redbytes™

Appify Your Business

HOW TO BUILD A GAME USING CORONA SDK

1. CORONA (45 Minutes)

Our tutorials provide practical guidelines intended to make beginners conversant with Corona platform. It covers all topics ranging from initial installation to development of mobile applications thus, providing an approach to have a crystal clear understanding of basic to advance concepts.

1.1 Why Corona?

Corona is an ultimate platform for the development of 2D game and native applications.
This particular software development kit allows publishing on all the major platforms including iOS (including Apple TV), Android (including Android TV), Kindle, Mac, Windows, etc.
It has its own featured API library, which even allows running complex functions such as cryptography and networking with only a few lines of code.

– provides high performance graphics
– Simple and high-performance
– Cross platform
– Uses Lua scripting language which is easy-to-learn and embed
– Develops an apps 10x faster
– Huge set of robust and innovative API library
– Featured plug-ins and physics engines
– Provides a real-time simulator
– Cost effective

1.2 Prerequisites

Basic understanding of any programming language like C, Java, etc.

1.3 Lua:

Lua is an open source, cross-platform, multi-paradigm programming language which is layered on the top of C language. It is designed as a scripting language which is highly portable and can run on a wide range of devices from high-end network to small devices.

It consists of two parts:

- Lua Interpreter and
- Functioning software system

1.3.1 Variable declaration

Variables are containers or a storage area that holds a certain value. There are 3 kinds of variables in Lua:

a. Local variables: These variables are indicated using local keyword. Scope of these variables will be within the block in which they are defined and can't be accessed outside that particular block. There is no default value for local variables, so their initial value should be assigned before their first use.

Eg:

```
local a
local b = {}
```

b. Global variables: Default variables are treated as global variables. To create a global variable, you simply need to assign a value instead of declaring. These variables are visible throughout the program and remain alive till your application is in running state.

Eg:

```
b = {}
b = {10, 20}
```

c. Table fields:

Lua tables can be used like arrays. To assign values to a table field, you need to index values (numbers, strings, etc. except nil) in an array.

Eg:

```
local  b = display.newRect(100,200, 300, 400)
b.value = 10
b.number = 20
b.name = “Corona”
```

1.3.2 Function Declaration

A function is a group of statements that are used to perform a certain task and return a value, as per the requirement. Following example specifies function declaration:

Eg:

```
local method
method = function (int x)
    print("Value", x)
end
```

1.3.3 Loops

Loops are used in a situation when a block of a code or a certain process needs to be executed several times. There are several types of loops in Lua, few are listed below:

a. for loop
b. while loop
c. repeat...until loop

a. For loop:

This loop is designed to repeat a certain block of code number of times.

Syntax:

```
for variable = initialVal, finalVal, incrementVal do
// code to be executed
end
```

where: initialVal: first value which is to be displayed
flagVal: last value which is to be displayed
incrementVal: difference between the next value to be printed and the preceding value.

Eg:

```
for i = 1, 7, 2 do
        print( i )
end
```

Output is => 1
3
5
7

Explanation:

Printing starts from initial value specified I.e 1. The next value printed is incremented by the specified incrementVal I.e next value = 1 + incrementVal(2).
Similarly, the succeeding values are incremented by incrementVal =< end value to be displayed.

=> Program number : 1

Write a Lua program using for loop to print even numbers from 1 to 50

--> Code 1:

```
local n = 50

print("Even numbers 1 to "..n)

  for i = 1, n do

    if i%2 == 0 then

    print(i)

  end
```

```
end
```

Output:-

```
Even numbers 1 to 50
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
```

Explanation:

The above program displays the number between 1 (initialVal) and 50 (finalVal). We are suppose to print only even numbers, logic (i%2 == 0) of dividing the number by 2 has been used. If the remainder is zero, then that particular number is considered as

even and is displayed. For loop iteration starts with 1 and iterates till value has reached 50.

=> Program number : 2
Write a program using for loop to print even numbers 50 to 1

- ->Code 2:

```
local n = 50

print("Even numbers "..n.." to 1")

  for i = n, 1, -1 do

     if i%2 == 0 then

      print(i)

   end

end
```

Output:-

```
Even numbers 50 to 1
50
48
46
44
42
40
38
36
34
```

32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2

Explanation:

The above program displays the number between 50 (initialVal) and 1 (finalVal). We are suppose to print only even numbers, logic (i%2 == 0) of dividing the number by 2 has been used. If the remainder is zero, then that particular number is considered as even and is displayed. For loop iteration starts with 50 and iterates till value has reached 1.
This program is similar to previous one, just in reverse order.

=> Program number : 3
Write a program in Lua using for loop to print multiplication table of a particular number.

-> Code 3:

local n = 6

```
print("Multiplication Table of ", n)

for i = 1, 10 do

   print(n," * ", i, " = ",n*i)

end
```

Output:-

```
Multiplication Table of      6
6   *     1     =     6
6   *     2     =     12
6   *     3     =     18
6   *     4     =     24
6   *     5     =     30
6   *     6     =     36
6   *     7     =     42
6   *     8     =     48
6   *     9     =     54
6   *     10    =     60
```

Explanation:
In the above program, for loop iterates from1 to 10 as we have to display multiplication table. Three numbers are being printed:

1st : our particular number for which multiplication table is to be made. Considered to be 6.

2nd : index of for loop iterates from 1 (initialVal) to 10 (finalVal).

3rd : result of multiplication of number and index (6 * index).

b. While loop:

This loop is designed to repeat a certain block of code as long as the given condition is true.

Syntax:

```
while condition do
        // code_to_executed
end
```

where condition: statement or expression which could be true or false.
code_to_executed: if condition is true, this code is iterated, else not.

Eg:

```
local i = 1
while i<3 then
    print("Hello")
    i = i + 1
end
```

Output is => Hello
Hello

Program number: 1
Write a program to print the product of digits.

Code 1:

```
 local number = 2351
local NO = number
 local product = 1
 while(number >=1) do
        local rmd = math.floor(number)%10
        number = math.floor(number)/10
```

```
        product = product * rmd
 end
print("The product of digits of number "..NO.." is "..product)
```

Output :
The product of digits of number 2351 is 30

Explanation:
-- Find last digit of number by performing modulo division by 10 i.e. lastDigit = num % 10.
-- Multiply last digit found above with product i.e. product = product * rmd.
– Remove last digit by dividing the number by 10 i.e. number = number / 10.
-- Finally you will be left with product of digits in product variable.

=> Program number: 2
Write a program in Lua to print the factorial of a number.

Code 2:

```
local Num = 5
 local NO = Num
local fact = 1

        while(Num > 1 ) do
        fact = fact * Num
        Num = Num - 1
end

print("The factorial of "..NO.." is "..fact)
```

Explanation:

Factorial of a number n is product of all positive integers less than or equal

to n. It is denoted as n!.
For example factorial of 5 = 1 * 2 * 3 * 4 * 5 = 120

Step by step descriptive logic to find factorial of a number:

-- Input a number from user. Store it in some variable say Num.
-- Initialize another variable that will store factorial say fact = 1.
Why initialize fact with 1 not with 0?
This is because you need to perform multiplication operation not summation.
-- Multiplying 1 by any number results same, same as summation of 0 and any other number results same.

-- Iterate a loop from 1 to num, increment by 1 in each iteration.
-- Multiply the Num value i.e. Num with fact i.e fact = fact * Num.

=>
Program Number: 3
Write a program to print fibonacci series upto 10.

--> Code 3:

```
local a,b,c = 0,1,0
print("The fibonacci series from 1 to 10 is")
print(a)
print(b)
while(c < 10) do
        c = a + b
        a = b
        b = c
        print(c)
end
```

Output
 is:
The fibonacci series from 1 to 10 is

0, 1, 1, 2, 3, 5, 8,13

Explanation:

Fibonacci series is a series of numbers where the current number is the sum of previous two terms.
For Example: 0, 1, 1, 2, 3, 5, 8, 13, 21, ... , (n-1th + n-2th)

Step by step descriptive logic to print n Fibonacci terms:

-- Declare and initialize three variables, call it as Fibonacci magic initialization. a=0, b=1 and c=0.
where c is the 3rd term, a is the 1st term and b is 2nd term.

-- Run a loop from 1 to n terms, increase loop counter by 1. It will iterate through n terms
-- Inside the loop copy the value of 2nd term to 1st term i.e. a = b.
-- Next, copy the value of 3rd to 2nd term b = c.

-- Finally compute the new term by adding previous two terms i.e. c = a + b.

-- Print the value of current Fibonacci term i.e. c.

c. repeat...until loop

This loop repeats a certain group of statements till until condition is met.

Syntax:

```
repeat
    //  code to be executed
until condition
```

Example:

```
i = 2

repeat

    print (i", ")
    i = 2 + i
    until i > 7

    Output: 2,  4,  6,
```

=> Program Number: 1

Write a program to print the factors of any number.

Code 1:

```
local i, num = 1,100

print("Factors of number",num)

    repeat

        if num%i == 0 then

        print(i)

    end

    i = i+1

    until( i>num )
```

Output is =>

Factors of number 100

1 2 4 5 10 20 25 50 100

Explanation:

What is factor of a number?

Factor of any number is a whole number which exactly divides the number into a whole number without leaving any remainder.

For example: 2 is a factor of 6 because 2 divides 6 exactly leaving no remainder.

Step by step descriptive logic to find all factors of a number:

-- Store number in some variable say “num”.

-- Iterate a loop from 1 to num, increment by 1 in each iteration.

-- For each iteration inside loop check current, whether the counter loop variable i is a factor of num or not.

-- To check factor we check divisibility of a number by performing modulo division i.e. if(num % i == 0)

-- then i is a factor of num.

-- If i is a factor of num then print the value of i.

=> Program Number: 2

Write a program in Lua to check whether a number is Perfect number or not.

Code 2:

local num = 6

```
local sum = 0

local i = 1

 repeat

        if (num)%i == 0 then

                        sum = sum + i;

        end

        i = i+1

until( i>num/2 )


if sum == num then

        print(num ," is PERFECT NUMBER")

else

         print(num ," is NOT PERFECT NUMBER");

end
```

Output:

6 is PERFECT NUMBER

Explanation:

What is Perfect number?

Perfect number is a positive integer which is equal to the sum of its proper positive divisors.

For example: 6 is the first perfect number

-- Proper divisors of 6 are 1, 2, 3

-- Sum of its proper divisors = 1 + 2 + 3 = 6.

-- Hence 6 is a perfect number.

Step by step descriptive logic to check Perfect number.

-- Store a number in a variable say “num”.

-- Initialize another variable to store sum of proper positive divisors, say sum = 0.

c) Repeat until loop with programs

-- repeat a loop from 1 to until i>num/2, increment 1 in each iteration.

--Why iterating from 1 to until i>num/2, why not till num?

Because a number does not have any proper positive divisor greater than num/2.

--Inside the loop if current number i.e. i is proper positive divisor of num, then add it to sum.

--Learn - check divisibility of a number.

Check if the sum of proper positive divisors equals to the original number.

Then, the given number is Perfect number otherwise not.

==> Program Number: 3

Write a program in Lua to check whether a number is a Prime number or not.

Code 3:

```
local num = 17

local isPrime = true // isPrime is used as flag variable.

local i = 2

 repeat

   if (num)%i == 0 then

                isPrime = false;

   end

   i = i+1

until( i>num/2 )

if isPrime == true then

        print(num ," is prime number")

else

        print(num ," is composite number");

end
```

Output:

17 is prime number

== >Explanation:

What is a Prime number?

Prime numbers are the positive integers greater than 1 that are only divisible by 1 and self. For example: 2, 3, 5, 7, 11, etc.

Step by step descriptive logic to check prime number:

– Store a number in a variable say “num”

– Declare and initialize another variable say isPrime = true.

-- isPrime variable is used as a notification or flag variable.

-- Assigning false means number is composite and true means prime.

--Run a loop from 2 to num/2, increment 1 in each iteration.

--Check, divisibility of the number i.e. if(num%i == 0) then,

the number is not prime.

Set isPrime = false indicating number is not prime and terminate from loop.

--Outside the loop check the current value of isPrime.

--According to our assumption if it is equal to 1 then the number is prime otherwise composite.

1.3.4 Conditions

Conditions are used to control the flow of the program. It allows a program to make decision as per the condition and the input provided. There are several types of conditions, for instance: If statement, If-else statement, If-else-if statement, etc.

If-else condition:
In this condition, if statement is followed by an else statement. If provided Boolean condition evaluates to true, then ’if’ block is executed, otherwise, the ‘else’ block will be executed.

Eg:

```
i = 10
j = 20
if  i < j then
        print(“Value ”, i)
        else
                print(“value”, j)
        end
```

2. Getting ready with Corona SDK (45 minutes)

2.1 System Requirement for Corona Application Development:

In order to develop applications using Corona SDK, you need to setup a congenial environment.
You need to have below mentioned system elements depending upon the platform availability:

On MacOS:
– macOS 10.11 or higher
– Xcode

On Windows:
- Windows 8, Windows 7, Vista, or XP operating system
- 1 GHz processor
- 1 GB of RAM
- OpenGL 2.1 or higher graphics system

2.2 Installation steps – macOS

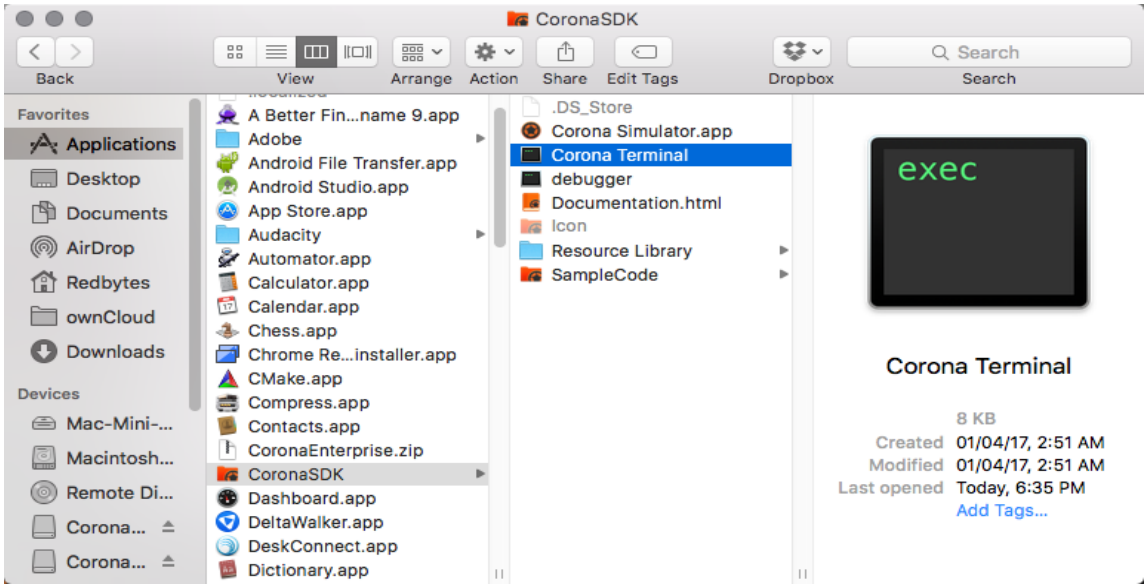
1. Download and install

- You need to download the Corona SDK from the respective link

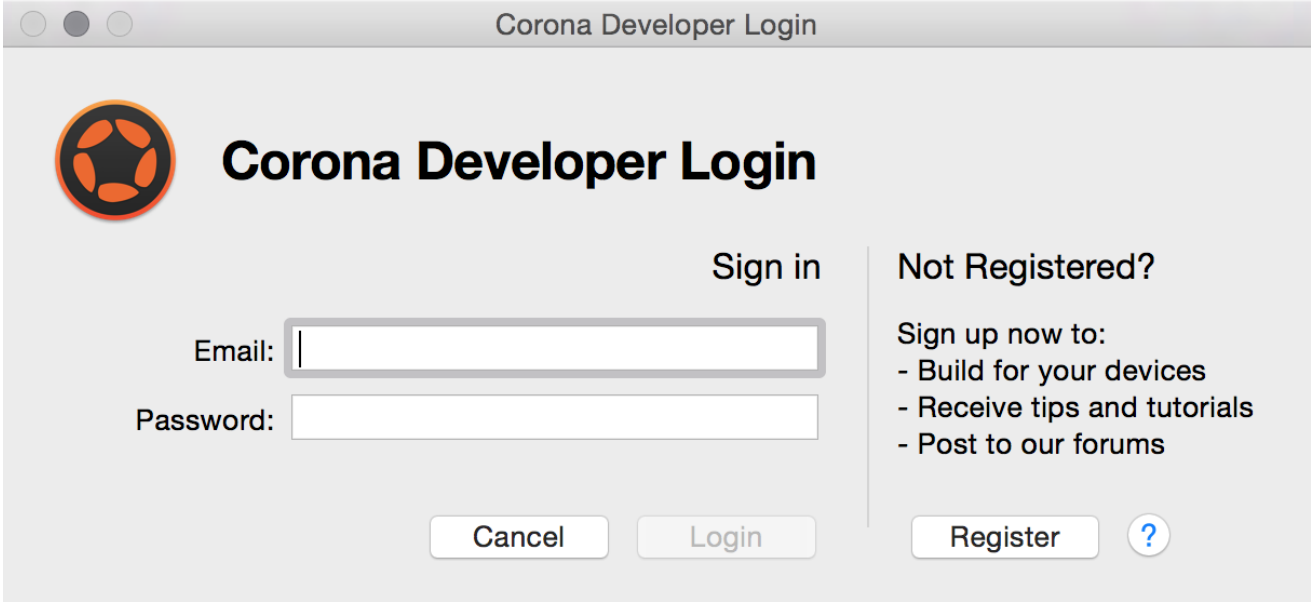
http://developer.coronalabs.com/downloads/coronasdk/
- Once you have downloaded, you need to double-click on the .dmg file.
It will further display the below window.

You need to drag the Corona SDK folder and drop it into the Applications folder, thus copying the contents into the destination folder.

- Further, you need to open Corona terminal from the folder where you have installed it, as shown in the below image.

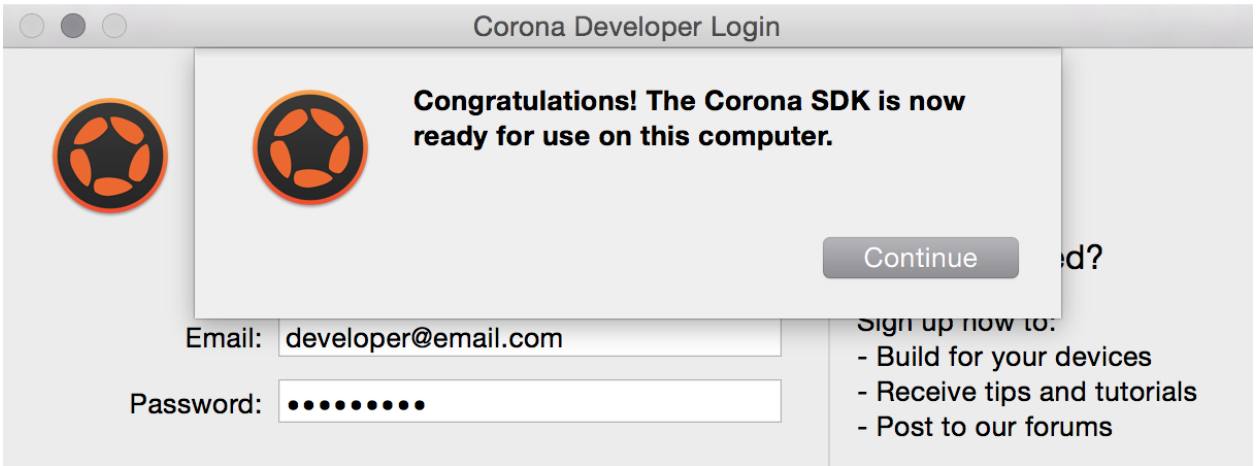


Accept the license terms to continue with the installation. It will display the below screen.



If you are already having an account, just provide your e-mail and password to initiate activation of the product. If you do not have an account, you need to register to create an account.

-Click on the pop up window ('continue')and you are done with the installation procedure. You can login to proceed.



2.3 Setting up the required environment:

Config.lua: is a Lua file which mainly intends to configure the app in advance of its primary functionality starting in main.lua. It basically manages certain alignment operations regarding the content area of the screen. You set your desired width/height and Corona will scale that up or down to fit the actual pixel resolution of the device. Config.lua is processed after the app is initialized. This file is, set up using a content table nested within an application table as given below:

```
application =
{
content =
{
width = 768,
height = 1024,
scale = "adaptive",
fps = 30
}
}
local aspectRatio = display.pixelHeight / display.pixelWidth

application = {
content = {
width = aspectRatio > 1.5 and 320 or math.ceil( 480 / aspectRatio ),
height = aspectRatio < 1.5 and 480 or math.ceil( 320 * aspectRatio ),
scale = "letterBox",
}
}
```

build.settings:

Properties of an app need to be defined at the time of creation which are included in a file named build.settings (written in Lua script). It includes device specific entries such as version, permissions for Android, iOS or Windows. This

is also the place where orientations have to be defined, since they are set in the device build before Corona starts up.

At the basic level build.settings file should contain a setting table which can further contain customized nested tables depending on the device and the requirement.

```
Settings =
{
   --nested table here
}
```

Build Settings for iOS platform:

Below the 'settings' table, within an 'iPhone' table (for all iOS devices, not specifically for iphones) you need to include a 'plist' table. This plist table is used to set values for the compiled apps. Values need to be defined within this plist table for the compiled app.

```
settings =
{
   iphone =
   {
      plist =
      {
         CFBundleIconFiles = {},  -- Required!
         UILaunchStoryboardName = "LaunchScreen",  -- Required!
         UIStatusBarHidden = true,
         CFBundleDisplayName = "Corona App",
         CFBundleName = "Corona App",
      },
   },
}
```

Build Settings for Android platform :

Below the 'settings' table, you can include an ' Android' table to define build settings for Android devices.

```
settings =
{
   android =  {  },
}
```

Build Settings for Windows platform (macOS desktop or Win32 desktop app):

Below the 'settings' table, values are defined within a window table for desktops.

```
settings =
{
   window =  {  },
}
```

App Orientation:

In 'build settings' file app orientation can be attained with respect to device's physical orientation in space. Entire orientation settings must be included within the orientation table under settings.

```
settings =
{
   orientation =   {   },
}
```

User is given the privilege to add two optional keys: default and supported. if required.

For instance:

```
settings =
{
```

```
   orientation =
   {
      default = "portrait",
      supported = {"portrait", "portraitUpsideDown"},
   },
}
```

Auto-orientation:

Auto-orientation comes into picture when a device is rotated or flipped during the runtime and then it is triggered by an accelerometer. It can be used to lock the orientation in one specific pattern or could support more than one pattern.

For instance:
supported = {"portrait", "portraitUpsideDown"},

Permissions:

When an application runs on your respective device, it is authorized to access limited number of the resources. User gets a prompt alert whenever a system needs to access a restricted resource. This alert specifies the purpose to grant access and can be customized as per the requirement.
To ensure the secure flow, brief description is specified in plist table of build.settings.

For instance:

```
settings {
   iphone =    {
      plist =       {
         NSCameraUsageDescription = "This app would like to access the camera.",
         NSPhotoLibraryUsageDescription = "This app would like to access the photo library.",
      },
   },
```

}

For Android:

The usesPermissions table creates <uses-permission> tags in the AndroidManifest.xml file:

```
settings
        {
          android          =  {
            usesPermissions          =  {
              "android.permission.INTERNET",
"android.permission.WRITE_EXTERNAL_STORAGE",
              "android.permission.ACCESS_FINE_LOCATION",

"android.permission.ACCESS_COARSE_LOCATION",
            },
          },
        }
```

Version Code for Android:

Version code is an integer number used to identify the version of the app. When an upgraded version of an app is to be released on a play store, it is necessary to replace it with a preceding number. In build.settings file, a

key-value pair is used.
For instance:

```
settings
{
   android =  {
      versionCode = "1",
   },
}
```

2.3 Project Structure:

Having a common directory layout would ensure smooth functioning during the development of the app. If an organized structure is maintained and subdirectories/subfolders are named as per the standard, it becomes more convenient to access a particular file.
A project folder should contain at least three files mentioned below and the remaining files/folders could be added as per the areas of functionality or requirement

1. main.lua
2. config.lua and
3. build.settings

- **Project_Folder/** *(top-level)*
  - **images/** *(folder)*
  - **audio/** *(folder)*
  - **videos/** *(folder)*
  - **data/** *(folder)*
  - **scripts/** *(folder)*
  - main.lua
  - config.lua
  - build.settings
  - Default.png
  - Default@2x.png
  - Icon.png
  - Icon@2x.png

3. Composer and its functions (20 minutes)

3.1 Scene Management:

A scene is current screen or page of the app visible to the user.

Composer is a centralized screen/scene management library, responsible for keeping a track when a switch is made from one scene to another in an application. It basically comes into picture in an application with multiple pages. When a transition is made, it handles unloading of a screen when a new one is loaded.
Depending on the requirement several actions can be performed by sample of templates available. For instance to create a new scene file, following function can be used:

```
local composer = require( "composer" )
local scene = composer.newScene()
```

Life cycle of a Composer starts from main.lua file which is the first file that a Corona simulator will look for when it is run. It not only gives the provision to pass data to the scene but also to add several transition effects such as: fade, crossFade, zoomOutIn, zoomOutInFade, etc. First scene appears after the execution of composer.gotoScene() command. Standard gotoScene command API is:

```
composer.gotoScene(sceneName [,option])
```

where sceneName: the lua file (withuout .lua extension) which is to be loaded.

Option: used to specify the type of transition effect and the amount of time it is to be implemented.

For instance: After the execution of the below mentioned command, 'firstScene' will appear with the 'slideDown' effect over the timestamp of 200 milliseconds.

```
local composer = require( "composer" )
composer.gotoScene( "nextScene", {effect=”slideDown”, time =200} )


local options =
{
effect = "slideDown",
time = 300,
params = {
para1 = "",
para2 = ""
}
}
composer.gotoScene( "nextScene", options )
```

Scene Events:

Life cycle of a scene comprises of four life cycle events: create, show, hide and destroy, which a composer can utilize.

Create: This will load all the information required before displaying the scene. If scene has already been visited before and is present in the memory, it will not execute. It ensures effective memory usage.

Both the events, scene: show() and scene:hide() are executed simultaneously when a switch is made from one screen to another.

Show: This event is called each time scene is to be displayed and is executed in below mentioned two phases:

Will: When scene is off screen, just before it is about to appear on screen. It gives the provision to reset the variable values, in order to bring it back to its start point.

Did: Executed when is scene is displayed on the screen.

Hide: This event is called each time when current scene is to be hidden from screen. It is called twice, once for each of these parameters:

Will: Executed when scene is on screen and is just about to go off screen. It gives the provision to pause or stop physics, cancel timers and transitions, and also stop scene-specific audio that was played in scene:show().

Did: Event is directed to 'did' parameter, after scene is off screen. Composer stores these scenes in memory in the event of displaying them again.

Destroy: It comes into picture when scene:destroy() is called in the event of destroying the scene i.e Composer nullifies the scene's display object. Composer keeps the scenes in the memory if there arises a requirement of again displaying that particular scene.

Code:

```
--=============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--=============================================================
--=====================Required Libraries=======================
local composer = require( "composer" )
--================Variables Declarations=========================
local scene = composer.newScene()
--====================Composer Functions=======================
function scene:create( event )
local sceneGroup = self.view
print("Scene is created.")
end

function scene:show( event )
local sceneGroup = self.view
local phase = event.phase
print("This event occur twice")
if ( phase == "will" ) then
print("Scene's show will phase.The scene is about to become active")
```

```
elseif ( phase == "did" ) then
print("Scene's show did phase.After the scene comes on screen")
end
end

function scene:hide( event )
local sceneGroup = self.view
local phase = event.phase
print("This event occur twice")
if ( phase == "will" ) then
print("Scene's hide will phase.when the scene is on screen")
elseif ( phase == "did" ) then
print("Scene's hide did phase. immediately after the scene goes off screen")
end
end

function scene:destroy( event )
local sceneGroup = self.view
print("Scene's destroy phase.")
end

---------------------------------------------------------------------------------
scene:addEventListener( "create", scene )
scene:addEventListener( "show", scene )
scene:addEventListener( "hide", scene )
scene:addEventListener( "destroy", scene )
---------------------------------------------------------------------------------

return scene
```

3.3 Functions:

Composer API is a collection of simplified events, to mention few functions:

- composer.setVariable()

it is used to set a variable within the Composer module which you can access from any other scene via composer.getVariable().
Syntax:
composer.setVariable(“variableName”,value)
where variableName: specifies the name of the variable to be set.

Eg:
composer.setVariable(“count”,10)

1.
2. -composer.getVariable()
3.
4. It is used to return the value of the variable which was set using composer.setVariable().

Syntax:
composer.getVariable(“variableName”)
where variableName: specifies the name of the variable to be retrieved.
Eg:

composer.getVariable(
“count”)

-composer.getSceneName():
1.
2. It is used to return the name of the current scene.

Syntax:
composer.getSceneName(scene_Name)

where scene_Name: specifies the name of the scene to be retrieved.

Eg:
1.
2. composer.getSceneName(“current”)

-composer.newScene()
It is used to create a new scene().
Syntax:
composer.newScene()

-composer.removeHidden():
It is used to removes or recycle all scene excluding the current scene.

Syntax:
composer.removeHidden() // removes all the scenes excluding the current active scene.
composer.removeHidden([toBeRecycled]) // recycles all the scenes excluding the current active scene.

where toBeRecycled: This parameter should be set to true, if you want to recycle a particular scene. Default value is false.

-composer.showOverlay()

It is used to show the overlay scene on the top of the currently active scene.
Syntax:
composer.showOverlay(scene_Name [, options])
where scene_Name: specifies the name of the scene to be displayed as an overlay.
options: it specifies the customized transition effects which are to be added.
Eg:
local option = {
time = 700,
isModal = true,
effect = effect,
params = {}
}
composer.showOverlay("login", option)

-composer.hideOverlay()
It is used to hide the current overlay scene.

Syntax:
composer.hideOverlay([toBeRecycled] [, effect] [, time])
where toBeRecycled: This parameter should be set to true, if you want to recycle a particular scene. It removes the self.view display object but the scene will be present in the memory.
effect: transition effects to be added to the overlay before it is hidden.
time: specifies the timestamp for which effect is to be added.

Eg:

```
composer.hideOverlay()
```

-- It removes the current scene and show the background scene which is active.

composer.hideOverlay("fadeOut",400)

4. Display Library (1 hour)

Corona display object comprises of a wide range of visual objects that gives the privilege to use images, text, shapes, lines, etc. The properties of these objects can be customized as per the requirement.

4.1 Properties:

-display.actualContentHeight
It returns the actual corona content height (in units) present in the screen.

Syntax:
print(display.actualContentHeight)

-display.actualContentWidth
It returns the actual corona content width (in units) present in the screen.

Syntax:
print(display.actualContentWidth)

-display.pixelHeight
It returns the height (in pixels) of the screen for a mobile or desktop app with consideration to screen orientation.

Syntax:
print (display.pixelHeight)

-display.pixelWidth
It returns the width (in pixels) of the screen for a mobile or desktop app with consideration to screen orientation.

Syntax:
print (display.pixelWidth)

-display.currentStage
It returns the reference with respect to the current stage of the object i.e the root group for all display groups and objects.

-display.screenOriginY
It returns the vertical distance (in corona units) from the top of the screen to the top of the content area.

-display.screenOriginX
It returns the horizontal distance (in corona units) from the left side of the screen to the left side of the content area.

-display.contentCenterX

It returns the center of the actual content area along the x axis.

-display.contentCenterY

It returns the center of the actual content area along the y axis.

4.2 Methods

Methods can be used to achieve customized effects with respect to creation of the objects as per the requirement.

-display.newCircle()
It is used to create a circle with a specified radii centered at mentioned coordinates.

Syntax:
display.newCircle([parent,] xCoordinate, yCoordinate, radius)
where parent: specifies display group in which circle is to be inserted, if required.

xCoordinate, yCoordinate: specifies the x and y coordinates with respect to center of the circle.
radii: A natural number that specifies the radius of the circle.

Eg:
local drawCircle = display.newCircle(150, 150, 60)
drawCircle:setFillColor(0.3)

-display.newLine()
It is used to create a line from one point to another. You are also given the privilege to append points to the end of the line, if required.

Syntax:
display.newLine([parent,] x1, y1, x2, y2 [,x3, y3,...})

where parent: specifies display group in which line is to be inserted, if required.

x1, y1: specifies the coordinates with respect to the starting point of the line.
x2, y2: specifies the coordinates with respect to the end point of the line.
x3, y3: Optional coordinates, to be added as per the requirements.

Eg:
local line = display.newLine(300, 200, 500, 200)

-display.newText()
It is used to display a text in single and multi line.

Syntax:
display.newText(options)
where options: is a single argument which can take several parameters such as text (text string to be displayed), x and y coordinates (with respect to text object), parent (display group in which text box is to be inserted), height and width of the text box, font and fontSize of the displayed text. These parameters are optional and are only inserted, if required.

Eg:

```
local text = display.newText( "Hello World!", 200, 200, 0, 0, native.systemFont, 14
)
text:setFillColor( 1, 0, 0 )
```

- display.remove()

It is used to remove a specified object.

Syntax:
display.remove(object)
where object: specifies the display group which is to be removed.

-display.loadRemoteImage()

It is used to return a display object containing the image and also saves the image in the mentioned file.

Syntax:
display.loadRemoteImage(url, method, listener [, parameters], file_name [, base_directory] [, x, y])
where url: request url
method: HTTP method for instance: GET (default) or PUT.
listener: listener function to be invoked.
parameters: It includes HTTP request or response parameters which supports keys such as: header, body, response, etc.
file_name: name of the file where HTTP response is to be saved.
base_directory: name of the directory where file is to be saved.
x and y: specifies the x and y coordinates to place the created object.

Eg:

```
display.loadRemoteImage( "http://coronalabs.com/images/coronalogogrey.png"
, "GET", networkListener, "logo.png", system.DocumentDirectory, 20, 20 )
```

-display.newGroup()

It is used to create a display object for organizing objects and to further include children in a group.

Syntax:
display.newGroup()
Eg:
local rect = display.newRect(0, 0, 100, 100)
rect:setFillColor(0.5)

local group = display.newGroup()
group:insert(rect)
-display.newImage()

It is used to display an image on the screen from the specified file.

Syntax:
display.newImage([parent,] file_name [, base_directory] [, x, y])
where parent: specifies display group in which image is to be inserted, if required.
file_name: name of the image file which is to be loaded.
base_directory: name of the directory where file is located.
x and y: specifies the x and y coordinates of the image.

Eg:
local myImage = display.newImage("image.png")
myImage:translate(100, 100)

5. Native Library (1 hour)

These libraries contain files that are specific to operating system platform and are controlled by the same, not by the corona engine. You need to manually remove a native text box. These objects are on the top of the hierarchy and pops up on the front screen, thus hiding the display object.

5.1 Functions

It allows you to access native features available on your device.

-native.newTextBox()

Used to create a scrollable text box on which user can type multiple lines. In order to remove this native text box you need to pass, 'object: removeSelf()'.

Syntax:
native.newTextBox(x, y, width, height)

where x and y: the coordinates of the text box.
Width and height: width and height of the text box.

Eg...

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration=============================
W = display.contentWidth
H = display.contentHeight
local textBox
local function textListener( event )
if ( event.phase == "began" ) then
-- User begins editing "textField"
```

```
elseif ( event.phase == "ended" or event.phase == "submitted" ) then
-- Output resulting text from "textField"
print( event.target.text )
elseif ( event.phase == "editing" ) then
print( event.newCharacters )
print( event.oldText )
print( event.startPosition )
print( event.text )
end
end
--
local myText = display.newText( "Text Box Example", W*.5, H*.02,
native.systemFont, 28 )
-- Create text field
textBox = native.newTextBox( W*.5, H*.35, W*.8, H*.5 )-----X, Y, Width, Height
textBox:addEventListener( "userInput", textListener )
textBox.inputType = "default"------------Keyboard Types = number, decimal,
phone, url, email, no-emoji
textBox.isEditable = true
textBox.size = 18--------------------Set Size to text
textBox.font = native.newFont( native.systemFontBold, 18 )-------------------Set
Size and font to text
```

-native.newTextField()
Used to create a text field on which user type text in single line. In order to remove this native text field you need to pass, 'object: removeSelf()'.

Syntax:
native.newTextField(x, y, width, height)
where x and y: the coordinates of the text box.
Width and height: width and height of the text box.

Eg..

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
display.setStatusBar( display.HiddenStatusBar )

--======Variables Device resolution=================================
W = display.contentWidth
H = display.contentHeight


local textField
local function textListener( event )
if ( event.phase == "began" ) then
-- User begins editing "textField"
elseif ( event.phase == "ended" or event.phase == "submitted" ) then
-- Output resulting text from "textField"
print( event.target.text )
elseif ( event.phase == "editing" ) then
print( event.newCharacters )
print( event.oldText )
print( event.startPosition )
print( event.text )
end
end
--
local myText = display.newText( "Text Field Example", W*.5, H*.1,
native.systemFont, 28 )
-- Create text field
textField = native.newTextField( W*.5, H*.2, W*.5, H*.07 )-----X, Y, Width, Height
textField:addEventListener( "userInput", textListener )
textField.inputType = "default"------------Keyboard Types = number, decimal,
phone, url, email, no-emoji

-native.newMapView()
```

It is used to load a map view within a certain boundary limit in your device.

Syntax:
native.newMapView(x, y, width, height)

where x and y: the coordinates of the map view with respect to its center.
Width and height: width and height of the map view object.

Eg:..

```
--
============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--======Variables Declaration for device resolution===========
W = display.contentWidth
H = display.contentHeight
--
local function mapLocationListener( event )
    print( "The tapped location is in: " .. event.latitude .. ", " .. event.longitude )
end
--
local myText = display.newText( "WebView Example", W*.5, H*.03,
native.systemFont, 28 )
--
local mapView = native.newMapView( W*.5, H*.53, W, H*.9 )
mapView.x = display.contentCenterX
mapView.y = display.contentCenterY

-- Display map as vector drawings of streets (other options are "satellite" and
"hybrid")
mapView.mapType = "standard"
-- Initialize map to a real location
mapView:addEventListener( "mapLocation", mapLocationListener )
--
```

```
local currentLocation = mapView:getUserLocation()
mapView:setCenter( currentLocation.latitude, currentLocation.longitude )
mapView:addMarker( currentLocation.latitude, currentLocation.longitude )
```

-native.showWebPopup()
Used to load a web page which could be either local or remote. It is used to display only a single web popup. Incase of multiple web views, you need to use native.newWebView() (covered in the next topic).

Syntax:
native.showWebPopup([x, y, width, height,] url, [options])
where x and y: the coordinates with respect to the left and top edge of the web popup.
Width and height: width and height of the web popup.

Eg:..

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration for device
resolution=============================
W = display.contentWidth
H = display.contentHeight
local function listener( event )
local shouldLoad = true
local url = event.url
if 1 == string.find( url, "corona:close" ) then
-- Close the web popup
shouldLoad = false
end
if event.errorCode then
-- Error loading page
```

```
print( "Error: " .. tostring( event.errorMessage ) )
shouldLoad = false
end
return shouldLoad
end
local options =
{
hasBackground = false,
baseUrl = system.DocumentsDirectory,
urlRequest = listener
}
native.showWebPopup( 50, 100, 220, 300, "localpage1.html", options )
```

- native.newWebView()

Used to load web contents in a mobile app for instance: a html table in a scene. In Android, if web pages are to loaded from the internet then it is must to provide internet permission on 'build.settings' file.

Syntax:
native.newWebView(x, y, width, height)
where x and y: the coordinates of the web view with respect to its center.
Width and height: width and height of the web view object.

Eg :

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
display.setStatusBar( display.HiddenStatusBar )
--======Variables Device resolution==============================

W = display.contentWidth
H = display.contentHeight
local webView, crtWebView
```

--======Listener Defination==========================

```
local function webListener( event )
print("Prints event name.",event.name)
print("Prints event type.",event.type)
print("Prints url.",event.url)
print("Prints error code if error is Generated.",event.errorCode)
print("Prints error Message if error is Generated.",event.errorMessage)
end
--
local myText = display.newText( "WebView Example", W*.5, H*.03,
native.systemFont, 28 )
--
crtWebView = function()
webView = native.newWebView( W*.5, H*.53, W, H*.9 )
webView:request( "http://www.coronalabs.com/" )----------Enter the the
URL to Show Web View For Requested URL
webView:addEventListener( "urlRequest", webListener )---------Listener for web
View
end

crtWebView()
```

-native.setKeyboardFocus()

It is used to set keyboard focus on native.newTextBox() or native.newTextField() and to customize the visibility of the keyboard (i.e hides or shows the keyboard) as per the requirement.

Syntax:
native.setKeyboardFocus(textField)
where textField: ID of the targeted text field on which you need to set the keyboard focus or hide (by passing nil) the keyboard.

Eg..
---==========

```
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration==============================
W = display.contentWidth
H = display.contentHeight

local textField1
local textField2
--
local function textListener( event )
if ( event.phase == "began" ) then
-- User begins editing "defaultField"
if string.len( textField1.text ) > 5 then
native.setKeyboardFocus(textField2)
end
elseif ( event.phase == "ended" or event.phase == "submitted" ) then
-- Output resulting text from "defaultField"
elseif ( event.phase == "editing" ) then
if string.len( textField1.text ) > 5 then
native.setKeyboardFocus(textField2)
end
end
end
-- Function to handle button events
local function handleButtonEvent( event )
if ( "ended" == event.phase ) then
native.setKeyboardFocus(nil)
end
end

--
```

```
local myText = display.newText( "KeyBoard Focus Example", W*.5, H*.1,
native.systemFont, 28 )

--
textField1 = native.newTextField( W*.5, H*.2, W*.5, H*.05 )
textField1:addEventListener( "userInput", textListener )
--
textField2 = native.newTextField( W*.5, H*.3, W*.5, H*.05 )
--
-- Create the widget
local button1 = widget.newButton(
{
x = W*.5,
y = H*.4,
id = "button1",
label = "Click Here invisible KeyBoard",
shape = "roundedRect",
width = W*.65,
height = H*.06,
cornerRadius = 2,
onEvent = handleButtonEvent
}
)
```

-native.showPopup()

It is used to display a default pop up window as directed by operating system for a particular service.

Syntax: native.showPopup(name)
where name: name of the pop up which is to be reflected, for instance: requestAppPermission (pop up for requesting an app permission), mail (pop up for composing a mail, sms (pop up for composing a sms), social (pop up regarding social plugins), etc

Eg: ..

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration=============================
W = display.contentWidth
H = display.contentHeight
-- Function to handle button events
local function handleButtonEvent( event )
if ( "ended" == event.phase ) then
local options =
{
body = "This Popup Example"
}
native.showPopup( "sms", options )
end
end
--
local myText = display.newText( "Show Popup Example", W*.5, H*.1,
native.systemFont, 28 )
-- Create the widget
local button1 = widget.newButton(
{
x = W*.5,
y = H*.2,
id = "button1",
label = "Click Here open Popup",
shape = "roundedRect",
width = W*.6,
height = H*.1,
cornerRadius = 2,
onEvent = handleButtonEvent
```

}
)
- native.showAlert()

Used to create a pop up alert box consisting of one or more options. Programming functionalities and activities will continue in the background but user activity is put to halt. In order to resume the flow a user needs to provide an input by selecting a provided option or by cancelling the dialogue box.

Syntax:
native.showAlert(title, message [, buttonLabels [, listener]])
where title: title of the alert.
message: message to be displayed in the alert.
buttonLabels: buttons on the alert, from which a user is suppose to make a selection. In most cases two buttons are provided, Ok and Cancel.
listener: listener function is invoked after a user has provided an input.

Eg..

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration=============================
W = display.contentWidth
H = display.contentHeight

local showAlert

-- Handler that gets notified when the alert closes
local function onComplete( event )
if ( event.action == "clicked" ) then
local i = event.index
if ( i == 1 ) then----------Yes----First button was clicked
native.requestExit()--------requestExit is works only for Android
```

```
elseif ( i == 2 ) then----------No----second button was clicked
end
end
end
--

-- Function to handle button events
local function handleButtonEvent( event )
if ( "ended" == event.phase ) then
showAlert = native.showAlert( "Request", "Exit", { "Yes", "No" },
onComplete )---------Open Alert
end
end
--

local myText = display.newText( "Show Alert Example", W*.5, H*.2,
native.systemFont, 28 )
local myText1 = display.newText( "Request Exit Example", W*.5, H*.3,
native.systemFont, 28 )

-- Create the widget
local button1 = widget.newButton(
{
x = W*.5,
y = H*.1,
id = "button1",
label = "Exit",
shape = "roundedRect",
width = W*.3,
height = H*.05,
cornerRadius = 2,
onEvent = handleButtonEvent
}
)
```

-native.newFont()

Used to create a font object in which you can specify the native text fields, native text boxes or text display objects.

Syntax:
native.newFont(name [, size])
where name: declare the name or type of font to be used.
size: size of the font to be used.
Eg:
textBox.font = native.newFont(native.systemFontBold, 18)----------Set Size and font to text

6. String Library (30 minutes)

Strings are a sequence of characters. Corona platform provides a string library containing several functions to perform operations on string such as trimming, concatenating, pattern matching, comparing, replacing strings, etc. Below is a list of widely used functions.

6.1 Functions

- string.find()
It will look for the first occurrence of a pattern in a string and returns start and end indices of the first and last characters that matched the string (str). If passed pattern is not present, it will return nil.

Syntax:

string.find(str, pattern)

where str: String provided.
Pattern: pattern which is to be matched.

Eg:
1.) print(string.find(“Learn Corona easily”, “Corona”)

Output is => 7 12

2.) print(string.find(“Learn Corona easily”, “hard”)
Output is => nil

– string.len()
It will return the count of the characters present in the passed string.

Syntax:
string.len(str)
where str: string passed

Eg:
print (string.len(“Corona”))

Output is => 6

– string.reverse()
Used to reverse the passed string.

Syntax:
string.reverse(str)
Where str: string passed

Eg:
print (string.reverse (“Lua”))

output is => uaL

- string.lower()
It changes all the uppercase characters of a passed string to lowercase.

Syntax:
string.lower(str)
Where str: string passed

Eg:
print (string.lower (“LeArn CorOna”))

output would be => learn corona

– string.ends()
It is used to check whether the passed string ends with the provided pattern or not. Accordingly it returns a boolean, true or false.

Syntax:
string.ends(str, pattern)
where str: passed string
pattern: String containing the pattern to be checked

Eg:

1.
2. print (string.ends(“Corona”, “ona”))

Output would be => true

2.) print (string.ends(“Corona”, “on”))
Output is => false

– string.upper()
It changes all the lowercase characters of a passed string to uppercase.

Syntax:
string.upper(str)
Where str: string passed

Eg:
print (string.upper (“leaRn CorOna”))

output is => LEARN CORONA

– string.sub()
Used to return a substring [i.e part of a passed string within the specified index (inclusive)].

Syntax:
string.sub(str, init, end))
where str: passed string
init: index of the start of the substring

end: index of the end of the substring. By default it considers the end of the string.

Eg:

1.)
print(string.sub(“Learn Corona easily”, 7, 10))
Output is => Coro

2.)
local str = “Learn Corona easily”
print(str:sub(7))
print(str:sub(7, 10))
print(str:sub(-13, -8))
Output is =>Corona User
=> Coro
=> Corona
--string.gsub()
It is used to substitute the replacement string for all the occurrences of the pattern inside the passed string.

Syntax:
→ string.gsub(str,pattern, repl) or
→ string.gsub(str,pattern, repl, num)

where
str: passed string
pattern: string pattern to be matched
replacement: String which is used for replacement.
num: is an additional parameter which can be added to limit the number of substitutions.

Eg:
1.)

```
print(string.gsub("Corona is hard", "hard", "easy"))
```

Output
would be => Corona is easy

2.)

```
print (string.gsub("Corona is very easy to learn", "e","A", 2))
```

Output
would be => Corona is vAry Aasy to learn

In
example two, first two (num=2) occurrences of 'e' are replaced by A.

```
print("***---String Operation---***")
print( "string.byte( \"ABCDE\", 3, 5 ) - ",string.byte( "ABCDE", 3, 5 ) )
print( "string.char( 65,66,67 ) - ",string.char( 65,66,67 ) )
print( "string.ends( \"12345678\", \"56789\" ) - ",string.ends( "12345678",
"56789" ) )
print( "string.ends( \"Test123\", \"123\" ) - ",string.ends( "Test123", "123" ) )
print( "string.find( \"Hello Corona user\", \"Corona\" ) - ",string.find( "Hello
Corona user", "Corona" ) )
print( "string.format( \"%s %q\", \"Hello\", \"Corona user!\" ) -
",string.format( "%s %q", "Hello", "Corona user!" ) )
print( "string.gsub( \"banana\", \"a\", \"A\", 2 ) - ",string.gsub( "banana", "a", "A",
2 ) )
print("string.len( \"Lua\" ) - ",string.len( "Lua" ) )
print( "string.lower( \"Hey HeLlO WoRlD!\" ) - ",string.lower( "Hey HeLlO
WoRlD!" ) )
print( "string.match( \"I have 2 questions for you.\", \"%d+ %a+\" ) -
",string.match( "I have 2 questions for you.", "%d+ %a+" ) )
print( "string.rep( \"Corona \", 5 )",string.rep( "Corona ", 5 ) )
print( "string.reverse( \"Corona\" ) - ",string.reverse( "Corona" ) )
print( "string.starts( \"Test123\", \"est\" ) - ",string.starts( "Test123", "est" ) )
```

```
print( "string.sub( \"Hello Corona user\", 7, 12) - ",string.sub( "Hello Corona user", 7, 12 ) )
print( "string.upper( \"Hello, Corona user!\" )",string.upper( "Hello, Corona user!" ) )

Output=>
***---String Operation---***
string.byte( "ABCDE", 3, 5 ) -   67      68      69
string.char( 65,66,67 ) -        ABC
string.ends( "12345678", "56789" ) -        false
string.ends( "Test123", "123" ) -       true
string.find( "Hello Corona user", "Corona" ) -      7       12
string.format( "%s %q", "Hello", "Corona user!" ) -       Hello "Corona user!"
string.gsub( "banana", "a", "A", 2 ) -   bAnAna      2
string.len( "Lua" ) -       3
string.lower( "Hey HeLlO WoRlD!" ) -        hey hello world!
string.match( "I have 2 questions for you.", "%d+ %a+" ) -     2 questions
string.rep( "Corona ", 5 )       Corona Corona Corona Corona Corona
string.reverse( "Corona" ) -     anoroC
string.starts( "Test123", "est" ) -       false
string.sub( "Hello Corona user", 7, 12) -    Corona
string.upper( "Hello, Corona user!" )       HELLO, CORONA USER!
```

7. Math Library (30 minutes)

Corona provides certain math functions for performing basic numeric operations such as square root, trigonometric functions, elementary exponential, etc. Below listed are several widely used functions.

7.1 Functions

-math.abs()
It is used get the absolute (I.e magnitude) of the passed number.

Syntax: math.abs(num)
Where num: passed num

Eg: print("Absolute value",math.abs(-10))
--The output will be:- Absolute value 10

– math.ceil()
It gives the smallest integer that is greater than or equal to the passed number.

Syntax: math.ceil(num)
where num: passed number

Eg:
Print (math.ceil(-2.5))
Output is => -2
Print (math.ceil(5.5))
Output is => 6

– math.floor()

It gives the largest integer that is less than or equal to the provided number.

Eg:
Print (math.floor(-2.5))

Print (math.floor(5.5))

Output is => -3
=> 5
– math.exp()
It returns the base of the natural logarithms (e) to the power of the passed parameter.
Syntax:

math.exp (x)
where x: considered as an exponent.

Eg:
print(math.exp (0))
print(math.exp (2))

Output is => 1 (e^0)
=> 7.3890560... (e^2)

– math.min()

It returns the smallest of all the passed values.

Syntax:
math.min(x, y, ...)
where x, y: passed arguments

Eg:
print(math.min(2, 2.5 , -5))

output is => -5

– math.pow()
It returns the value of the first passed parameter (base) to the power of the second passed parameter (exponent).

Syntax:
math.pow(x, y)
where x: considered as a base
y: considered as an exponent.

It is treated as x^y.

Eg:

print(math.pow (20,0))
print(math.pow (2,3))
Output is => 1 (20^0)
=> 8 (2^3)

– math.random()
It returns a random number between the specified range.

Syntax:
math.random() // returns a number between 0 and 1.
math.random(p) // returns a number between 1 and p (inclusive).
math.random(p,q) // returns a number between p and q (inclusive).

where p, q: numbers which defines the range.

Eg:
print(math.random()) // output would be a number between 0 and 1
Output => 0.47033780881685
print(math.random(5)) /// output would be a number between 1 and 5 (inclusive)
Output => 4
print(math.random(10, 30)) // output would be a number between 10 and 30 (inclusive).
Output => 27

8. Widgets (30 minutes)

Widgets can be assumed as efficient tools used to enhance the productivity of the app and makes it more user friendly. Following are few widgets which add flexibility to an application:

8.1 Table View:

It is used to display data in different rows presented in tabular format as per our requirement. It is mostly used to display data in a listed format.

TableViewWidget objects (inherited from Group Object) are created using widget.newTableView(). It creates an object containing mask which restricts its view to a specified height and width.

widget.newTableView(): It creates a TableViewWidget object.
Syntax: widget.newTableView(options)
where options: Above function will accept a single argument options, is a Lua table that can accept certain parameters, few are listed below:

x,y: Co-ordinates of the widget with respect to its center point.

left, top: Left and top position where widget is to be created. These values can override x and y parameters.
height, width: specifies the total height and total width of the table view. Default values are width and height of the screen.

MaxVelocity: specifies the maximum scrolling speed of the table view. Default values is 2.

8.1.1 Visual Options

To customize the appearance of table view widget parameters can be passed in the 'options' table as per the desired effects needed. Few are listed below:

backgroundcolor: Table specifying the RGB+A (red, blue, green and alpha channels) background color setting. Default is white.

hideBackground: To keep the background of the table view hidden, this value is set to true, but it still receives touches.

hideScrollBar: To hide the scroll bar of the table view, this value is set to true.

8.1.2 Method

object:insertRow():
Used to inserts rows into TableViewWidget.
Syntax: object:insertRow(options)

object:deleteRows():
Used to delete rows present inside TableViewWidget.
Syntax: object:deleteRows(rowArray [, animationOptions])
where rowArray: Lua array of row indices to delete, for instance {2, 3, 4}.
animationOptions: optional table of time animation preferences for the deletion effect. It can take two parameters,slideLeftTransitionTime (in milliseconds) and slideUpTransitionTime (in milliseconds).

object:getNumRows(): returns the total number of rows present in the TableViewWidget.
Syntax: object:getNumRows()

Example of Basic Table View

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require("widget")
W = display.contentWidth
H = display.contentHeight
```

```
local myData = {}
myData[1] = { photo="x.png", name="XYZ",phone = "9890345647" }
myData[2] = { photo="a.jpg", name="XYZ",phone = "9890345647" }
myData[3] = { photo="b.jpg", name="XYZ",phone = "9890345647" }
myData[4] = { photo="x.png", name="XYZ",phone = "9890345647" }
myData[5] = { photo="a.jpg", name="XYZ",phone = "9890345647" }
myData[6] = { photo="b.jpg", name="XYZ",phone = "9890345647" }
myData[7] = { photo="x.png", name="XYZ",phone = "9890345647" }
myData[8] = { photo="a.jpg", name="XYZ",phone = "9890345647" }
myData[9] = { photo="b.jpg", name="XYZ",phone = "9890345647" }

local function onRowRender(event)
local row = event.row
local id = row.index
local rowHeight = row.contentHeight
local rowWidth = row.contentWidth
local photo = display.newImageRect(row,
myData[row.index].photo,W*0.2,W*0.2)

photo.anchorX = 0
photo.anchorY = 0.5

photo.x = rowWidth*0.05
photo.y =rowHeight*0.5

nameText = display.newText(row,myData[row.index].name, 12, 0,
native.systemFont, 50 )
nameText.anchorX = 0
nameText.anchorY = 0
nameText:setFillColor( 1,0,0 )
nameText.x = photo.contentBounds.xMax
nameText.y =rowHeight*0.1
```

```
phoneText = display.newText(row,myData[row.index].phone, 12, 0,
native.systemFont, 50 )
phoneText.anchorX = 0
phoneText.anchorY = 0
phoneText:setFillColor( 1,0,0 )
phoneText.x = photo.contentBounds.xMax
phoneText.y =nameText.y+ nameText.height
end

local function onRowTouch(event)
local row = event.target.index
if event.phase == "tap" then
                print("row no"..row)
elseif event.phase == "press" then
                 print("press")
        elseif event.phase == "release" then
         print("release")
end
end

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)

local title = display.newText( "Table View Example",W*.5, H*0.1,
native.systemFont, W*0.07 )
local tableView = widget.newTableView({
left = W*0.15,
top = title.y + H*0.2,
height = H*0.5,
width = W*0.7,
isBounceEnabled = false,
onRowRender = onRowRender,
onRowTouch = onRowTouch
})
```

```
for i=1,#myData do

        local isCategory = false
        local rowHeight = H*0.15
        local rowColor = {default = {176/255,226/255,255/255}, over = {
        141/255,182/255,205/255,0.2}}
        local lineColor = { 0.5, 0.5, 0.5 }
tableView:insertRow(
{
isCategory = isCategory,
rowHeight = rowHeight,
rowColor = rowColor,
lineColor = lineColor
})
end

local function tableViewReload()
tableView:reloadData()
end
```

8.2 Scrollview (30 minutes)

It is implemented to display large content in tabular format in mobile devices where we can also scroll the content. It is brought into use to achieve effective and smooth custom design as per our requirement.

ScrollViewWidget objects (inherited from Group Object) are created using widget.newScrollView(). It creates an object containing mask which restricts its view to a specified height and width.

widget.newScrollView():
It creates a ScrollViewWidget object.
Syntax: widget.newScrollView(options)
where options: Above function will accept a single argument options, is a Lua table that can accept certain parameters, few are listed below:

x,y: Co-ordinates of the widget with respect to its center point.

left, top: Left and top position where widget is to be created. These values can override x and y parameters.

height, width: specifies the total height and total width of the table view.
MaxVelocity: specifies the maximum scrolling speed of the table view. Default values is 2.

8.2.1 Visual Options

To customize the appearance of scrollview widget parameters can be passed in the 'options' table as per the desired effects needed. Few are listed below:

backgroundcolor: Table specifying the RGB+A (red, blue, green and alpha channels) background color setting. Default is white.

hideBackground: To keep the background of the tableview hidden, this value is set to true, but it still receives touches.

hideScrollBar: To hide the scroll bar of the table view, this value is set to true.

8.2.2 Method

-getView()
Used to return a reference to the object of Scrollview Widget.
Syntax:
object:getView()

-scrollToPosition()
Used to scroll to a particular position with respect to x and y.
Syntax:
object:scrollToPosition(options)
where options: includes several parameters for instance x (when you intend to scroll horizontally only) or y (when you intend to scroll vertically).

-object:setScrollHeight()
It redefines the content scrollable height, user is permitted to scroll within these limits.
Syntax:
object:setScrollHeight(newHeight)
where newHeight: new specified height within which user can scroll vertically.

-object:setScrollWidth()
It redefines the content scrollable width, user is permitted to scroll within these limits.
Syntax:
object:setScrollWidth(newWidth)
where newWidth: new specified width within which user can scroll horizontally.

Eg:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
```

```
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration for device
resolution============================
W = display.contentWidth
H = display.contentHeight

-- Create the scrollView
local scrollView = widget.newScrollView({
top = 10,
left = 10,
width = display.contentWidth - 20,
height = display.contentHeight - 20,
scrollWidth =600,
scrollHeight = 700,
horizontalScrollDisabled = true,
bottomPadding = 50
})

local title = display.newText( "The eagle who lived as a chicken",W*.53, H*0.14,
W*0.99, H*0.2, native.systemFontBold, W*0.059 )
title:setFillColor( 0 )
scrollView:insert(title)

local multiLineText = "A baby eagle became orphaned when something happened
to his parents. He glided down to the ground from his nest but was not yet able
to fly. A man picked him up. The man took him to a farmer and said, “This is a
special kind of barnyard chicken that will grow up big.” The farmer said, “Don’t
look like no barnyard chicken to me.” “Oh yes, it is. You will be glad to own it.”
The farmer took the baby eagle and placed it with his chickens. The baby eagle
learned to imitate the chickens. He could scratch the ground for grubs and
worms too. He grew up thinking he was a chicken. Then one day an eagle flew
over the barnyard. The eagle looked up and wondered,“What kind of animal is
that? How graceful, powerful, and free it is.” Then he asked another chicken,
“What is that?” The chicken replied, “Oh, that is an eagle. But don’t worry
```

yourself about that.You will never be able to fly like that." And the eagle went back to scratching the ground. He continued to behave like the chicken he thought he was. Finally he died, never knowing the grand life that could have been his."

```
local str =
{
text = multiLineText,
x = W*0.5,
y = title.y + H*0.01,
width = W*0.9,
font = native.systemFont,
fontSize = W*0.05,
align = "center"

}
local txt = display.newText(str )
-- local txt = display.newText( multiLineText,W*.5, title.y + H*0.1, W*0.8, 0,
native.systemFont, W*0.05 )
txt.anchorY = 0
txt:setFillColor( 0)
-- scrollView:insert(txt)

scrollView:insert(txt)
```

8.3 Button (20 minutes)

It creates a buttonWidget object.
- widget.newButton()
Syntax:
widget.newButton(options)
where options: is in the form of table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), onPress (an additional function could be called when button is pressed), onRelease (an additional function could be called when button is released), etc.

8.3.1 Methods

Methods are used to add functionality in order to set or to get the label of the button and also to enable or disable the touch, as following:

-object:setLabel(label)
Used to set or reset the text that appears on the button.
where label: represents the label to be displayed on the button.

-object:getLabel()
Used to return the text on the button which is in the form of string.

-object:setEnabled(isEnabled)
Used to enable or disable the buttonWidget.

where isEnabled: to enable (set to true) or disable (set to false) touch event on the button.

8.3.2 Basic Visual Options

To customize the label, font, fontSize, colour, alignment, etc. of the buttonWidget, you need to pass the desired parameter in the 'options' table.

8.3.3 Shape Construction

Corona gives you the provision to select the desired shape of the button from the available shapes for example “rect”, “roundedRect”,”circle”.

Syntax:

```
local button1 = widget.newButton( {
label = "button",
onEvent = listner,
shape = "roundedRect",
cornerRadius = 3,
width = 300,
height = 70,
fillColor = { default={1,1,1,1}, over={0.5,0.5,0.5,0.4} },
strokeColor = { default={0,0,0,1}, over={0.5,0.5,0.5,1} },
strokeWidth = 2 }
)
```

Eg:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================\
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--=======Variables Declaration for device
resolution==============================
W = display.contentWidth
H = display.contentHeight

local widget = require( "widget" )
```

```
local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)

local txt = display.newText( "Click the button",W*.72, H*0.14, W*0.9, 0,
native.systemFont, W*0.06 )

local smallRect = display.newRect(W*0.5,H*0.3,W*.3,H*.2)
smallRect:setFillColor(0,0,0)

-- Function to handle button events
local function handleButtonEvent( event )
if ( "ended" == event.phase ) then
smallRect:setFillColor(math.random(1,255)/255,math.random(1,255)/255,mat
h.random(1,255)/255)
end
end
-- Create the widget
local button = widget.newButton(
{
id = "button1",
label = "Button",
width = W*.25,
height = H*.1,
font = native.systemFont,
fontSize = W*0.08,
shape = "rect",
fillColor = { default={0.7,0.7,0.7,1}, over={0.9,0.9,0.9,1} },
strokeColor = { default={0.9,0.9,0.9,1}, over={0.7,0.7,0.7,1} },
onEvent = handleButtonEvent
}
)

button.x,button.y = W*0.5,H*0.5
```

8.4 PickerWheel (20 minutes)

It authorizes a user to either select or represent a data in a desired pattern for instance, circular or a column list.
--widget.newPickerWheel()

It creates a PickerWheel object.
Syntax:
widget.newPickerWheel(options)
where options: is in the form of table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), onValueSelected (lister function is called when the user selects an item in the picker wheel), etc.

8.4.1 Methods

-object:getValues()
Used to obtain the table of the corresponding selected index of the respective column.

Syntax:

object:getValues()

– object:selectValues()
Used to select a particular row within a respective column of the PickerWheelWidget.

Syntax:
object:selectValue(targetColumn, targetIndex [, snapToIndex])
where targetColumn: represents the targeted column from 1 to n (total number of columns)
targetIndex: represents the corresponding row to be selected within the targetColumn.

SnapToIndex: specified column+row selection will be selected (scrolling not needed).
Eg:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--======Variables Declaration for device resolution============================
W = display.contentWidth
H = display.contentHeight

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)

local txt = display.newText( "Select time",W*.7, H*0.1, W*0.9, 0, native.systemFont, W*0.05 )

-- Set up the picker wheel columns
local columnData =
{
{
align = "center",
width = W*0.25,
labelPadding = 20,
startIndex = 2,
labels = { "1","2","3","4","5","6","7","8","9","10","11","12" }
},
{
align = "center",
width = W*0.25,
labelPadding = 10,
startIndex = 1,
```

```
labels = { "01","02","03","04","05","06","07","08","09","10","11","12" }
},
{
align = "center",
width = W*0.25,
labelPadding = 10,
startIndex = 1,
labels = { "AM", "PM"}
}
}
-- Create the widget
local pickerWheel = widget.newPickerWheel(
{
columns = columnData,
style = "resizable",
width = 280,
rowHeight = W*0.1,
fontSize = W*0.03
})

pickerWheel.x,pickerWheel.y = W*0.5,H*0.45

local function handleButtonEvent( event )
if ( "ended" == event.phase ) then
local values = pickerWheel:getValues()
txt.text = "Selected time : "..values[1].value..":"..values[2].value.."
"..values[3].value
end
end

-- Create the widget
local button = widget.newButton(
{
id = "button1",
label = "SET",
```

```
width = W*.25,
height = H*.1,
font = native.systemFont,
fontSize = W*0.08,
shape = "rect",
fillColor = { default={0.7,0.7,0.7,1}, over={0.9,0.9,0.9,1} },
strokeColor = { default={0.9,0.9,0.9,1}, over={0.7,0.7,0.7,1} },
onEvent = handleButtonEvent
}
)
button.x,button.y = W*0.5,H*0.8
```

8.5 Switch (20 minutes)

Switches are basically used when a user has to select a state between on and off position.

- widget.newSwitch()

It is used to create a SwitchWidget object.

Syntax:
widget.newSwitch(options)
where options: is in the form of table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), onPress (an additional function could be called when switch is pressed), onRelease (an additional function could be called when switch is released), etc.

8.5.1 Methods

object:setState()
It is used to set the state of a SwitchWidget, achieved systematically via a program.

Syntax:
object:setState(options)
Where - options: is a table that can accept several parameters as: isOn [sets the switch to on (true) or off (false)], isAnimated (add animation effect at the time of state change, if true) and onComplete (sends a callback function acknowledging the state change.

8.5.2 Visual Customization

To customize the Radio or Checkbox of the SwitchWidget, you need to specify the two frames, frameOn (index number of the on frame with respect to radio/checkbox switch) and frameOff (index number of the off frame with respect to radio/checkbox switch) in the constructor of the respective widget.

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

local onOffSwitch,profession
local radioBtn,txt,chektxt,chekbxBtn = {},{},{},{}
local rect = display.newRect(W*.5,H*.5,W,H)

local nameTxt = display.newText( "Example of Switch",W*.7, H*0.14, W*0.8, 0,
native.systemFont, W*0.06 )
nameTxt:setFillColor(0.7,0.2,0.3)


local function saveBtnListnr(event)
if event.phase == "ended" then
local likes = {}
local hobbies = "My hobbies are"
for i=1,#radioBtn do
if radioBtn[i].isOn then
profession = radioBtn[i].id
end
end
for i=1,#chekbxBtn do
if chekbxBtn[i].isOn then
likes[#likes+1] = chekbxBtn[i].id
end
end
```

```
if #likes == 1 then
        hobbies = "My Hobby is"
end
for i=1,#likes do
tempstr = likes[i]
if i == 1 then
hobbies = hobbies.." "..tempstr
else
hobbies = hobbies..","..tempstr
end
end
print("Is sound on",onOffSwitch.isOn)
print("profession",profession)
print(hobbies)
end
end

local switchTxt = display.newText("Sound",W*.15,H*.22,system.font,W*.06)
switchTxt:setFillColor(0)

-- This is the example of onOff switch
onOffSwitch = widget.newSwitch(
{
left = W*.7,
top = H*.2,
style = "onOff",
id = "onOffSwitch",
onPress = onSwitchPress
}
)

-- This is the example of radio switch
local xpos = W*.16
local profsin = {"Developer","Analyst","Tester"}
local switchStat = {true,false,false}
```

```
for i=1,3 do
txt[#txt+1] = display.newText(profsin[i],xpos + W*.05,H*.45,system.font,W*.06)
txt[#txt]:setFillColor(0.2)
radioBtn[#radioBtn+1] = widget.newSwitch(
{
left = xpos,
top = H*.5,
style = "radio",
id = profsin[i],
initialSwitchState = switchStat[i],
onPress = RadioBtnlistenr
}
)
if i == 1 then radioBtn[#radioBtn]._view.initialSwitchState = true end
xpos = xpos + W*.3
end

-- This is the example of checkbox switch
local hobbies = {"Playing","Reading","Swimming"}
local switchStat = {true,false,true}
local xx = W*.16
for i=1,3 do
chektxt[#chektxt+1] = display.newText(hobbies[i],xx,H*.65,system.font,W*.06)
chektxt[#chektxt]:setFillColor(0.2)
chekbxBtn[#chekbxBtn+1] = widget.newSwitch(
{
left = xx,
top = H*.7,
style = "checkbox",
id = hobbies[i],
initialSwitchState = switchStat[i],
onPress = chekboxListnr
}
)
xx = xx + W*.3
```

```
end

local button = widget.newButton({
top = H*.85,
left = W*.5,
width = W*.15,
height = H*.08,
fontSize = W*.07,
id = "save",
label = "Save",
onEvent = saveBtnListnr
})
```

- The output will be:-
Is sound on false
profession Developer
My hobbies are playing, swimming

8.6 Stepper (15 minutes)

It is basically used when a user needs to increase or decrease a value within the maximum or minimum limits set.

- widget.newStepper()
It creates a StepperWidget object.
Syntax: widget.newStepper(options)
where options: is in the form of table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), initialValue (the value at which stepper begins), maximum Value (the maximum value that a stepper can have), onPress (an additional function could be called when stepper is pressed), minimum Value (the minimum value that a stepper can have) etc.

8.6.1 Methods

- object:getValue()

Used to get the present value of the StepperWidget.

Syntax:
object:getValue()

- object:setValue()
Used to set the value of a StepperWidget within the range of maximum and minimum values assigned.

Syntax:
object:setValue(value)

Where value: value to be set as the present value of the Stepper.
Eg:
---==========
-- Company: Redbytes Software Pvt. Ltd. Pune

```
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--======Variables Declaration==============================
W = display.contentWidth
H = display.contentHeight

local widget = require( "widget" )

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)

local smallRect = display.newRect(W*0.5,H*0.3,W*.3,H*.2)
smallRect:setFillColor(0,0,0)
local txt = display.newText( "0",smallRect.x, smallRect.y, native.systemFont,
W*0.1 )
-- Handle stepper events
local function onStepperPress( event )
if ( "increment" == event.phase ) then
smallRect.width,smallRect.height =
smallRect.contentWidth+W*0.01,smallRect.contentHeight+H*0.006
txt.text = txt.text +1
elseif ( "decrement" == event.phase ) then
smallRect.width,smallRect.height = smallRect.contentWidth-
W*0.01,smallRect.contentHeight-H*0.005
txt.text = txt.text -1
end
end
-- Create the widget
local newStepper = widget.newStepper(
{
left = W*0.43,
top = H*0.5,
minimumValue = 0,
```

```
maximumValue = 10,
onPress = onStepperPress
}
)
```

8.7 Slider (15 minutes)

Sliders are basically used to select a value from a range between 0 to 100 irrespective of its size and position.

-widget.newSlider()
It creates a SliderWidget object.

Syntax:
widget.newSlider(options)
where options: is in the form of a table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), width (total width of the slider), value (represents the initial value of the slider in terms of percentage), etc.

8.7.1 Method

-object:setValue()
Used to change the location of the handle of a SliderWidget.

Syntax:
object:setValue(percent)
where percent: location of the slider in terms of percentage.

8.7.2 Visual Customization

Customization of Slider widget can be achieved with the help of frames, considering outer border (left cap, middle span and left cap) and the inner fill area.

8.8 Progress View Widget (15 minutes)

ProgressView widget is basically used to indicate the progress or the status of a certain activity.

widget.newProgressView()

It creates a progressViewWidget object.

Syntax:
widget.newProgressView(options)

where options: is in the form of a table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), width (total width of the progressView), isAnimated (if true, animation effetcs are reflected), fillXOffset and fillYOffset (to set this to horizontally and vertically offset the fill region respectively) etc.

8.8.1 Methods

-object:setProgress(progress)

Used to set the progress value of a ProgressViewWidget.
where progress: the value at which progress is to be set (in terms of percentage).

-object:getProgress()

Used to obtain the progress value of a ProgressViewWidget with respect to current time.

- object:resizeView(newWidth)
Used to resize the width of the ProgressViewWidget.
where newWidth: new width of the progressViewWidget.

8.8.2 Visual Customization

Customization of progressView widget can be achieved with the help of frames, considering outer border (left cap, middle span and left cap) and the inner fill area.

Eg:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--=======Variables Declaration for device
resolution==============================
local W = display.contentWidth
local H = display.contentHeight

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)
local temp = 0
local point = 0.1
local title = display.newText( "Progress View Example",W*.5, H*0.1,
native.systemFont, W*0.07 )

local percentage = display.newText( "0",W*.5, title.y+H*0.2, native.systemFont,
W*0.07 )

-- Create the widget
local progressView = widget.newProgressView(
{
left = W*0.1,
top = H*0.5,
width = W*0.8,
```

```
isAnimated = true
}
)
local tmr = timer.performWithDelay(500,function()
        if point <= 1 then
                temp = temp + 10
                point = point + 0.1
                if tmr ~= nil then
                        timer.cancel( tmr )
                end
        end
        percentage.text = temp.."%"
        progressView:setProgress( point )
end,-1)
```

8.9 Tab Bar (15 minutes)

It is basically used to represent a menu alignment mostly located in the form of tabs in the extreme positions of the screen.

-widget.newTabBar()

It creates a TableBarWidget object.

Syntax:
widget.newTabBar(options)
where options: is in the form of a table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), width (total width of the tab bar), buttons (a table that includes the sub table of the respective tab bar), etc.

8.9.1 Method

-object:setSelected(buttonIndex)
Used to set a desired TableBarWidget button to its selected position.

where buttonIndex: buttonIndex on the table bar, starting from left.

Eg: tabBar:setSelected(2)
It will select the second tab from the left.

Eg:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration for device
resolution==============================
local W = display.contentWidth
```

```
local H = display.contentHeight

local widget = require( "widget" )
local startTransition,completeTransition

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)

local txt = display.newText( "Tab bar Example",W*.5, H*0.14, native.systemFont,
W*0.06 )


local smallRect = display.newRect(W*0.5,H*0.34,W*.3,H*.2)
smallRect:setFillColor(0,0,1)

function handleTabBarEvent( event )
	if event.target._id == "tab1" then
		smallRect:setFillColor(0,0,1)
	elseif event.target._id == "tab2" then
		smallRect:setFillColor(0,1,0)
	elseif event.target._id == "tab3" then
		smallRect:setFillColor(1,1,0)
	elseif event.target._id == "tab4" then
		smallRect:setFillColor(1,0,0)
	end
end

tabButtons = {
{
label = "Blue",
labelColor = { default={ 1, 0, 0 }, over={ 1, 0, 0, 0.5 } },
font = native.systemFont,
size = W*0.06,
id = "tab1",
selected = true,
```

```
onPress = handleTabBarEvent
},
{
label = "Green",
labelColor = { default={ 1, 0, 0 }, over={ 1, 0, 0, 0.5 } },
font = native.systemFont,
size = W*0.06,
id = "tab2",
onPress = handleTabBarEvent
},
{
label = "Yellow",
labelColor = { default={ 1, 0, 0 }, over={ 1, 0, 0, 0.5 } },
font = native.systemFont,
-- labelYOffset = -12,
size = W*0.06,
id = "tab3",
onPress = handleTabBarEvent
},
{
label = "Red",
labelColor = { default={ 1, 0, 0 }, over={ 1, 0, 0, 0.5 } },
font = native.systemFont,
size = W*0.06,
id = "tab4",
onPress = handleTabBarEvent
}
}

-- Create the widget
tabBar = widget.newTabBar(
{
top = H-H*0.06,
width = display.contentWidth,
height=H*0.06,
```

```
buttons = tabButtons
}
)
```

8.10 Spinner Widget (15 minutes)

Spinner widget can be though of a circular indicator that provides a user about the status of a certain activity in process. For instance, a page is being loaded and processor seems to be busy, it can be indicated using this particular widget.

-- widget.newSpinner()

It creates a SpinnerWidget object.

Syntax:
widget.newSpinner(options)
where options: is in the form of a table and can supports several parameters, such as x and y (coordinates), left and top (exact position where widget is to be created), width (total width of the spinner), etc.

8.10.1 Methods

-object:start()
Used to start the animation of a SpinnerWidget.

-object:stop()
Used to stop the animation of a SpinnerWidget.

Eg:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--======Variables Declaration for device
resolution==============================
local W = display.contentWidth
```

```
local H = display.contentHeight

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.4)

local title = display.newText( "Spinner Example",W*.5, H*0.1, native.systemFont,
W*0.07 )

local spinner = widget.newSpinner({
top = H*0.4,
left = W*.4,
                width = W*0.2,
                height = W*0.2,
                incrementEvery = 20
})

spinner:start()

timer.performWithDelay(2300,function()
        spinner:stop()
        local title2 = display.newText( "completed",W*.5, H*0.7, native.systemFont,
W*0.07 )
end)
```

9. Transition

Transitions (Animations) are used to give special effects in order to provide high quality feel to your mobile application. Transition library comprises of several functions and methods to make transition between display groups and display objects over a predefined time span. Transition effects include: fade, zoomIn, zoomOut, blink, scale, etc.

9.1 Functions:

It gives you a provision to pause, resume, or cancel a transition or all transition as per your convenience. Few are listed below:

-transition.cancel()
This particular function will abort the transition depending on the parameter passed. In similar manner transition.resume() and transition.pause() can be implemented for.

Syntax:
- transition.cancel()
- transition.cancel(transitionReference)
where transitionReference: particular transition to be cancelled.
- transition.cancel(displayObject)
where displayObject: all the transitions with respect to this display object would be aborted.

- transition.to()
It specifies the transition effects (rotate, fade, scale, etc.) to be used on display objects and also the time stamp for which it is to be performed.

Syntax:
transition.to(target, param)
where target: display object on which effect is to be performed
param: it includes a wide range of parameters such as time, iterations, delay, onResume, onCancel, x or y coordinate, size, etc.

Eg.:

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )

--======Variables Declaration for device
resolution=============================
local W = display.contentWidth
local H = display.contentHeight

local widget = require( "widget" )
local startTransition,completeTransition

local bgRect = display.newRect(W*0.5,H*0.5,W,H)
bgRect:setFillColor(0.5)

local circle = display.newCircle(W*0.2,H*0.2,W*0.1)
circle:setFillColor(1,0,0)

function completeTransition(circle)
circle.alpha = 1
transition.to(circle,{time=2000 ,x=W*0.1, alpha=0,
yScale=1,onComplete=startTransition})
end

function startTransition(circle)
circle.alpha = 1
transition.to(circle,{time=2000 ,x=W, alpha=0,
yScale=1,onComplete=completeTransition})
end

startTransition(circle)
```

9.2 Convenience Methods

It equips the developer to give animation effects and also the time stamp for which it is to be implemented. Several methods are as follows:

- transition.blink()
It gives blinking effect repeatedly to the object over a specified time limit.
Syntax:
transition.blink(target, params)
where target: display object on which effect is to be performed.
param: it includes a wide range of parameters such as time, iterations, delay (delay in milliseconds before transition is initiated), onResume (Listener function invoked when the transition is resumed.), onCancel (Listener function invoked when the transition is cancelled), etc.

In similar manner, following methods could be implemented:
- transition.fadeOut()
It is used to give fading effect to the object over a specified time period.

Syntax:
transition.fadeOut(target, params)

- transition.scaleBy()
It scales the object by specified x and y scale over a specified time period.

Syntax:
transition.scaleBy(target, param)

- transition.moveTo( )

It moves the object to the specified x and y coordinates over a specified time period.
Syntax:
transition.moveTo(target, param)

```
local obj = display.newRoundedRect( 50, 50, 200, 200, 20 )

transition.moveTo( obj, { x=100, y=200, time=1000 } )
```

10. Timer (15 minutes)

When there is a requirement of calling a particular function after a predefined time interval, timers are put into use. Corona provides a Timer library which includes several basic functions and it also manages event listener.

10.1 Functions

Below listed are few basic timer functions:

1. timer.performWithDelay(): It will call a function after a delay of specified time. This function may also returns a table that can further include other timer functions.

Basic Syntax: timer.performWithDelay(delay, listener [, iterations])
where delay: Delay (in milliseconds) which should be less than the runtime framerate of the app.

Listener: Either a function listen or table listener to be invoked.

iterations: number of times function needs to be called. By default, it takes 1. To execute it for endless number of times, you need to pass 0 or -1.

Eg.: local function timerListener(event)
print("Corona SDK")
end
timer.performWithDelay(1000, timerListener)

After execution of these lines, listener will be invoked after a delay of 1000 milliseconds.

2. timer.cancel(): It will cancel a scheduled timer which was initiated with timer.performWithDelay() function.
Syntax: timer.cancel(timerID)

3. timer.pause(): It will pause the timer initiated by timer.performWithDelay().
Syntax: timer.pause(timerId)

4. timer.resume(): It will resume the scheduled timer that was paused by executing timer.pause().
Syntax: timer.resume(timerId)

11 Directories (20 minutes)

Depending on your specific requirement, it gives use the provision to store a data while accessing a particular app. Import directories to be used are briefed further.

11.1 Document Directory

It is preferred to store the permanent data to a file that has been generated by the app. Files present in this directory are removed, when the application present in your device is un-installed by the user.
The path is '/Documents'.

Syntax:
system.DocumentsDirectory

Eg.
local image = display.newImage("username.png",system.DocumentsDirectory)

11.2 Resource Directory

It is an integral directory used to store core files (including main.lua file). This acts as a default directory and is exposed to read access only (i.e neither it can be modified nor files can be added).

Syntax:
system.ResourceDirectory
Eg.
local image = display.newImage("username.png",system.ResourceDirectory)

11.3 Temporary Directory

It is used to store the files containing temporary data which needs to be kept until current session of an app is killed. Once the user closes the app, data is lost. The path is '/tmp'.
Syntax:
system.TemporaryDirectory

Eg.
local image = display.newImage("username.png",system.TemporaryDirectory)

12. File I/O (30 minutes)

File is a form of data for instance: images, audio files, etc related to a particular mobile application and is stored in a specific location (directory). It is must to add a functionality to a mobile app for reading or writing a file depending on its type. To implement this, you use I/O library provided by Lua.

-system.pathForFile()

It represents an absolute path with consideration to the directories on a standardized filesystem.

Syntax:
system.pathForFile(filename [, baseDirectory])
where filename: represents the name of the file (or the path to that particular file) which is relative to baseDirectory.
BaseDirectory: represents a constant with respect to base directory where a file is located. Default is system.ResourceDirectory.

12.1 Writing a file

It gives you the provision to write or make changes in a file which already exists. A new file gets created, if that particular file does not exist.

Below is a fully functioning example of how to write a file.

Eg.

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
data = {}
data.name = "Redbytes software"
data.address = "Pune."
```

```
local myData = json.encode(data)
local path = system.pathForFile( "userInput.txt" , system.DocumentsDirectory )
local file = io.open( path, "w" )
file:write(myData)
io.close(file)
```

12.2 Reading a file

To read a file, you need to pass the file path so that it can be opened and read further. Depending on the requirement and location of the file, it can be read from any of the four directories (covered previously). Below is a fully functioning example of how to read a file.

Eg.

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local myData={}
local path = system.pathForFile( "userInput.txt" , system.ResourceDirectory )
if path ~= nil then
local file = io.open( path, "r" )
if file then
data=file:read("*a")
myData=json.decode(data)
io.close(file)
end
print("Name",myData.name)
        print("Address",myData.address)
else
        print("Not available")
end
```

13. JSON (15 minutes)

JSON is language independent and stands for JavaScript Object Notation. Before starting JSON, it is good to have knowledge of OOPs concepts. This section will guide you, how to encode and decode JSON objects using Lua.
The json library enables serializing and transmitting structured data between web applications and server over the network.

13.1 Encoding JSON in Lua

Following is a simple example to encode JSON object into Lua tables.
Syntax:
json.encode(table)
Where table: Lua table with reference to JSON library.

Eg:
abc = {userName:abc, id:30}
encodedJson = json.encode(abc)

13.2 Decoding JSON in Lua

Following is a simple example to decode JSON encoded data into Lua table.

Synatx:
json.decode(data [, position [, null_value]])
where data: String having JSON data
position: index location to start decoding
null_value: used to know the location of the data which is null

Ex:
abc = {"userName":"abc","id":30}
decodedJson = json.decode(tableArr)

14. Audio Library (45 minutes)

Corona gives you the provision to access advanced OpenAL (Open Audio Library) features. OpenAL is a cross-platform 3D audio API (application programming interface) efficient to be used with gaming and other types of audio applications. Audio Library facilitates you to control sound elements and add sound effects to your app, to enhance its performance. It supports different formats (.wav, .mp3, .aif, .aac, etc.) depending on the requirement and the platform.

Below listed are several functions and properties which will improve the functionality of the app.

14.1 Functions

- audio.loadSound()

It loads the file which is to be play and return the handle.
audio.loadSound(audioName [,basDir])

where- audioName is the name of the file.
basDir is the directory where file is located.

Eg :- local snd = audio.loadSound("backgroundmusic.mp3")
It will load the sound named as “backgroundmusic.mp3” from resource directory

-audio.loadStream()

It loads a file which is to be read as streaming audio. It load file in chunks to minimize the memory use.

Syntax: audio.loadStream(audio_File [, baseDir])
where - audio_File is name of the file to be loaded.
baseDir is directory where file is located.
Default location of the audio file (System.ResourceDirectory).

Eg:- local snd = audio.loadStream("backgroundmusic.mp3")

-audio.play()

It plays the specific audio with respect to the mentioned audioHandle.
Syntax:
audio.play(audioData [, options])
where - audioData: It is the audio handle which is to be play.
options: includes listener,channel etc

Eg:- local sndChanl = audio.play(snd)

- audio.pause()

It is used to pause playback on a specified channel which are being played. If a channel is not specified, it will pause all the channels.
Syntax: audio.pause() // pauses all the active channels
audio.pause([channel]) // would pause the specified channel

where - channel: specifies the channel number.

Eg :- audio.pause(sndChanl)
-audio.resume()

It is used to resume playback on a channel which is in pause state.

Syntax: audio.resume([channel])
where- channel: specifies the channel number which is to be resumed.

Eg:- audio.resume(sndChanl)

-audio.dispose()

It is used to free out the memory with reference to channels.

Syntax:

audio.dispose(audio_channel)
where audio_channel: Audio channel or handle which is to be disposed or released.

- audio.getVolume()

It is used to get the volume for a particular channel or a master channel.

Syntax: audio.getVolume({ channel = chan })
where - chan: specifies the channel number for which you need the volume (currently channel range is 1 to 32).

Eg: audio.getVolume(sndChanl)

local masterChannelVolume = audio.getVolume() //gives you the volume of the master channel
local specificChannel1Volume = audio.getVolume({ channel=5}) // gives you the volume of the specific channel

- audio.setMaxVolume()

It is used to set a particular value as a maximum volume on a desired channel (currently, within the range of 1 to 32).

Syntax: audio.setMaxVolume(volume, channel)
where- volume: the new maximum value which is to be set(range between 0.0 to 1.0).
channel: channel number on which the volume is to be set.

Eg: audio.setMaxVolume(0.5, {channel=5})

- audio.stop()

It is used to stop the audio which are being played back and clear all the channels. It gives the number of channels that have been stopped (-1, in case of an error).

Syntax:
audio.stop([audio_channel])
where audio_channel: channel that is to be stopped. To stop all the channels, no parameter should be passed.

14.2 Properties

-audio.freeChannels
It returns the total number of available channels for playback i.e it excludes the occupied channels that are being played or paused.

Syntax:
audio.freeChannels

Eg: local chanlCnt = audio.freeChannels

- audio.totalChannel

It returns the total number of channels (currently 32 are available).

Syntax: audio.totalChannels

-audio.unreservedUsedChannels
It returns the number of channels that are being used with respect to current time (i.e channels being played or paused). It excludes the reserved channels.

Syntax:
audio.unreservedUsedChannels

Eg: local chanlCnt = audio.unreservedUsedChannels

---==========
-- Company: Redbytes Software Pvt. Ltd. Pune
--==========

```
W = display.contentWidth
H = display.contentHeight

local widget = require "widget"

local snd
local sndChanl
local btn = {}
local rect = display.newRect(W*.5,H*.5,W,H)

local txt = display.newText( "Example of Audio",W*.7, H*0.14, W*0.8, 0,
native.systemFont, W*0.06 )
txt:setFillColor(0.7,0.2,0.3)

local function setLblNId(obj,id)
obj:setLabel(id)
obj.id=id
end

local function btnListnr(event)
if event.phase == "ended" then
if event.target.id == "Play" then
setLblNId(event.target,"Pause")
snd = audio.loadSound("backgroundmusic.mp3")
sndChanl = audio.play(snd)
elseif event.target.id == "Pause" then
setLblNId(event.target,"Resume")
audio.pause(sndChanl)
elseif event.target.id == "Resume" then
setLblNId(event.target,"Pause")
audio.resume(sndChanl)
elseif event.target.id == "Stop" then
setLblNId(btn[1],"Play")
audio.stop(sndChanl)
sndChanl = nil;
```

```
audio.dispose( snd )
end
end
end

local xpos = W*.5 - W*.36
local ids = {"Play","Stop"}
for i=1,2 do
btn[#btn+1] = widget.newButton(
{
left = xpos,
top = H*.5,
width = W*.35,
height = H*.07,
id = ids[i],
label = ids[i],
shape = "roundedRect",
cornerRadius = 2,
fillColor = { default={0.5}, over={0.4} },
labelColor = { default={ 1 }, over={ 0 } },
strokeColor = { default={1,0.4,0,1}, over={0.8,0.8,1,1} },
strokeWidth = 4,
fontSize = W*.08,
onEvent = btnListnr
}
)
xpos = xpos + W*.4
end
```

15 Network Library (1 hour)

Corona provides a network library to simplify the process of networking operations for example connecting to an online service for retrieving data, etc.

15.1 Functions

- network.request()

It provides a protocol to handle HTTP or HTTPS (for instance: GET, POST, HEAD, DELETE, etc) request to a URL.

Syntax:
network.request(url, method, listener [, parameter])
where url: request url
method: HTTP method for instance: GET (default), HEAD, PUT, etc.
listener: listener function to be invoked.
Parameters: It includes HTTP request or response parameters which supports keys such as: header, body, response, etc.

Eg.

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration=============================
W = display.contentWidth
H = display.contentHeight
--
local myText = display.newText( "Network Request Example", W*.5, H*.2,
native.systemFont, 24 )
```

```
local status = display.newText( "Requesting Image..", W*.5, H*.8,
native.systemFont, 24 )
-- The following code demonstrates how to download a file, with progress
updates.
local function networkListener( event )
if ( event.isError ) then
print( "Network error: ", event.response )
elseif ( event.phase == "began" ) then
if ( event.bytesEstimated <= 0 ) then
print( "Download starting, size unknown" )
else
print( "Download starting, estimated size: " .. event.bytesEstimated )
end
elseif ( event.phase == "progress" ) then
if ( event.bytesEstimated <= 0 ) then
print( "Download progress: " .. event.bytesTransferred )
else
print( "Download progress: " .. event.bytesTransferred .. " of estimated: " ..
event.bytesEstimated )
end
elseif ( event.phase == "ended" ) then
print( "Download complete, total bytes transferred: " .. event.bytesTransferred )
local responseImg = display.newImageRect(event.response.filename,
event.response.baseDirectory, W*.5, W*.5 )
responseImg:translate(W*.5, H*.5)
status.text = "Request Image Complete"
end
end
local params = {}
-- Tell network.request() that we want the "began" and "progress" events:
params.progress = "download"
-- Tell network.request() that we want the output to go to a file:
params.response = {
filename = "corona.jpg",
baseDirectory = system.DocumentsDirectory
```

}
network.request("http://docs.coronalabs.com/images/simulator/image-mask-base2.png", "GET", networkListener, params)

-network.upload()
It is used to upload the response to a mentioned local file.
Syntax:
network.upload(url, method, listener [, parameter], file_name [, base_directory] [, content_type])
where url: request url
method: HTTP method for instance: GET (default), HEAD, PUT, etc.
listener: listener function to be invoked.
parameters: It includes HTTP request or response parameters which supports keys such as: header, body, response, etc.
file_name: name of the file where response is to be saved.
base_directory: name of the directory where response is to be saved. Default directory is system.DocumentsDirectory.

Eg.

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--======Variables Declaration=============================
W = display.contentWidth
H = display.contentHeight
--
local myText = display.newText( "Network Upload Example", W*.5, H*.2, native.systemFont, 24 )
local status = display.newText( "Uploading..", W*.5, H*.4, native.systemFont, 24 )
--
local function networkListener( event )
if event.phase == "ended" then
```

```
status.text = "Upload complete!"
end
end
--
network.upload(
"http://raficbawabco.com/api/event_list.php",
"POST",
networkListener,
"object.json",
system.DocumentsDirectory,
"application/json"
)
```

-network.download()
It is used to download the response to a mentioned local file.

Syntax:
network.download(url, method, listener [, parameters], file_name [, base_directory])
where url: request url
method: HTTP method for instance: GET (default), HEAD, PUT, etc.
listener: listener function to be invoked.
parameters: It includes HTTP request or response parameters which supports keys such as: header, body, response, etc.
file_name: name of the file where response is to be saved.
base_directory: name of the directory where response is to be saved. Default directory is system.DocumentsDirectory.

Eg.

```
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--============================================================
local widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
```

```
--==============Variables Declaration for device resolution============
W = display.contentWidth
H = display.contentHeight
--
local myText = display.newText( "Network Download Example", W*.5, H*.2,
native.systemFont, 24 )
local status = display.newText( "Downloading..", W*.5, H*.8, native.systemFont,
24 )

-- The following code demonstrates how to download a file, with progress
updates.
local function networkListener( event )
if ( event.isError ) then
print( "Network error: ", event.response )
elseif ( event.phase == "began" ) then
if ( event.bytesEstimated <= 0 ) then
print( "Download starting, size unknown" )
else
print( "Download starting, estimated size: " .. event.bytesEstimated )
end
elseif ( event.phase == "progress" ) then
if ( event.bytesEstimated <= 0 ) then
print( "Download progress: " .. event.bytesTransferred )
else
print( "Download progress: " .. event.bytesTransferred .. " of estimated: " ..
event.bytesEstimated )
end
elseif ( event.phase == "ended" ) then
print( "Download complete, total bytes transferred: " .. event.bytesTransferred )
local responseImg = display.newImageRect(event.response.filename,
event.response.baseDirectory, W*.5, W*.5 )
responseImg:translate(W*.5, H*.5)
status.text = "Download Complete"
end
end
```

```
local params = {}
params.progress = true
network.download(
--"http://docs.coronalabs.com/images/simulator/image-mask-base2.png",
"http://www.freeiconspng.com/uploads/png-los-minions-from-annnto-2.png",
"GET",
networkListener,
params,
"helloCopy.png",
system.TemporaryDirectory
)
-network.cancel()
It is used to cancel the request initiated before which could be
network.request(),  network.upload() or network.download().

Syntax:
network.cancel( request_Id )
where request_Id: Id of the request to be canceled.
Eg.
---============================================================
-- Company: Redbytes Software Pvt. Ltd. Pune
--
============================================================loca
l widget = require( "widget" )
display.setStatusBar( display.HiddenStatusBar )
--==========Variables Declaration for device resolution==================
W = display.contentWidth
H = display.contentHeight
local cancelButton
local myText = display.newText( "Network Cancel Example", W*.5, H*.2,
native.systemFont, 24 )
local status = display.newText( "Img Downloading..", W*.5, H*.75,
native.systemFont, 24 )
local cancelTxt = display.newText( "Download Cancel", W*.7, H*.9,
native.systemFont, 24 )
```

```
-- The following code demonstrates how to download a file, with progress
updates.
local function networkListener( event )
if ( event.phase == "ended" ) then
local responseImg = display.newImageRect(event.response.filename,
event.response.baseDirectory, W*.5, W*.5 )
responseImg:translate(W*.5, H*.5)
timer.performWithDelay( 1000,function()
os.remove( system.pathForFile( event.response.filename,
event.response.baseDirectory ) ) end)
status.text = "Download Complete"
cancelTxt.alpha = 0
cancelButton.alpha = 0
end
end
local params = {}
-- Tell network.request() that we want the "began" and "progress" events:
params.progress = "download"
-- Tell network.request() that we want the output to go to a file:
params.response = {
filename = "corona.jpg",
baseDirectory = system.DocumentsDirectory
}
local requestId =
network.request( "http://docs.coronalabs.com/images/simulator/image-mask-
base2.png", "GET", networkListener, params )
local onCancelButton = function( event )
network.cancel( requestId )
status.text = "Download Cancel"
end
-- Create a cancel button that can cancel the request:
cancelButton = display.newImage( "error.png" )
cancelButton:translate(W*.3, cancelTxt.y)
cancelButton:addEventListener( "tap", onCancelButton )
```

16. Physics (6 hours)

Corona facilitates you to add physics to develop gaming apps by giving you the privilege to use physics engine. Physics provides a simulation of physical system, such as soft body dynamics, collision or physical forces like gravity.

16.1 Physics Bodies

- Dynamic

Dynamic bodies are completely simulated and can collide with any type of body. Such bodies can also be moved manually in a code but function according to the forces like collision or gravity.

-Static

Static bodies does not react to any force, impulse, or collision and does not move. By default, every Box2D body is considered as a static body. A static body also does not collide with other static or kinematic bodies.

-Kinematic

Kinematic bodies are hybrid between static and dynamic bodies. Such bodies do not react to forces and collisions like a static body but can moved with a linear velocity like a dynamic body. A Kinematic body does not collide with other kinematic or static bodies.

16.2 Physics Engine

Physics engine gives effective graphics and replicates real world physics. Box2D is widely used open source two dimensional physics simulator that can simulate bodies composed of several shapes such as rectangles, polygons, etc. The engine also applies friction and also, allows bodies to be joined together with 'joints' and further acted upon by forces.

In order to initiate functioning of physics, you need to include library as follows:

local physics = require("physics")

16.2.1 Functions for Physics Setup

Below mentioned functions ensure the start, pause and stop of the physics simulation.

-physics.start()
It is used to start the physics simulation or resume the simulation (if it is paused).

-physics.pause()
It is used to pause the physics simulation.
-physics.stop()
It is used to stop the entire physics simulation.

16.3 Physics Simulation Methods

Physics engine introduces you to several features to enhance the smooth functioning of physics simulation.
Below are listed are few widely used simulation methods:

-physics.addBody()
It is used to change a corona display object into a simulated physical object.

Syntax:
physics.addBody(object, [body_Type,] [parameters])
where object: display object to which physical simulation is to be added.
body_Type: Either of the three body type (dynamic, static or Kinematic) mentioned in the previous topic.

Parameters: A table containing key-pair value mentioning the properties of the physics body. For instance: density, bounce (velocity after collision), friction, radii (for circular objects), outline (outline body), etc.

Eg:

```
local circle = display.newCircle(200,200,150)
circle:setFillColor(0,0,1)
physics.addBody( circle, "dynamic", { bounce=1 } )
```

-physics.removeBody()
It is used to remove a physics body from display object which was created using physics.addBody().

Syntax:
physics.removeBody(object)
where object: specifies the object whose physics body is to be removed.
Eg:
physics.removeBody(circle)
-physics.newJoint()
It is used to put together different objects while creating complex physical objects.

Eg: wheels to be joined to a vehicle, etc. If one of the joined bodies is destroyed, it will destroy the respective joint also.

Syntax:
physics.newJoint(joint_Category, …)
where joint_Category: specifies the type of joint to be added. For instance: pivot, wheel, rope, etc.

-physics.setGravity()

It is used to provide the x and y components (units m/s^2) with respect to standard gravity vector.

Eg:
physics.setGravity(0, 7)
physics.setGravity(0, 9.8) // Default value

-physics.setScale()
It is used for maintaining pixel per meter ratio to achieve accuracy. This particular ratio is responsible for mapping on-screen Corono coordinates with simulated physics coordinates.

Default scale values is 30 which could be increased or decreased depending on the requirement.

Eg:
physics.setScale(50)

-physics.setDrawMode()
It is used to set either of the following three modes: hybrid (to overlay the collision boundaries on usual display objects), normal (default, with no collision boundaries) and debug (it displays only outlines of collision engine).

Eg: physics.setDrawMode("normal") // default

-physics.setContinuous()
It is used to set the continuous physics i.e it enables or disables tunneling effects. By default, Box2D carries out collision detection.
Eg: physics.setContinuous(false)

16.4 Physics Joints

16.4.1 Pivot Joint

Pivot joint is used to join two bodies that overlap at a certain point such that the relative rotation of the bodies is not restricted. For instance: rotating door, etc.

Syntax:
local pivot_Joint = physics.newJoint(“pivot”, body_A, body_B, localAnchor_X, localAnchor_Y)
where body_A and body_B: two bodies to be joined
localAnchor_X: point in body A around which it will rotate.
localAnchor_Y: point in body B around which it will rotate.

Eg:

```
local circle = display.newCircle(W*0.5,H*0.5,W*0.4,W*0.4)
circle:setFillColor(0.9)
physics.addBody( circle, "static", { density=1.6, friction=0.5, bounce=0.2 } )
local rect = display.newRect(circle.x,circle.y,W*0.01,circle.contentHeight*.5)
rect.anchorY = 0
rect:setFillColor(1,0,0)

physics.addBody( rect, "dynamic", { density=1.6, friction=0.5, bounce=0.2 } )

local pivotJoint = physics.newJoint( "pivot", circle, rect, circle.x,circle.y )
pivotJoint.isMotorEnabled = true
pivotJoint.motorSpeed = 20
pivotJoint.maxMotorTorque = 1000
```

16.4.2 Piston Joint:

Pivot point is used to create a joint between two bodies such that bodies can be only be moved along a specified axis. For instance: sliding doors, etc.

Syntax:

local piston_Joint = physics.newJoint(“piston”, body_A, body_B, localAnchor_X, localAnchor_Y, axis_X, axis_Y)
where body_A and body_B: two bodies to be joined
localAnchor_X: point in body A to restrict the movement along the axis line.
localAnchor_Y: point in body B to restrict the movement along the axis line.
axis_X: the axis (line) of movement with respect to body_A)
axis_Y: the axis (line) of movement with respect to body_B)

Eg:

```
local rect = display.newRect(W*0.5,H*0.5,W*0.7,H*0.8)
rect:setFillColor(0.9,0.9,0.9,0.8)
local block1,block2 = {},{}
local ypos = rect.contentBounds.yMax-H*0.02

local box1 = display.newRect(rect.x,rect.contentBounds.yMax-
H*0.02,W*0.85,rect.contentHeight*0.08)
box1:setFillColor(1,0,0)
physics.addBody( box1, "static", { isSensor=true } )

local box2 = display.newRect(rect.x,rect.contentBounds.yMax-
H*0.22,W*0.4,rect.contentHeight*0.25)
box2:setFillColor(1,0,0)
box2:setStrokeColor(0.5)
box2.strokeWidth = 5
physics.addBody( box2, "dynamic", { bounce=1 } )

joint = physics.newJoint( "piston", box1, box2, box2.x, box2.y, 0, 1 )
rect:toFront()
```

```
for i=1,3 do
block1[#block1] = display.newRect(rect.x,ypos,W*0.85,rect.contentHeight*0.08)
ypos = ypos - rect.contentHeight*0.5
end

ypos = rect.contentBounds.yMax-H*0.22
for i=1,2 do
block2[#block2] = display.newRect(rect.x,ypos,W*0.4,rect.contentHeight*0.25)
block2[#block2]:setFillColor(0,0,0,0)
block2[#block2]:setStrokeColor(0.5)
block2[#block2].strokeWidth = 5
ypos = ypos - rect.contentHeight*0.5
end

joint.isLimitEnabled = true
joint:setLimits( -H*0.4, 0 )
joint.isMotorEnabled = true
joint.motorSpeed = -30
joint.maxMotorForce = 1000
```

16.4.3 Distance Joint

It is used to create a joint between two bodies fixed at a certain specified distance (greater than zero).

Syntax:
local distance_Joint = physics.newJoint(“distance”, body_A, body_B, anchorA_X, anchorA_Y, anchorB_X, anchorB_Y)
where body_A and body_B: two bodies to be joined
anchorA_X, anchorA_Y, anchorB_X, anchorA_Y: anchor points (point on a respective body specifying its interaction).

Eg:

```
local rope = display.newRect(W*0.5,H*0.3,W,H*0.007)
```

```
physics.addBody( rope, "static", { density=1.6, friction=0.5, bounce=0.2 } )

local icon = display.newImageRect("ball.png", W*0.2, W*0.2)
icon:setFillColor(1,0,0)
icon.anchorX = 0.5
icon.anchorY = 0.5
icon.x = rope.x
icon.y = rope.y + H*0.1

physics.addBody( icon, "dynamic", { density=1.6, friction=0.5, bounce=0.2 } )
icon:setLinearVelocity( 40, 40 )

local distanceJoint = physics.newJoint( "distance", rope, icon, rope.x, rope.y,
icon.x, icon.y )
distanceJoint.dampingRatio = 1
distanceJoint.frequency = 100
distanceJoint.length = 100
```

16.4.4 Wheel joint

It is used for simulating a body such that it moves freely like a wheel mounted on a shock absorber of a four wheeler. It works on the combined principle of piston and pivot joints.

Syntax:
local wheel_Joint = physics.newJoint(“wheel”, body_A, body_B, anchor_X, anchor_Y, axis_X, axis_Y)
where body_A and body_B: two bodies to be joined
localAnchor_X and localAnchor_Y: anchor points (point on a respective body specifying its interaction).
axis_X: the axis (line) of movement with respect to body_A)
axis_Y: the axis (line) of movement with respect to body_B)
Eg:

```
local carbody = display.newGroup()
local platform = display.newRect(W*0.5,H,W,H*0.1)
physics.addBody( platform, "static", { bounce=0, friction=1 } )

local letfRect =
display.newRect(platform.contentBounds.xMin,platform.contentBounds.yMin,W
*0.1,H*0.1)
physics.addBody( letfRect, "static", { bounce=0, friction=1 } )

local rightRect =
display.newRect(platform.contentBounds.xMax,platform.contentBounds.yMin,W
*0.1,H*0.1)
physics.addBody( rightRect, "static", { bounce=0, friction=1 } )

local car = display.newRect(W*0.3,platform.y,W*0.28,H*0.07)
physics.addBody( car, "dynamic" )
car.isFixedRotation = true
carbody:insert(car)
local carwheel = display.newImageRect("car-wheel.png", W*0.08, W*0.08)
carwheel:setFillColor(1,0,0)
carwheel.anchorX = 0.5
carwheel.anchorY = 0.5
carbody:insert(carwheel)

carwheel.x = car.contentBounds.xMin+car.contentWidth*0.2
carwheel.y = car.y
physics.addBody( carwheel, "dynamic", { bounce=0.5, friction=0.8, radius=15 } )

local carwheel2 = display.newImageRect("car-wheel.png", W*0.08, W*0.08)
carwheel2:setFillColor(1,0,0)
carwheel2.anchorX = 0.5
carwheel2.anchorY = 0.5

carwheel2.x = car.contentBounds.xMax-car.contentWidth*0.2
carwheel2.y = car.y
```

```
physics.addBody( carwheel2, "dynamic", { bounce=0.5, friction=0.8,
radius=15 } )
carbody:insert(carwheel2)

joint1 = physics.newJoint( "wheel", carwheel,car ,
car.contentBounds.xMin+car.contentWidth*0.2, car.y, 1, 1 )
joint1.springFrequency = 1
joint1.springDampingRatio = 0.1
carbody:insert(carwheel2)
joint2 = physics.newJoint( "wheel", carwheel2,car, car.contentBounds.xMax-
car.contentWidth*0.2, car.y, 1, 1 )
joint2.springFrequency = 1
joint2.springDampingRatio = 0.1

carwheel:applyTorque( 2 )
carwheel2:applyTorque( 2 )

transition.to( carbody, { time=2000, alpha=0, delay=80, x=W*0.8 } )
```

16.4.5 Pulley Joint

Pulley joint is designed in such a way to support the movement between two bodies attached together by a specific drive element. Drive elements could be a rope, cable or a belt that runs over the pulley inside the groove.

Syntax:
local pulley_Joint = physics.newJoint(“pulley”, body_A, body_B, anchorA_X, anchorA_Y, anchorB_X, anchorB_Y, bodyA_X, bodyA_Y, bodyB_X, bodyB_X, 1.00)
where body_A and body_B: two bodies to be joined
anchorA_X, anchorA_Y, anchorB_X, anchorA_Y: anchor points (point on a respective body specifying its interaction).

1.00: default pulley ratio which could be adjusted as per the requirement.

Eg:

```
local icon = display.newImageRect("balance.png", W*0.9, W*0.9)
icon:setFillColor(1,0,0)
icon.anchorX = 0.5
icon.anchorY = 0.5
icon.x = W*0.5
icon.y = H*0.5

local bodyA = display.newRect( 0, 0, W*0.3,H*0.01)
bodyA:setFillColor( 1, 0.2, 0.4 )
physics.addBody( bodyA, "dynamic", { radius=W*0.001,bounce=0.8,
radius=40 } )
bodyA.x, bodyA.y =icon.contentBounds.xMin+W*0.16, icon.y


local bodyB = display.newRect( 0, 0, W*0.3,H*0.008 )
bodyB:setFillColor( 1, 0.2, 0.4 )
physics.addBody( bodyB, "dynamic", {radius=W*0.001, bounce=0.8,
radius=26 } )
bodyB.x, bodyB.y = icon.contentBounds.xMax-W*0.16, icon.y

joint = physics.newJoint( "pulley", bodyA, bodyB, bodyA.x, bodyA.y-H*0.1,
bodyB.x, bodyB.y-H*0.09, bodyA.x, bodyA.y, bodyB.x, bodyB.y, 1.0 )
```

16.4.6 Weld Joint

Weld joint is used to attach two bodies at a specific orientation in such a way that it restricts any sort of movement.
Syntax:

```
local weld_Joint = physics.newJoint( “weld”, body_A, body_B, localAnchor_X,
localAnchor_Y)
```

where body_A and body_B: two bodies to be joined
localAnchor_X: point in body A to restrict the movement along the axis line.

localAnchor_Y: point in body B to restrict the movement along the axis line.

Eg:

```
local rect = display.newRect(W*0.5,H*0.5,W*0.15,H*0.035)
rect:setFillColor(1,0,0)
physics.addBody( rect, "dynamic", { bounce=1 } )
local rect1 = display.newRect(rect.x,rect.y+rect.contentHeight,W*0.3,H*0.05)
rect1:setFillColor(0,1,0)
physics.addBody( rect1, "dynamic", { bounce=1 } )
local circle1 =
display.newCircle(rect1.contentBounds.xMin+rect1.contentWidth*0.25,rect1.y+r
ect1.contentHeight*0.5,W*0.04)
circle1:setFillColor(0,0,1)
physics.addBody( circle1, "dynamic", { bounce=1 } )
local circle2 = display.newCircle(rect1.contentBounds.xMax-
rect1.contentWidth*0.25,rect1.y+rect1.contentHeight*0.5,W*0.04)
circle2:setFillColor(0,0,1)
physics.addBody( circle2, "dynamic", { bounce=1 } )

local platform = display.newRect(W*0.5,H,W,H*0.1)
physics.addBody( platform, "static", { bounce=0, friction=1 } )

local letfRect =
display.newRect(platform.contentBounds.xMin,platform.contentBounds.yMin,W
*0.1,H*0.1)
physics.addBody( letfRect, "static", { bounce=0, friction=1 } )

local rightRect =
display.newRect(platform.contentBounds.xMax,platform.contentBounds.yMin,W
*0.1,H*0.1)
physics.addBody( rightRect, "static", { bounce=0, friction=1 } )

joint1 = physics.newJoint( "weld", rect1, rect, rect.x, rect.y )
joint2 = physics.newJoint( "weld", circle1, rect1, rect1.x, rect1.y )
joint3 = physics.newJoint( "weld", circle2, rect1, rect1.x, rect1.y )
```

16.4.7 Frictional Joint

Frictional Joints are used to restrict or control the relative motion between two bodies.

Syntax:

local weld_Joint = physics.newJoint(“weld”, body_A, body_B, localAnchor_X, localAnchor_Y)
where body_A and body_B: two bodies to be joined
localAnchor_X: point in body A to restrict the movement along the axis line.
localAnchor_Y: point in body B to restrict the movement along the axis line.

Eg:

```
local obstacles = {}
local xPos = W*.25

local BG = display.newRect(W*.5,H*.5,W,H)
local heading = display.newText("Example of Friction Joint",W*.5,H*.05,native.systemFont,W*.06)
heading:setFillColor(0)

local msg= "The car on right side moving slow due to bad condition of road and left side moving fast because it's smooth condition of road"
local txt = display.newText(msg,W*.5,H*.15,W*.8,H*.1,native.systemFont,W*.04)
txt:setFillColor(0)

local grp = display.newGroup()
local roadObj1 = display.newImage("road_line.png",xPos,H*.58)
local bgrect1 = display.newRect(roadObj1.x,roadObj1.y-roadObj1.contentHeight*.4,W*.3,H*.3)
physics.addBody( bgrect1, "static" )
bgrect1.alpha = 0.01
roadObj1.width = W*.7
```

```
roadObj1.height = H*.7
grp:insert(roadObj1)
grp:insert(bgrect1)

local roadObj2 = display.newImage("road_line.png",xPos +
roadObj1.contentWidth*.5 + W*.17,H*.58)
local bgrect2 = display.newRect(roadObj2.x,roadObj2.y-
roadObj2.contentHeight*.4,W*.3,H*.3)
physics.addBody( bgrect2, "static")
bgrect2.alpha = 0.01
bgrect2.type = "dirt"
roadObj2.width = W*.7
roadObj2.height = H*.7
grp:insert(roadObj2)
grp:insert(bgrect2)

local cars1 =
display.newImage("car.png",roadObj1.x,roadObj1.contentBounds.yMin+H*.05)
cars1.rotation = 180
physics.addBody( cars1, "dynamic", { friction = 0.9,isSensor = true } )
-- cars1.angularVelocity = 100
cars1.isFixedRotation = true
cars1.width,cars1.height = W*.2,W*.2
cars1.id = 1
grp:insert(cars1)

local cars2 =
display.newImage("car.png",roadObj2.x,roadObj2.contentBounds.yMin+H*.05)
cars2.rotation = 180
physics.addBody( cars2, "dynamic", { friction = 1,isSensor = true } )
-- cars2.angularVelocity = 100
cars2.isFixedRotation = true
cars2.width,cars2.height = W*.2,W*.2
cars2.id = 2
grp:insert(cars2)
```

```
local obstacle = {}
local xx, yy = W*.7,H*.3
for i=1,5 do
obstacle[#obstacle+1] = display.newCircle(xx,yy,W*.03)
obstacle[#obstacle]:setFillColor(121/255, 76/255, 19/255)
grp:insert(obstacle[#obstacle])
xx = xx + W*.15
yy = yy + H*.1
if i%2 == 0 then
xx = W*.7
end
end

local joints1 = physics.newJoint( "friction", bgrect1, cars1, cars1.x, cars1.y )
-- Possible properties/methods for the joint
joints1.maxForce = 0.22
joints1.maxTorque = 0.022

local joints2 = physics.newJoint( "friction", bgrect2, cars2, cars2.x, cars2.y )
-- Possible properties/methods for the joint
joints2.maxForce = 0.35
joints2.maxTorque = 0.035
cars1:toFront()
cars2:toFront()
```

16.4.8 Touch Joints

Touch joint is used to pull a point on one body to a desired targeted location.
Syntax:

```
local touch_Joint = physics.newJoint( “touch”, body, localAnchor_X,
localAnchor_Y)
```

where body: body on which operation is to be performed.
anchor_X, anchor_Y: anchor points (point on a respective body specifying its interaction).

Eg.

```
local obj = {}
local objectGroup = display.newGroup()
local jointPoint
local touchPoint

local function moveObj(xPos , yPos)
jointPoint:setTarget( xPos, yPos)

touchPoint.x, touchPoint.y = xPos, yPos
touchPoint.alpha = 1
transition.to( touchPoint, { time=600, alpha=0.2, delay=80, tag="touchPoint",
transition=easing.outQuad } )
end

local function onTouch(event)
local phase = event.phase
local x,y
if "began" == phase then
elseif "ended" == phase or "cancelled" == phase then
x = event.x
y = event.y
moveObj(x, y)
end
return true
end

local BG = display.newRect(W*.5,H*.5,W,H)
local noteTxt = display.newText("Touch on screen to move the
anchor.",W*.5,H*.05,native.systemFont,W*.05)
noteTxt:setFillColor(0)

local anchorObj = display.newImage( objectGroup,"anchor.png", 0, 0, W*.08,
W*.08 )
```

```
anchorObj.width,anchorObj.height = W*.15, W*.15
physics.addBody( anchorObj, "dynamic", { bounce=0 } )
anchorObj.x, anchorObj.y = display.contentCenterX, 300
obj[#obj+1] = anchorObj

touchPoint = display.newCircle( objectGroup, 0, 0, 15 )
touchPoint:setFillColor( 1,0,0.2 )
touchPoint.alpha = 0

anchorObj:toBack()
touchPoint:toBack()

jointPoint = physics.newJoint( "touch", anchorObj, anchorObj.x, anchorObj.y-20 )
jointPoint.anchorX,jointPoint.anchorY = 0.5,0

jointPoint.maxForce = 1000
jointPoint.frequency = 1
jointPoint.dampingRatio = 0.5
Runtime:addEventListener("touch",onTouch)
```

16.4.9 Gear joint

It is used to create gear-like-structure. This joint can connect two or more bodies in such a way that the movement of one has an affect on other.

Syntax:
local gear_Joint = physics.newJoint("gear", body_A, body_B, joint_1, joint_2, gear_ratio)

where body_A and body_B: two bodies to be joined
joint_1 and joint_2: piston or pivot joint with respect to a particular gear joint.
gear_ratio: it specifies the ratio at which motor-driven joint drives the otherjoint in gear setup structure. It can either be positive or negative depending on the direction.

Eg:

```
local joints = {}
local staticBox = display.newRect( 0, 0, 160, 40 )
staticBox:setFillColor( 0.2, 0.2, 1 )
physics.addBody( staticBox, "static", { isSensor=true } )
staticBox.x, staticBox.y = display.contentCenterX-30, 300

local gearL = display.newImageRect("industrial.png", W*0.2, W*0.2)
gearL.anchorX = 0.5
gearL.anchorY = 0.5
physics.addBody( gearL, "dynamic", { bounce=0, friction=0, radius=40 } )
gearL.x, gearL.y = staticBox.x-55, 300
gearL.radius = 40


local gearR = display.newImageRect("car-wheel.png", W*0.2, W*0.2)
gearR.anchorX = 0.5
gearR.anchorY = 0.5
physics.addBody( gearR, "dynamic", { bounce=0, friction=0, radius=70 } )
gearR.x, gearR.y = staticBox.x+55, 300
gearR.radius = 70

local bar = display.newImageRect("robot.png", W*0.2, W*0.2)

physics.addBody( bar, "dynamic", { bounce=0, friction=0 } )
bar.x, bar.y = staticBox.x+140, gearR.y+(bar.height/2)-20
bar.anchorX = 0.5
bar.anchorY = 0.5
joints[#joints+1] = physics.newJoint( "pivot", staticBox, gearL, gearL.x, gearL.y )
joints[#joints+1] = physics.newJoint( "pivot", staticBox, gearR, gearR.x, gearR.y )

joints[#joints+1] = physics.newJoint( "piston", staticBox, bar, bar.x, bar.y, 0, 1 )
joints[#joints+1] = physics.newJoint( "gear", gearL, gearR, joints[1], joints[2], 1.0
)
```

```
--joints[#joints].ratio = 1.0

joints[#joints+1] = physics.newJoint( "gear", gearR, bar, joints[2], joints[3],
-1*(gearL.radius/gearR.radius) )

joints[1].isMotorEnabled = true
joints[1].motorSpeed = 15
joints[1].maxMotorTorque = 1000
joints[1].isLimitEnabled = true
joints[1]:setRotationLimits( -90, 180 )
```

16.4.10 Rope Joint:

It is used to constrain a point on each respective body to a maximum distance apart.

Syntax:

```
local rope_Joint = physics.newJoint( "rope", body_A, body_B, offsetA_x, offsetA_y,
offsetB_x, offsetB_y )
```

body_A and body_B: two bodies to be joined
offset values: Optional values set in accordance with the center point. By default set to zero (I.e rope will be attached to center of each body).

Eg:

```
local platform = display.newRect(W*0.5,H,W,H*0.1)
physics.addBody( platform, "static", { density=1.6, friction=0.5, bounce=0.2 } )

local balloon = display.newImageRect("balloon.png", W*0.2, W*0.2)
balloon:setFillColor(1,0,0)
balloon.anchorX = 0.5
balloon.anchorY = 0.5
balloon.x = W*0.5
balloon.y = H*0.2
```

```
physics.addBody( balloon, "dynamic", { bounce=1, radius=25 } )

local man = display.newImageRect("man.png", W*0.2, W*0.2)
man:setFillColor(1,0,0)
man.anchorX = 0.5
man.anchorY = 0.5
man.x = balloon.x
man.y = balloon.contentBounds.yMax+balloon.contentHeight
physics.addBody( man, "dynamic", { bounce=0.5, radius=25 } )

local joint = physics.newJoint( "rope", balloon, man, 0, -20, 0, 20 )
-- Possible properties/methods for the joint
joint.maxLength = 200
```

16.5 Game Examples (2 hours)

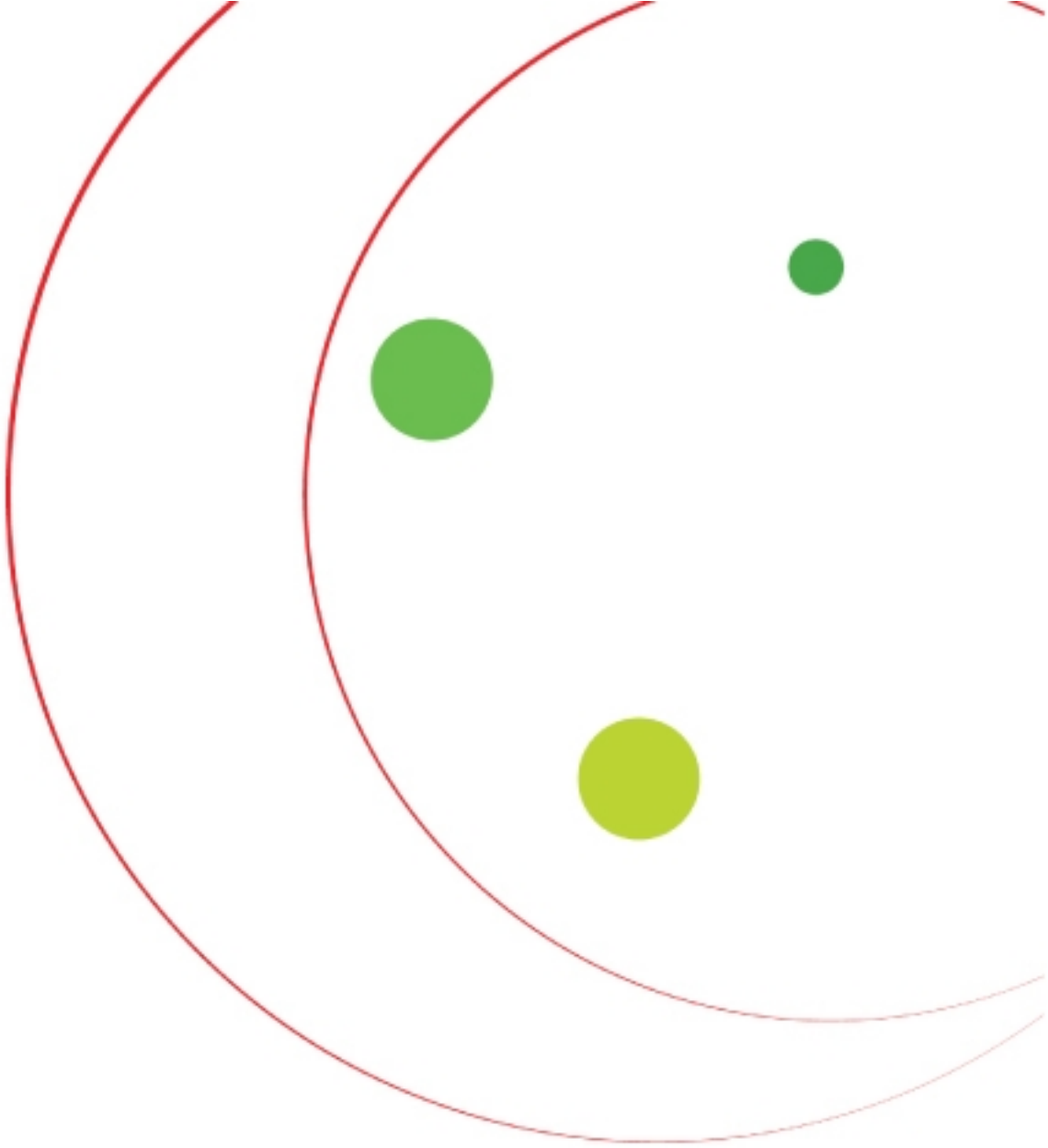
1. Balloon Tap Game:
   https://www.dropbox.com/sh/cdn7kskxoj7p6sd/AAATehmOfYfX3FrKIeco9fyEa?dl=0

2. Collect bricks Game:
   https://www.dropbox.com/sh/r4gv7o6q6tq7if5/AADQHexrfkugdvwKvaq-ntHQa?dl=0

3. Bouncing ball (Physics):
   https://www.dropbox.com/sh/xa7hcc1ava8739a/AADXvkIkbI6sAz5FuWwF_JsFa?dl=0

* *Corona is one of the best cross-platform app development SDK owned by Appodeal. Based on Lua language, Corona SDK allows mobile game developers to create apps and 2D games. It serves as a perfect solution for those looking for a flexible and powerful cross-platform development tool.*

This training material from Redbytes (a leading Corona App development company in India) is a compilation of what we have learned and experienced over years by using this platform. It is in no way connected or endorsed by Corona or Appodeal. Through this training tutorial, our team aims to share what we have learned through the years about Corona 2D game development in simple language.



Kalas road, Vishrantwadi,
Pune, Maharashtra-411015,

Email : info@redbytes.in
Fax : 904 701 6310
Phone :+91 81 1386 0000 +91 82 3748 2321

www.redbytes.com