidp

Name	intp_to_voidp
Synopsis	require('SDL_util') intp_to_voidp(x)
Description	This typecasts an integer pointer to void pointer
Return Value	void pointer
Parameters	x [in] : integer pointer

Example :

```
myiptr = new_intp()
myvptr = intp_to_voidp(myiptr)
```

13.2.336. boolp_to_voidp

Name	boolp_to_voidp
Synopsis	require('SDL_util') boolp_to_voidp(x)
Description	This typecasts an boolean pointer to void pointer
Return Value	void pointer
Parameters	x [in] : boolean pointer

13.2.337. voidp_to_boolp

Name	voidp_to_boolp
Synopsis	require('SDL_util') voidp_to_boolp (x)
Description	This typecasts an void pointer to boolean pointer
Return Value	boolean pointer
Parameters	x [in] : void pointer

13.2.338. delete_SDL_RectpAr

Name	delete_SDL_RectpAr
Synopsis	require('SDL_util') delete_SDL_RectpAr(ary)
Description	This deletes the array of SDL_Rect pointers.
Return Value	none
Parameters	ary [in] : pointer to the first SDL_Rect pointer in the array of SDL_Rect pointers

Example :

```
mySDLrectparr = new_SDL_RectpAr(5)
delete_SDL_RectpAr(mySDLrectparr)
```

13.2.339. SDL_RectpAr_getitem

Name	SDL_RectpAr_getitem
Synopsis	require('SDL_util') SDL_RectpAr_getitem(ary, index)
Description	This returns the pointer to SDL_Rect at location index in an array of SDL_Rect pointers
Return Value	SDL_Rect*
Parameters	ary [in] : pointer to first SDL_Rect pointer in SDL_Rect pointer array index [in] : offset of SDL_Rect pointer array Note: The first element of the SDL_Rect pointer array ary is ary[0].

Example :

```
mySDLrectparr = new_SDL_RectpAr(5)
mySDLrectpitem = SDL_RectpAr_getitem(mySDLrectparr, 0)
```

13.2.340. new_SDL_RectAr

Name	new_SDL_RectAr
Synopsis	require('SDL_util') new_SDL_RectAr(nelements)
Description	This creates a new SDL_Rect array
Return Value	Pointer to the first SDL_Rect in the array
Parameters	nelements [in] : number of array elements

Example :

mySDLrectarr = new_SDL_RectAr(5)

13.2.341. delete_SDL_RectAr

Name	delete_SDL_RectAr
Synopsis	require('SDL_util') delete_SDL_RectAr(ary)
Description	This deletes the SDL_Rect array
Return Value	none
Parameters	ary [in] : pointer to first SDL_Rect in the array

Example :

```
mySDLrectarr = new_SDL_RectAr(5)
delete_SDL_RectAr(mySDLrectarr)
```

13.2.342. SDL_RectAr_getitem

Name	SDL_RectAr_getitem
Synopsis	require('SDL_util') SDL_RectAr_getitem(ary, index)
Description	This returns the SDL_Rect at location index in the SDL_Rect array.
Return Value	SDL_Rect
Parameters	ary [in] : pointer to first SDL_Rect in SDL_Rect array index [in] : offset of the SDL_Rect array. Note: The first element of the SDL_Rect array ary is ary[0].

Example :

```
mySDLrectarr = new_SDL_RectAr(5)
mySDLrect = SDL_RectAr_getitem(mySDLrectarr, 0)
```

13.2.343. SDL_RectpAr_getitem

Name	SDL_RectpAr_getitem
Synopsis	require('SDL_util') SDL_RectpAr_getitem(ary, index)
Description	This returns the SDL_Rect pointer at location index in the SDL_Rect array.
Return Value	SDL_Rect
Parameters	ary [in] : pointer to first SDL_Rect in SDL_Rect array index [in] : offset of the SDL_Rect array. Note: The first element of the SDL_Rect array ary is ary[0].

13.2.344. SDL_ free

Name	SDL_free
Synopsis	require('SDL_util') SDL_free(Ins)
Description	The function shall cause the space pointed to by instance to be deallocated; that is, made available for further allocation.
Return Value	nil
Parameters	Ins[IN]:Instance

13.2.345. SDL_calloc

Name	SDL_calloc
Synopsis	require('SDL_util') SDL_calloc(size_t nelem, size_t elsize)
Description	The function shall allocate unused space for an array of *nelem* elements each of whose size in bytes is *elsize*. The space shall be initialized to all bits 0.
Return Value	Upon successful completion with both *nelem* and *elsize* non-zero, *calloc*() shall return a pointer to the allocated space.
Parameters	nelem[IN]: element size elsize[IN]: element size

13.2.346. SDL_realloc

Name	SDL_realloc
Synopsis	require('SDL_util') SDL_realloc (void *ptr, size_t size)
Description	The function shall change the size of the memory object pointed to by object to the size specified by size. The contents of the object shall remain unchanged up to the lesser of the new and old sizes.
Return Value	Upon successful completion with a size not equal to 0, realloc() shall return a pointer to the (possibly moved) allocated space.
Parameters	ptr[IN]: Instance size[IN]: size of memory to be allocate

13.2.347. SDL_memset

Name	SDL_memset
Synopsis	require('SDL_util') SDL_memset (void *s, int c, size_t n)
Description	The function copies c (converted to an unsigned char) into each of the first n bytes of the object pointed to by s.
Return Value	The function returns s; no return value is reserved to indicate an error.
Parameters	s1[IN]: string c[IN]: number n[IN]: number - length

13.2.348. SDL_ memcpy

Name	SDL_memcpy
Synopsis	require('SDL_util') SDL_memcpy (void *restrict s1, const void *restrict s2, size_t n)
Description	The function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined.
Return Value	The function shall return s1; no return value is reserved to indicate an error.
Parameters	s1[IN]: string s2[IN]: string n[IN]: number - length

13.2.349. SDL_ memmove

Name	SDL_memmove
Synopsis	require('SDL_util') SDL_memmove(void *s1, const void *s2, size_t n)
Description	The function shall copy n bytes from the object pointed to by s2 into the object pointed to by s1. Copying takes place as if the n bytes from the object pointed to by s2 are first copied into a temporary array of n bytes that does not overlap the objects pointed to by s1 and s2, and then the n bytes from the temporary array are copied into the object pointed to by s1.
Return Value	The function shall return s1; no return value is reserved to indicate an error.
Parameters	s1[IN]: string s2[IN]: string n[IN]: number - length

13.2.350. SDL_memcmp

Name	SDL_memcmp
Synopsis	require('SDL_util') SDL_memcmp (const void *s1, const void *s2, size_t n)
Description	The function shall compare the first n bytes (each interpreted as unsigned char) of the object pointed to by s1 to the first n bytes of the object pointed to by s2.
Return Value	The function shall return an integer greater than, equal to, or less than 0, if the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2, respectively.
Parameters	s1[IN]: string s2[IN]: string n[IN]: number - length

13.2.351. SDL_strlen

Name	SDL_strlen
Synopsis	require('SDL_util') SDL_strlen(string str)
Description	Getting the length of a string
Return Value	Length of the string
Parameters	str[IN]: string

13.2.352. SDL_strlcpy

Name	SDL_strlcpy
Synopsis	require('SDL_util') SDL_strlcpy(char *dst, const char *src, size_t size)
Description	The function copies up to size - 1 characters from the nil-terminated string src to dst, nil-terminating the result.
Return Value	return the total length of the string they tried to create.
Parameters	dst[IN]: destination string src[IN]: source string size[IN]: length

13.2.353. SDL_ strlcat

Name	SDL_strlcat
Synopsis	require('SDL_util') SDL_strlcat (char *dst, const char *src, size_t size)
Description	The function copies up to size - 1 characters from the nil-terminated string src to dst, nil-terminating the result.
Return Value	return the total length of the string they tried to create.
Parameters	dst[IN]: destination string src[IN]: source string size[IN]: length

13.2.354. SDL_ strdup

Name	SDL_strdup
Synopsis	require('SDL_util') SDL_strdup (string str)
Description	The function shall return a pointer to a new string, which is a duplicate of the string pointed to by s. The returned pointer can be passed to free(). A null pointer is returned if the new string cannot be created.
Return Value	The function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set errno to indicate the error.
Parameters	str[IN]: string

13.2.355. SDL_ strrev

Name	SDL_strrev
Synopsis	require('SDL_util') SDL_strrev(string str)
Description	Reverse the string
Return Value	Reversed string address
Parameters	str[IN]: string

13.2.356. SDL_strupr

Name	SDL_strupr
Synopsis	require('SDL_util') SDL_strupr(string str)
Description	Convert a string to uppercase
Return Value	Upper case string's address
Parameters	str[IN]: string

13.2.357. SDL_strlwr

Name	SDL_strlwr
Synopsis	require('SDL_util') SDL_strlwr(string str)
Description	Convert a string to lowercase
Return Value	Lower case string's address
Parameters	str[IN]: string

13.2.358. SDL_strchr

Name	SDL_strchr
Synopsis	require('SDL_util') SDL_strchr(const char *s, int c)
Description	The function shall locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string.
Return Value	Upon completion, strchr() shall return a pointer to the byte, or a null pointer if the byte was not found.
Parameters	s[IN]: string c[IN]: number

13.2.359. SDL_strrchr

Name	SDL_strrchr
Synopsis	require('SDL_util') SDL_strrchr(const char *s, int c)
Description	The function shall locate the last occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string.
Return Value	Upon successful completion, strrchr() shall return a pointer to the byte or a null pointer if c does not occur in the string.
Parameters	s[IN]: string c[IN]: number

13.2.360. SDL_strstr

Name	SDL_strstr
Synopsis	require('SDL_util') SDL_strstr(const char *s1, const char *s2)
Description	The function shall locate the first occurrence in the string pointed to by *s1* of the sequence of bytes (excluding the terminating null byte) in the string pointed to by*s2*.
Return Value	Upon successful completion, this function shall return a pointer to the located string or a null pointer if the string is not found.
Parameters	s1[IN]: string s2[IN]: string

13.2.361. SDL_itoa

Name	SDL_itoa
Synopsis	require('SDL_util') SDL_itoa(int value, char * str, int base)
Description	Value to be converted to a string
Return Value	A pointer to the resulting null-terminated string, same as parameter str.
Parameters	value[IN]:Value to be converted to a string. str[IN]:Array in memory where to store the resulting null-terminated string. base[IN]: Numerical base used to represent the value as a string, between 2 and 36, where 10 means decimal base, 16 hexadecimal, 8 octal, and 2 binary.

13.2.362. SDL_ltoa

Name	SDL_ltoa
Synopsis	require('SDL_util') SDL_ltoa(long value,char* buffer,int radix)
Description	convert a long interger to a string
Return Value	An Instance to the result
Parameters	value [IN]:The value to convert into a string. buffer[IN]:A buffer in which the function stores the string. The size of the buffer must be at least 33 bytes when converting values in base 2 (binary). radix[IN]:The base to use when converting the number. This value must be in the range:2 ≤ radix ≤ 36

13.2.363. SDL_strcmp

Name	SDL_strcmp
Synopsis	require('SDL_util') SDL_strcmp(const char *str1, const char *str2)
Description	The function shall compare the string pointed to by s1 to the string pointed to by s2.
Return Value	Upon completion, this function shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2, respectively.
Parameters	s1[IN]: string s2[IN]: string

13.2.364. SDL_strncmp

Name	SDL_strncmp
Synopsis	require('SDL_util') SDL_strncmp(const char *s1, const char *s2, size_t n)
Description	The function shall compare not more than n bytes (bytes that follow a null byte are not compared) from the array pointed to by s1 to the array pointed to by s2.
Return Value	Upon successful completion, this function shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2 respectively.
Parameters	s1[IN]: string s2[IN]: string n[IN]: number - length

13.2.365. SDL_strcasecmp

Name	SDL_strcasecmp
Synopsis	require('SDL_util') SDL_strcasecmp(const char *s1, const char *s2)
Description	The function compares, while ignoring differences in case, the string pointed to by s1 to the string pointed to by s2. The strncasecmp() function compares, while ignoring differences in case, not more than n bytes from the string pointed to by s1 to the string pointed to by s2.
Return Value	Upon completion, this function shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is, ignoring case, greater than, equal to, or less than the string pointed to by s2, respectively.
Parameters	s1[IN]: string s2[IN]: string

13.2.366. SDL_strncasecmp

Name	SDL_strncasecmp
Synopsis	require('SDL_util') SDL_strncasecmp(const char *s1, const char *s2, size_t n)
Description	The function performs a case-insensitive string comparison for up to the first n characters of the strings pointed to by s1 and s2.
Return Value	The function returns a positive integer if, within the first n bytes of string s1 and string s2, disregarding case string s1 is lexically greater than string s2. It returns zero if, other than case, the first n bytes of the two strings are identical. Finally, strncasecmp() returns a negative integer if with the first n bytes, disregarding case string s1 is lexically less than string s2.
Parameters	s1[IN]: string s2[IN]: string n[IN]: number - length

13.2.367. SDL_memset4

Name	SDL_memset4
Synopsis	require('SDL_util') SDL_memset4(dst, val, len)
Description	Setting up the memory.
Return Value	Memory address
Parameters	dst[IN]: destination val[IN]: value len[IN]: length

13.3.Application Code

The application code is available on Samsung website and CD along with the product.

13.4.Contact Information

For any support kindly contact Samsung Developer Forum (SDF).