Graphical User Interfaces and Widgets Part 2

Haeyong Chung
Spring 2017

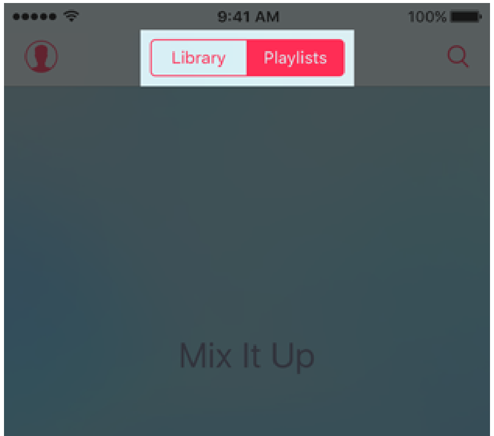
Slides contain examples from the official Corona docs as well as the textbook.

Segmented Control

Segmented Control

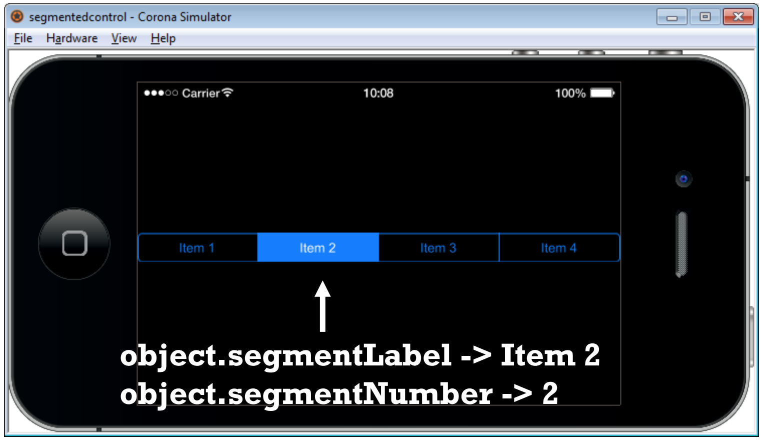
- widget.newSegmentedControl(options)
- Options:

```
local segmentedControl = widget.newSegmentedControl(
    {
        left = 0,
        top = 150,
        segmentWidth = 120,
        segments = { "Item 1", "Item 2", "Item 3", "Item 4" },
        defaultSegment = 2,
        onPress = onSegmentPress
    }
)
```



Segmented Control

- Properties:
  - object.segmentLabel
  - object.segmentNumber
- Basic Visual Options:
  - labelSize
  - labelFont
  - labelColor
  - labelXOffset, labelYOffset

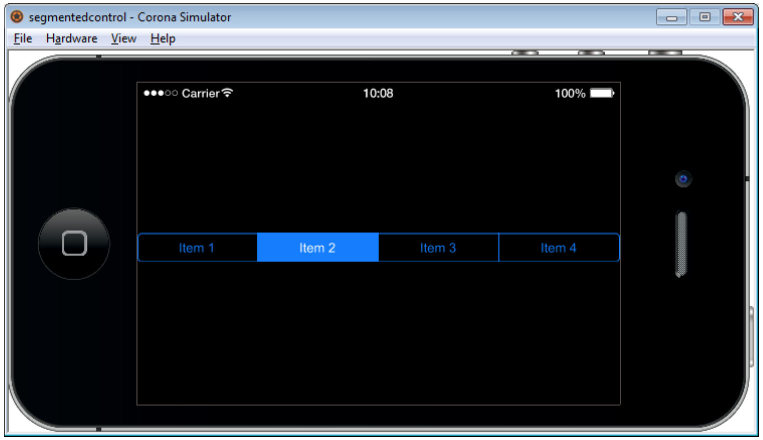


object.segmentLabel -> Item 2
object.segmentNumber -> 2

Segmented Control Example

```
local widget = require( "widget" )
-- Listen for segmented control events
local function onSegmentPress( event )
    local target = event.target
    print( "Segment Label is:",
target.segmentLabel )
    print( "Segment Number is:",
target.segmentNumber )
end

-- Create a default segmented control
local segmentedControl =
widget.newSegmentedControl(
    {
        left = 50,
        top = 150,
        segmentWidth = 150,
        segments = { "Item 1", "Item 2", "Item
3", "Item 4" },
        defaultSegment = 2,
        onPress = onSegmentPress
    }
)
```



Slider

+ Slider

- Creates a ScrollViewWidget object.
  - widget.newScrollView(***options***)

- ***options:***
  - x & y or left & top
  - orientation (“horizontal”/”vertical”; default is “horizontal”)
  - width & height
  - value (percentage; default is 50 meaning that the slider handle begins at 50%)
  - Listner

- Properties
  - Object.value

- Methods
  - object:setValue()

Slider Example—Horizontal

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

-- Slider listener
local function sliderListener( event )
    print( "Slider at " .. event.value .. "%" )
end

-- Create the widget
local slider = widget.newSlider(
    {
        top = 200,
        left = 50,
        width = 400,
        value = 10,  -- Start slider at 10% (optional)
        listener = sliderListener
    }
)
```

Slider Example—Vertical

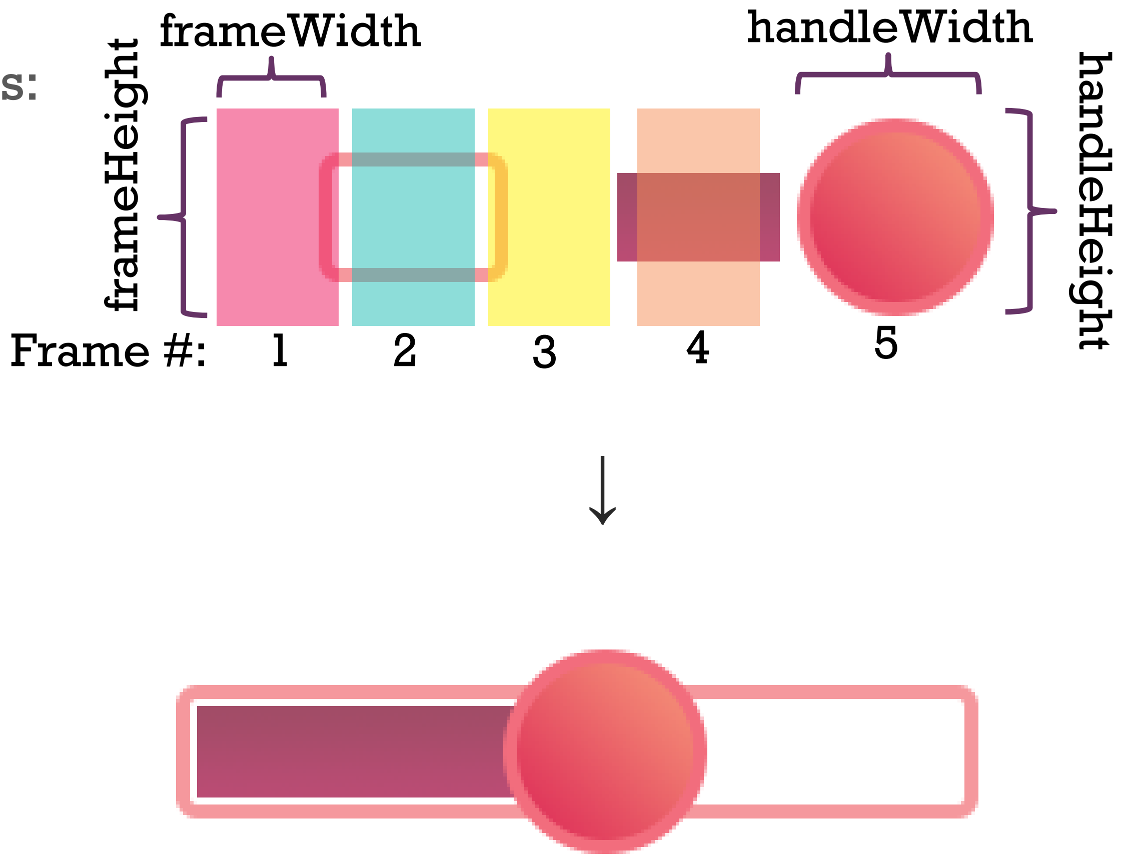
```
local widget = require( "widget" )

-- Slider listener
local function sliderListener( event )
    print( "Slider at " .. event.value .. "%" )
end

-- Create the widget
local slider = widget.newSlider(
    {
        top = 200,
        left = 50,
        orientation = "vertical",
        height = 200,
        value = 10,  -- Start slider at 10% (optional)
        listener = sliderListener
    }
)
```

Slider-Visual Customization & Image Sheet

- Using an Image Sheet
- Horizontal Slider Options:
  - sheet = sliderSheet
  - leftFrame = 1
  - middleFrame = 2
  - rightFrame = 3
  - fillFrame = 4
  - frameWidth = 36
  - frameHeight = 64
  - handleFrame = 5
  - handleWidth = 64
  - handleHeight = 64



Slider Example—Image Sheet

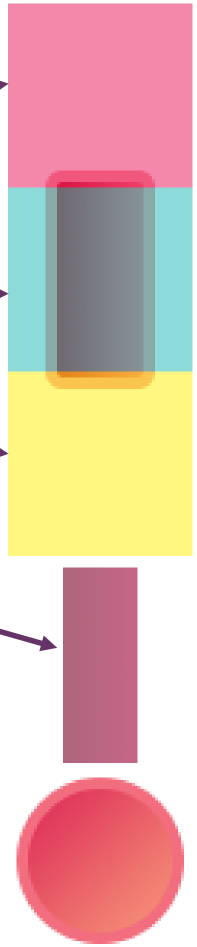
```
local widget = require( "widget" )
local function sliderListener( event )
    print( "Slider at " .. event.value .. "%" )
end
local options = {
    frames = {
        { x=0, y=0, width=36, height=64 },
        { x=40, y=0, width=36, height=64 },
        { x=80, y=0, width=36, height=64 },
        { x=124, y=0, width=36, height=64 },
        { x=168, y=0, width=64, height=64 }
    },
    sheetContentWidth = 232,
    sheetContentHeight = 64
}
local sliderSheet = graphics.newImageSheet( "sliderSheet.png", options )
```

Slider Example—Image Sheet (continued)

```
-- Create the widget
local slider = widget.newSlider(
    {
        sheet = sliderSheet,
        leftFrame = 1,
        middleFrame = 2,
        rightFrame = 3,
        fillFrame = 4,
        frameWidth = 36,
        frameHeight = 64,
        handleFrame = 5,
        handleWidth = 64,
        handleHeight = 64,
        top = 200,
        left= 50,
        orientation = "horizontal",
        width = 300,
        listener = sliderListener
    }
)
```

Slider-Visual Customization & Image Sheet

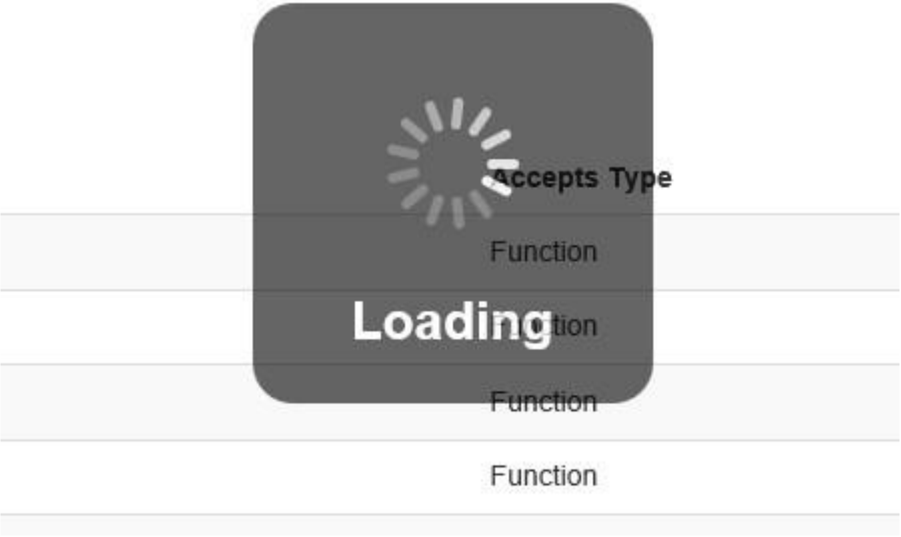
- Vertical Slider Options:
  - topFrame
  - middleVerticalFrame
  - bottomFrame
  - fillVerticalFrame
  - frameWidth, frameHeight



Spinner

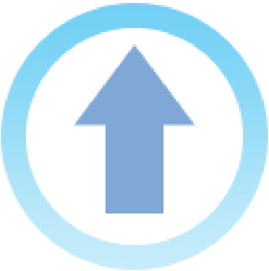
Spinner

- Create a Spinner object:
  - widget.newSpinner(options)
- Options:
  - x & y and left & top
  - width and height

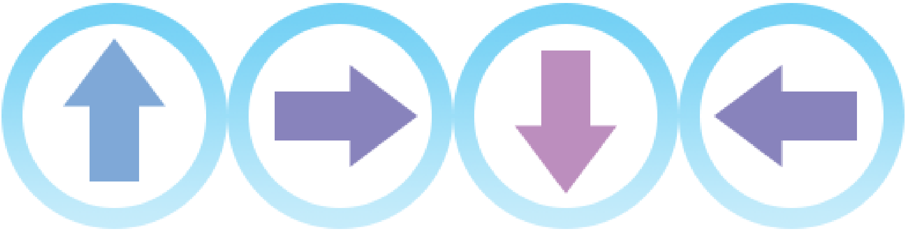


Spinner (continued)

- Single Frame Construction
  - startFrame ,
  - deltaAngle (degrees),
  - incrementEvery (milliseconds)



- Multi-Frame Construction
  - startFrame ,
  - count ,
  - time



Spinner (continued)

- Methods:
  - object:start()
  - object:stop()

Spinner Single-Frame

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

-- Image sheet options and declaration
-- For testing, you may copy/save the image under "Single Frame Construction" above
local options = {
    width = 128,
    height = 128,
    numFrames = 1,
    sheetContentWidth = 128,
    sheetContentHeight = 128
}
local spinnerSingleSheet = graphics.newImageSheet( "widget-spinner-single.png", options )

-- Create the widget
local spinner = widget.newSpinner(
    {
        width = 128,
        height = 128,
        sheet = spinnerSingleSheet,
        startFrame = 1,
        deltaAngle = 10,
        incrementEvery = 20
    }
)
```

Spinner Multi-Frame

```
local widget = require( "widget" )
local options = {
    width = 128,
    height = 128,
    numFrames = 4,
    sheetContentWidth = 512,
    sheetContentHeight = 128
}
local spinnerMultiSheet = graphics.newImageSheet( "widget-spinner-multi.png",
options )

local spinner = widget.newSpinner(
    {
        width = 128,
        height = 128,
        sheet = spinnerMultiSheet,
        startFrame = 1,
        count = 4,
        time = 800
    }
)
spinner:start()
```

Switch

Switch

- widget.newSwitch(***options***)
- ***Options***:
  - style: "radio“ or “checkbox”
  - initialSwitchState (true/false),
  - onPress (the same as event.phase = “began”)
  - onRelease (the same as event.phase = “ended”)
  - onEvent (you can check everything; “began”, “moved”, or “ended)
- Methods
  - Object.setState()
- Property
  - Object.isOn

Switch—Radio Button

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

-- Handle press events for the buttons
local function onSwitchPress( event )
   local switch = event.target
   print( "Switch with ID '"..switch.id.."' is on: "..tostring(switch.isOn) )
end

-- Create a group for the radio button set
local radioGroup = display.newGroup()
```

```
-- Create two associated radio buttons (inserted into the same display group)
local radioButton1 = widget.newSwitch(
   {
      left = 150,
      top = 200,
      style = "radio",
      id = "RadioButton1",
      initialSwitchState = true,
      onPress = onSwitchPress
   }
)
radioGroup:insert( radioButton1 )

local radioButton2 = widget.newSwitch(
   {
      left = 250,
      top = 200,
      style = "radio",
      id = "RadioButton2",
      onPress = onSwitchPress
   }
)
radioGroup:insert( radioButton2 )
```

+ Switch—Check Box

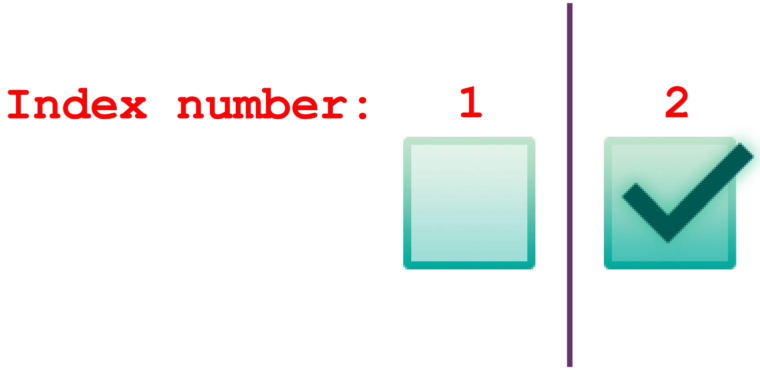
```
local widget = require( "widget" )

-- Handle press events for the checkbox
local function onSwitchPress( event )
   local switch = event.target
   print( "Switch with ID '"..switch.id.."' is on: "..tostring(switch.isOn) )
end

-- Create the widget
local checkboxButton = widget.newSwitch(
   {
      left = 250,
      top = 200,
      style = "checkbox",
      id = "Checkbox",
      onPress = onSwitchPress
   }
)
```

+ Switch—Check Box (Image Sheet)

```
local options = {
    width = 100,
    height = 100,
    numFrames = 2,
    sheetContentWidth = 200,
    sheetContentHeight = 100
}
local checkboxSheet = graphics.newImageSheet( "checkboxSheet.png", options )
local checkbox = widget.newSwitch(
    {
        left = 250,
        top = 200,
        style = "checkbox",
        id = "Checkbox",
        width = 100,
        height = 100,
        onPress = onSwitchPress,
        sheet = checkboxSheet,
        frameOff = 1,
        frameOn = 2
    }
)
```



Index number: 1 | 2

Textbox

Text Box

- native.newTextField(centerX, centerY, width, height)

```
local defaultBox
local function textListener( event )
    if ( event.phase == "began" ) then
        -- User begins editing "defaultBox"
    elseif ( event.phase == "ended" or event.phase == "submitted" ) then
        -- Output resulting text from "defaultBox"
        print( event.target.text )
    elseif ( event.phase == "editing" ) then
      print( event.newCharacters )
      print( event.startPosition )
      print( event.oldText )
       print( event.text )
    end
end
defaultBox = native.newTextBox( 140, 70, 280, 140 )
defaultBox.text = "This is line 1.\nAnd this is line2"
defaultBox.isEditable = true
defaultBox:addEventListener( "userInput", textListener )
defaultBox.isFontSizeScaled = true
defaultBox.size = 20
```

Text Box

- native.newTextField(centerX, centerY, width, height)

```
local defaultBox
local function textListener( event )
  if ( event.phase == "began" ) then
    -- User begins editing "defaultBox"
  elseif ( event.phase == "ended" or event.phase == "submitted" ) then
    -- Output resulting text from "defaultBox"
    print( event.target.text )
  elseif ( event.phase == "editing" ) then
   print( event.newCharacters )
   print( event.startPosition )
   print( event.oldText )
   print( event.text )
  end
end
defaultBox = native.newTextBox( 140, 70, 280, 140 )
defaultBox.text = "This is line 1.\nAnd this is line2"
defaultBox.isEditable = true
defaultBox:addEventListener( "userInput", textListener )
defaultBox.isFontSizeScaled = true
defaultBox.size = 20
```

Text Box

- native.newTextField(centerX, centerY, width, height)

```
local defaultBox
local function textListener( event )
    if ( event.phase == "began" ) then
        -- User begins editing "defaultBox"
    elseif ( event.phase == "ended" or event.phase == "submitted" ) then
        -- Output resulting text from "defaultBox"
        print( event.target.text )
    elseif ( event.phase == "editing" ) then
      print( event.newCharacters )
      print( event.startPosition )
      print( event.oldText )
       print( event.text )
    end
end
defaultBox = native.newTextBox( 140, 70, 280, 140 )
defaultBox.text = "This is line 1.\nAnd this is line2"
defaultBox.isEditable = true
defaultBox:addEventListener( "userInput", textListener )
defaultBox.isFontSizeScaled = true
defaultBox.size = 20
```

Text Field

+ Text Fields

- native.newTextField(centerX, centerY, width, height)

```
local defaultField
local function textListener( event )
    if ( event.phase == "began" ) then
        -- User begins editing "defaultField"

    elseif ( event.phase == "ended" or event.phase == "submitted" )
then
        -- Output resulting text from "defaultField"
        print( event.target.text )

    elseif ( event.phase == "editing" ) then
        print( event.newCharacters )
        print( event.oldText )
        print( event.startPosition )
        print( event.text )
    end
end
-- Create text field
defaultField = native.newTextField( 150, 150, 180, 30 )
defaultField:addEventListener( "userInput", textListener )
```

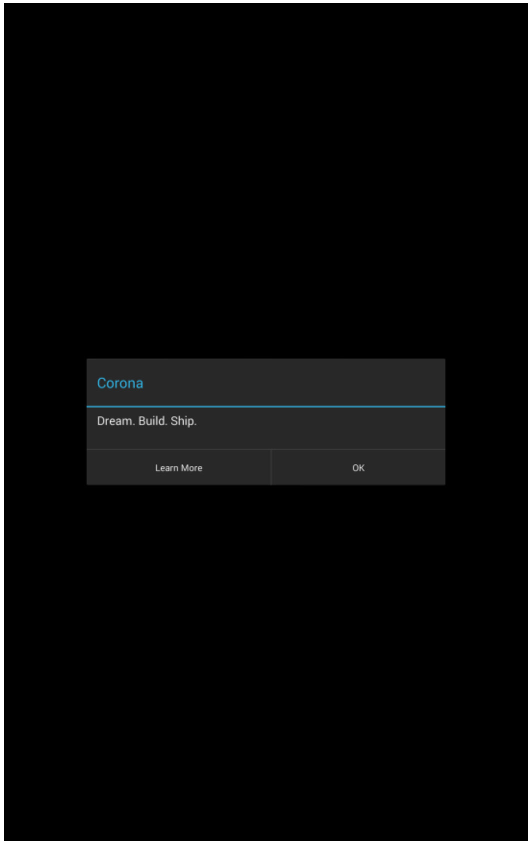
Alert Dialog

Alert Dialog

- native.showAlert(title, message [, buttonLabels [, listener]])

```
- 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
            -- Do nothing; dialog will simply dismiss
        elseif ( i == 2 ) then
            -- Open URL if "Learn More" (second button) was
clicked
            system.openURL( "http://www.coronalabs.com" )
        end
    end
end

-- Show alert with two buttons
local alert = native.showAlert( "Corona", "Dream. Build.
Ship.", { "OK", "Learn More" }, onComplete )
```



Corona

Dream. Build. Ship.