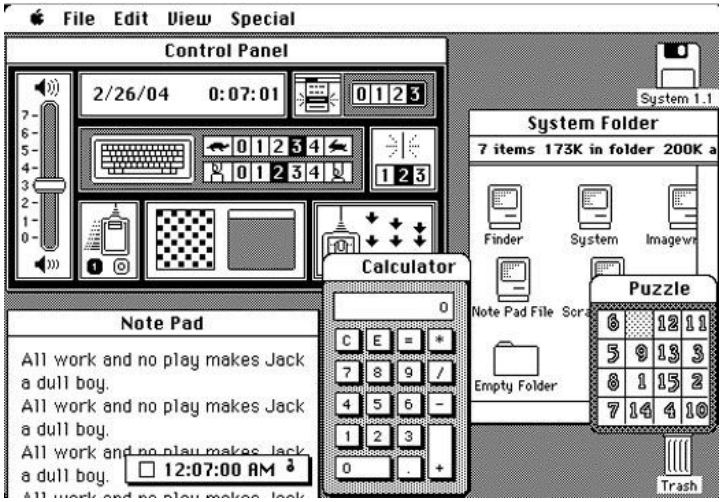
Graphical User Interfaces and Widgets

Haeyong Chung
Spring 2017

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

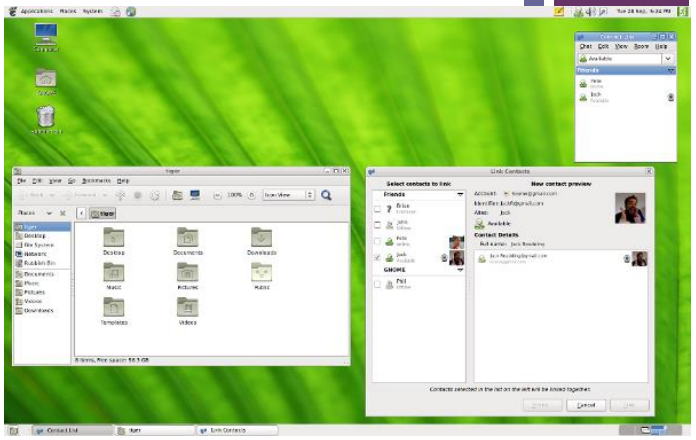
Graphical User Interface (GUI)

- "A visual way of interacting with a computer using items such as windows, icons, and menus, used by most modern operating systems." – Oxford Dictionary
- Allows the user to communicate and interact with a computer through
  - The use of symbols
  - Visual metaphors
  - Pointing devices
- GUI has replaced textual interfaces (command-line interfaces) with these visual and intuitive interfaces
- GUI generally consists of small and large **Widgets** (buttons, a scroll bar, window, etc.)



Brief Introduction to GUI

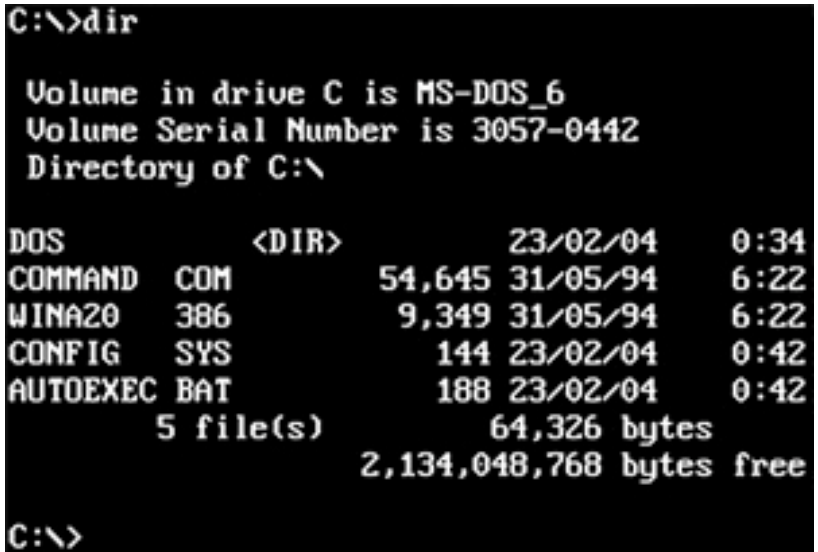
- GUI history
  - Xerox Parc's Alto, Xerox Star, and Apple Macintosh/Lisa
- WIMP
  - Windows, Icons, Menus, and Pointers
  - All desktop GUI OSs are based on WIMP
- Post-WIMP
  - WIMP components are replaced with some different forms
- Design Considerations for Mobile UI



Before the GUI Era…

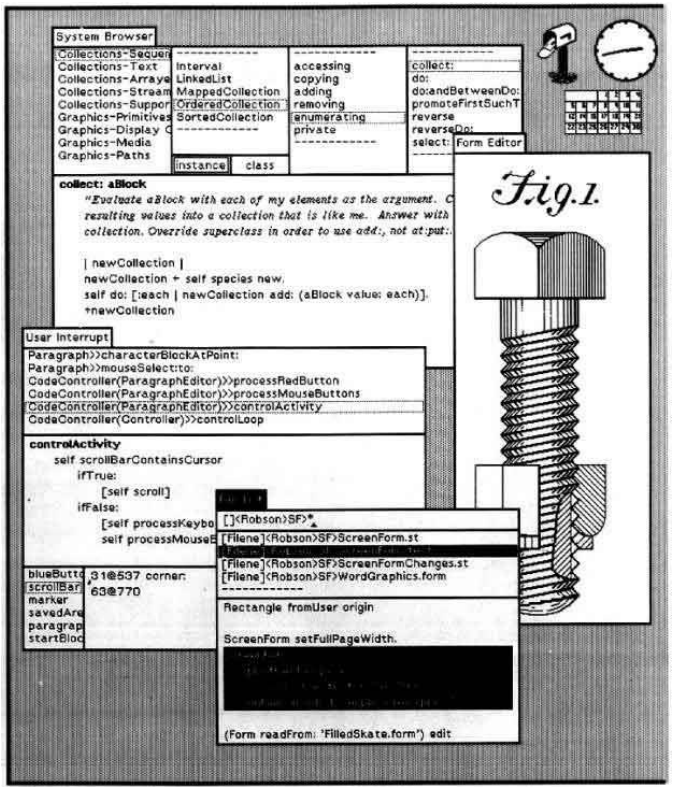
Before the GUI Era…

- Command-line interfaces (CLI)
  - It works through the user's entering typed commands with a keyboard without use of a mouse
  - No WIMP (windows, icons, menus and pointers)
  - No spelling mistakes are allowed



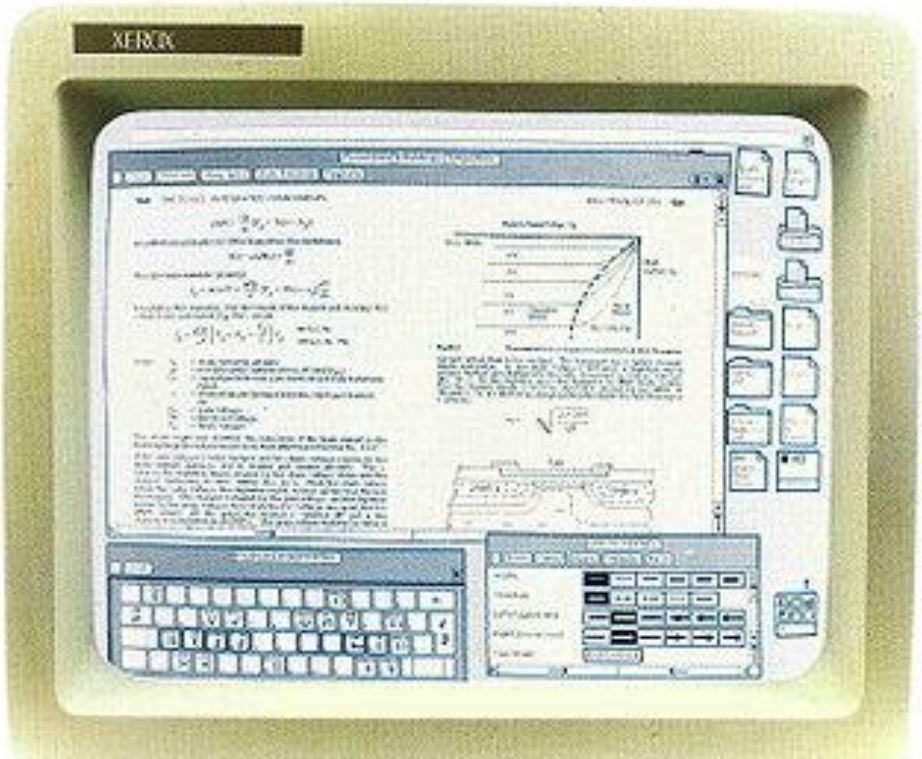
GUI History

- Xerox Alto supported the first GUI based on WIMP (1973)



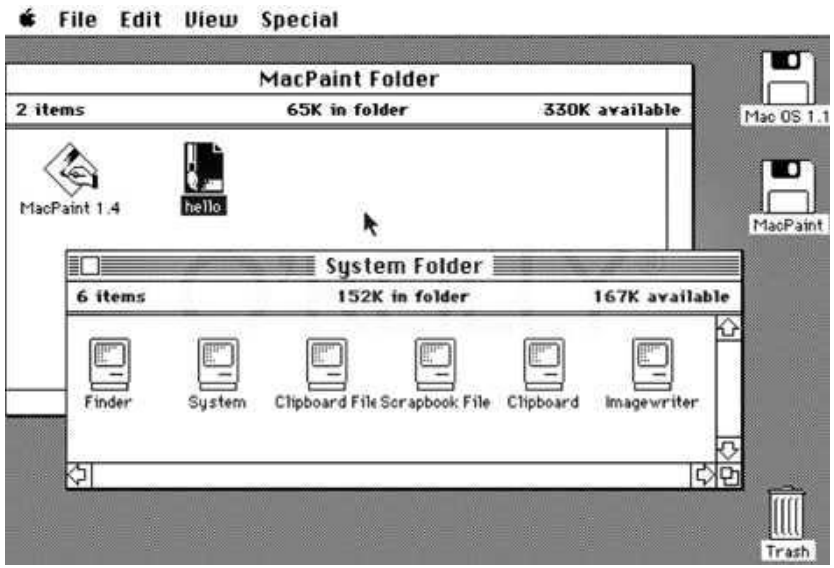
GUI History (continued)

- Xerox Star and PARC user interface (1981)
  - The first commercial GUI
  - The first WIMP GUI



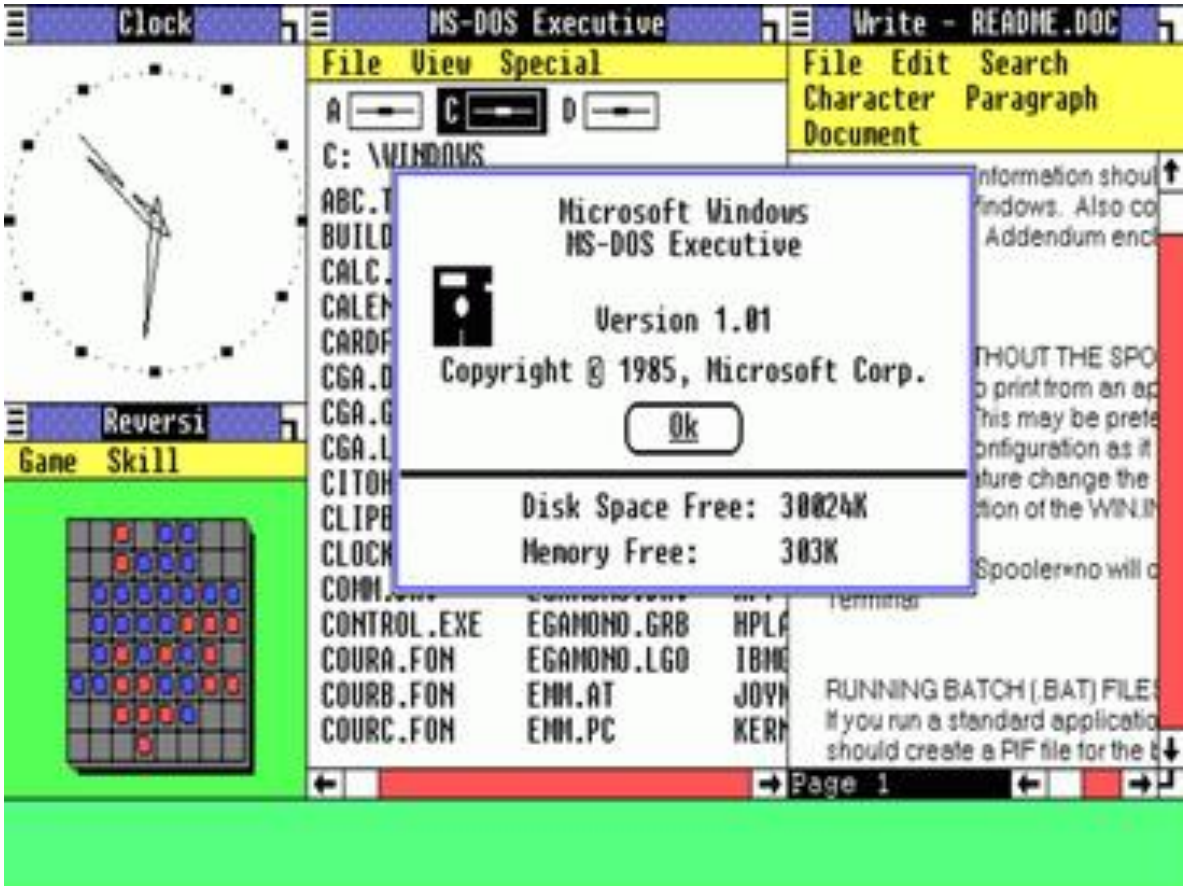
GUI History (continued)

- Apple Macintosh (1984)
  - First commercially successful PC including a GUI



GUI History (continued)

- MS Windows 1.0 (1985)



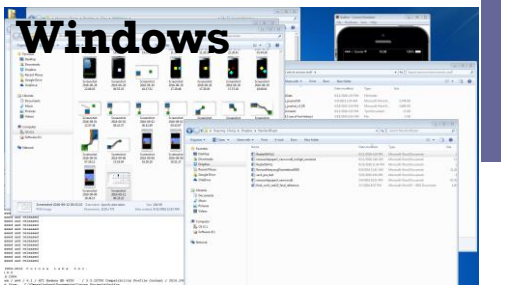
GUI History (continued)

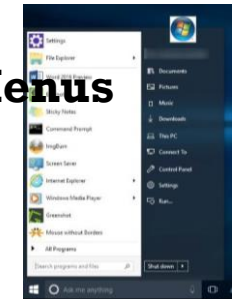
- The POST-WIMP Era begins in 2007
  - iPhone and iPad

http://iosappspy.com/wp-content/uploads/2014/10/iPhone-and-iPad-camera.png

WIMP

- **WIMP** stands for "Windows, Icons, Menus, Pointer"
  - Windows: A window is an area on the screen that displays information, with its contents being displayed independently from the rest of the screen
  - Icons: An icon is a small picture that represents objects such as a file, program, web page, or command.
  - Menus: Menus allow the user to execute commands by selecting from a list of choices.
  - Pointers: The pointer echoes movements of the pointing device, commonly a mouse or touchpad.





POST-WIMP and Mobile UI

- Post WIMP interfaces are supported by mobile devices such as:
  - Smart Phones
  - Tablets
  - Touch-enabled displays (Tabletop displays)
- These devices generally do not have room for a mouse
  - Therefore they need different ways to interact with devices
  - User's fingers takes the place of the mouse and pointer.
  - Fingers can be moved in different ways (we called 'genstures') to interact with devices.

POST-WIMP and Mobile UI (continued)

- Mobile UI design considerations:
  - Mobile UI should be based on touch-sensitive displays.
    - Click points cannot be too small or narrow in any direction (Fat Finger Problem)
  - Consider the small screen real estate.
  - Use Symbols more extensively (there is no enough room for text)
  - Maximize the content window size.
  - Not too many UIs at the same time

Corona SDK Widget

Don’t forget to include this in your code

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

Buttons

Building Buttons

- Utilize widgets:

```
local widget = require( "widget" );
local button1 = widget.newButton( options );
```

  - ***Options in a table***:
    - x & y
    - label = “Tap here!“
    - labelColor = { default={ 1, 1, 1 }, over={ 1, 1, 1, 1 } }
      - default: color
      - over: color changes to this when you press the button
    - onEvent = eventHandlerFunction
      - See also onPress, onRelease

Building Buttons

- Utilize widgets:
  - local widget = require("widget");
  - local button1 = widget.newButton(***options***);
  - ***Options in a table***:
    - x & y
    - label = “Tap here!“
    - labelColor = { default={ 1, 1, 1 }, over={ 1, 1, 1, 1 } }
      - default: color
      - over: color changes to this when you press the button
    - onEvent = eventHandlerFunction

Example: Button Default

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

-- Function to handle button events
local function handleButtonEvent( event )

    if ( "ended" == event.phase ) then
        print( "Button was pressed and released" )
    end
end

-- Create the widget
local button1 = widget.newButton(
    {
        left = 100,
        top = 200,
        id = "button1",
        label = "Default",
        onEvent = handleButtonEvent
    }
)
```

Button Options w/ 2 Images

- x & y
- label = “Tap here!“
- labelColor = { default={ 1, 1, 1 }, over={ 1, 1, 1, 1 } }
- onEvent = eventHandlerFunction
- defaultFile = "FileName.png“ --(the un-pressed state)
- overFile = "FileName2.png“ --(the pressed state)
- width & height

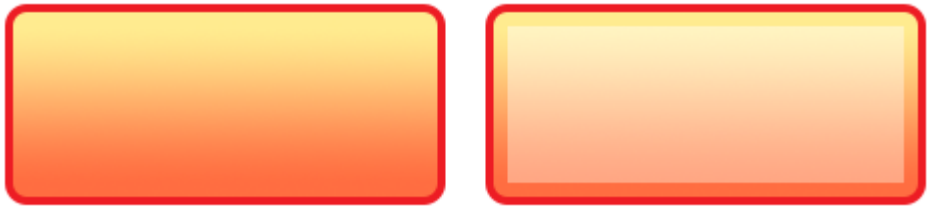
Example: Button 2-Image

```
local widget = require( "widget" )
-- Function to handle button events
local function handleButtonEvent( event )
    if ( "ended" == event.phase ) then
        print( "Button was pressed and released" )
    end
end
local button1 = widget.newButton(
    {
        width = 300,
        height = 300,
        defaultFile = "buttonDefault.png",
        overFile = "buttonOver.png",
        label = "button",
        onEvent = handleButtonEvent
    }
)
-- Center the button
button1.x = display.contentCenterX
button1.y = display.contentCenterY
-- Change the button's label text
button1:setLabel( "2-Image" )
```

Button Options w/ 2 Frames

- x & y
- label = “Tap here!“
- labelColor = { default={ 1, 1, 1 }, over={ 1, 1, 1, 1 } }
- onEvent = eventHandlerFunction
- sheet = sheetName
- defaultFrame = 1
- overFrame = 2

Image Sheet (default and over)

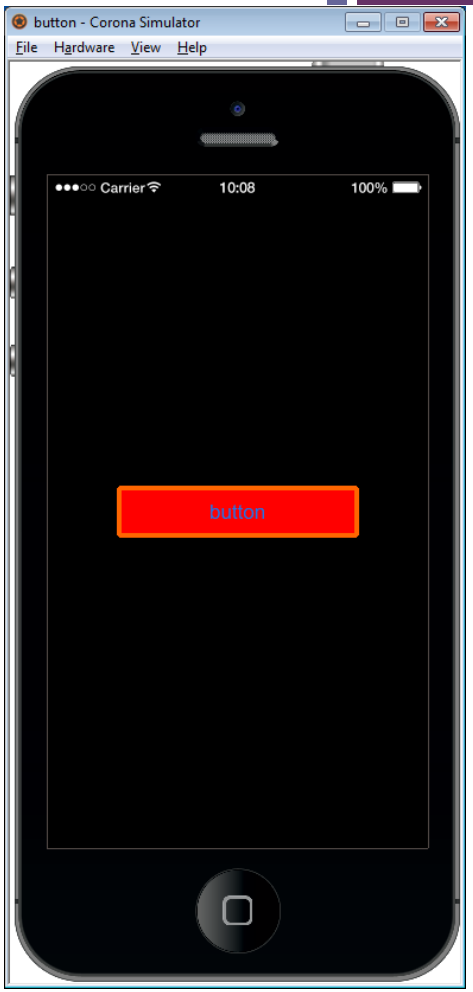


Example: Button 2-Frame

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

-- Function to handle button events
local function handleButtonEvent( event )
    if ( "ended" == event.phase ) then
        print( "Button was pressed and released" )
    end
end

-- Image sheet options and declaration
-- For testing, you may copy/save the image under "2-Frame Construction" above
local options = {
    width = 240,
    height = 120,
    numFrames = 2,
    sheetContentWidth = 480,
    sheetContentHeight = 120
}
local buttonSheet = graphics.newImageSheet( "widget-button-file.png", options )

-- Create the widget
local button1 = widget.newButton(
    {
        sheet = buttonSheet,
        defaultFrame = 1,
        overFrame = 2,
        label = "button",
        onEvent = handleButtonEvent
    }
)

-- Center the button
button1.x = display.contentCenterX
button1.y = display.contentCenterY


-- Change the button's label text
button1:setLabel( "2-Frame" )
```

Button Options w/ Shape

- label = "button",
- onEvent = handleButtonEvent,
- -- Properties for a rounded rectangle button
- shape = "roundedRect",
- width = 200,
- height = 40,
- cornerRadius = 2,
- fillColor = { default={1,0,0,1}, over={1,0.1,0.7,0.4} },
- strokeColor = { default={1,0.4,0,1}, over={0.8,0.8,1,1} },
- strokeWidth = 4



Example: Button Shape

```
local widget = require( "widget" )
-- Function to handle button events
local function handleButtonEvent( event )

    if ( "ended" == event.phase ) then
        print( "Button was pressed and released" )
    end
end
-- Create the widget
local button1 = widget.newButton(
    {
        label = "button",
        onEvent = handleButtonEvent,
        emboss = false,
        -- Properties for a rounded rectangle button
        shape = "roundedRect",
        width = 200,
        height = 40,
        cornerRadius = 2,
        fillColor = { default={1,0,0,1}, over={1,0.1,0.7,0.4} },
        strokeColor = { default={1,0.4,0,1}, over={0.8,0.8,1,1} },
        strokeWidth = 4
    }
)
-- Center the button
button1.x = display.contentCenterX
button1.y = display.contentCenterY

-- Change the button's label text
button1:setLabel( "Shape" )
```

Button and Animate

```
local opt =
{
        frames = {
                { x = 0, y = 0, width = 16, height = 32}, --frame 1
                { x = 16, y = 0, width = 16, height = 32}, --frame 2
                { x = 32, y = 0, width = 16, height = 32}, --frame 3
                { x = 48, y = 0, width = 16, height = 32}, --frame 4
                { x = 64, y = 0, width = 16, height = 32}, --frame
                { x = 80, y = 0, width = 16, height = 32}, --frame 4
                { x = 96, y = 0, width = 16, height = 32}, --frame 4
                { x = 112, y = 0, width = 16, height = 32}, --frame 4
                { x = 128, y = 0, width = 16, height = 32}, --frame 4
                { x = 144, y = 0, width = 16, height = 32}, --frame 4
                { x = 160, y = 0, width = 16, height = 32}, --frame 4
        }
}
local sheet = graphics.newImageSheet( "mario.png", opt);
```

```
local seqData = {
        {name = "normal", start=2 ,count = 3, time=800},
        {name = "faster", frames={1,2,3,4, 5}, time = 800},
}
local anim = display.newSprite (sheet, seqData);
anim.anchorX = 0.5;
anim.anchorY = 0.5;
anim.x = display.contentCenterX;
anim.y = 160 --display.contentCenterY+45;
anim.xScale = 3;
anim.yScale = 3;
anim:setSequence("normal");
-- default: will play the first seq listed in seqData
anim:play();
```

Button and Animate (continued)

```
-- Function to handle button events
local function handleButtonEvent( event )
        if (event.phase == "ended") then
                if(anim.isPlaying) then
                        anim:pause();
                else
                        anim:play();
                end
        end
end
```

```
local widget = require( "widget" )
local button1 = widget.newButton(
{
x = display.contentCenterX,
y = 40,
id = "button1",
label = "Go!",
labelColor = { default={ 0, 0, 0 }, over={ 1, 1, 0} },
onEvent = handleButtonEvent,
width = 120,
height = 120,
defaultFile = "sbuttonDefault.png",
overFile = "sbuttonOver.png"
}
);
```

Progress View

Progress View

- widget.newProgressView(***options***)
- ***options***:
  - x & y or left & top
  - Width
- It can be also created through ImageSheet (https://docs.coronalabs.com/api/library/widget/newProgressView.html)

Progress View Default

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

-- Create the widget
local progressView = widget.newProgressView(
    {
        left = 50,
        top = 200,
        width = 220,
        isAnimated = true
    }
)

-- Set the progress to 50%
progressView:setProgress( 0.5 )
```

Scroll View

Scroll View—Basic Options

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

```
local scrollView = widget.newScrollView(
    {
        top = 0,
        left = 0,
        width = display.contentWidth,
        height = display.contentHeight,
        scrollWidth = 50,
        scrollHeight = 50,
        listener = scrollListener
    }
)
local background = display.newImageRect( "street.png", 750, 1167 )
background.anchorX = 0
background.anchorY = 0
scrollView:insert( background )
```

- Other options:
  - friction (real number)
  - horizontalScrollDisabled/ verticalScrollDisabled (true/false)
  - isLocked (true/false)
  - isBounceEnabled (true/false)

Basic ScrollView - Image

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

-- ScrollView listener
local function scrollListener( event )

    local phase = event.phase
    if ( phase == "began" ) then print( "Scroll view was touched" )
    elseif ( phase == "moved" ) then print( "Scroll view was moved" )
    elseif ( phase == "ended" ) then print( "Scroll view was released" )
    end

    -- In the event a scroll limit is reached...
    if ( event.limitReached ) then
        if ( event.direction == "up" ) then print( "Reached bottom limit" )
        elseif ( event.direction == "down" ) then print( "Reached top limit" )
        elseif ( event.direction == "left" ) then print( "Reached right limit" )
        elseif ( event.direction == "right" ) then print( "Reached left limit" )
        end
    end

    return true
end
```

```
-- Create the widget
local scrollView = widget.newScrollView(
    {
        top = 0,
        left = 0,
        width = display.contentWidth,
        height = display.contentHeight,
        scrollWidth = 50,
        scrollHeight = 50,
        listener = scrollListener
    }
)

-- Create a image and insert it into the scroll view
local background = display.newImageRect( "street.png", 768, 1024 )
background.anchorX = 0
background.anchorY = 0
scrollView:insert( background )
```

Basic ScrollView - Text

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

-- ScrollView listener
local function scrollListener( event )
    local phase = event.phase
    if ( phase == "began" ) then print( "Scroll view was touched" )
    elseif ( phase == "moved" ) then print( "Scroll view was moved" )
    elseif ( phase == "ended" ) then print( "Scroll view was released" )
    end
    -- In the event a scroll limit is reached...
    if ( event.limitReached ) then
        if ( event.direction == "up" ) then print( "Reached bottom limit" )
        elseif ( event.direction == "down" ) then print( "Reached top limit" )
        elseif ( event.direction == "left" ) then print( "Reached right limit" )
        elseif ( event.direction == "right" ) then print( "Reached left limit" )
        end
    end
    return true
end
```

```
-- Create the widget
local scrollView = widget.newScrollView(
    {
        top = 0,
        left = 0,
        width = display.contentWidth,
        height = display.contentHeight,
        scrollWidth = 50,
        scrollHeight = 50,
        listener = scrollListener
    }
)
```

```
-- Create a long text and insert it into the scroll view
Local lotsOfText = "Contrary to popular belief, ... "
local lotsOfTextObject= display.newText ( lotsOfText, 0, 0, 300, 0, "Times Roman", 14 )
lotsOfTextObject:setTextColor(0)
lotsOfTextObject.anchorX = 0
lotsOfTextObject.anchorY =0
scrollView:insert(lotsOfTextObject)
```

Scroll View—Visual Options

- Some Visual Options:
  - backgroundColor
  - hideBackground
  - hideScrollBar
  - scrollBarOptions
    - scrollBarOptions = { sheet = scrollBarSheet, -- Reference to the image sheet

      topFrame = 1, -- Number of the "top" frame

      middleFrame = 2, -- Number of the "middle" frame

      bottomFrame = 3 -- Number of the "bottom" frame }