School of Science

CSP2108 Introduction to Mobile Application Development

Workshop: Using strings to display device info

In this workshop you will use strings to create messages to display on the mobile device. You'll also more about display coordinates, and how to get information about the device using Corona's libraries. Lastly, you'll learn one way to install an app that you have built onto a mobile device.

Instructions

1. Create a new Corona project for this week, calling it *DisplayStrings*. We are going to build up and test this program a little at a time (a good way to write a complicated program).

2. Edit this project's *main.lua* so that it looks like this:

```
-----------------------------------------------------------------------------------------
--
-- main.lua
--
-- Tutorial program for displaying information about
-- the mobile device that we are running on
-- CSP2308 Workshop 2
--  Author: phi
--  Date: May 2016
--
-----------------------------------------------------------------------------------------

FONT = "Arial"
TEXT_SIZE = 20

-- put a mark in the middle of the screen
display.newText("x", display.contentCenterX, display.contentCenterY,
                              FONT, TEXT_SIZE)
```

Obviously put in your own name as the Author and adjust the Date comment if you need to. It is good practice to use comments like this to describe the purpose of a program source file, and add authorship information.

A couple of other things to note:

FONT and *TEXT_SIZE* are constants, which in *Lua* are just the same as variables. To show that they are intended to be constants (i.e. their initial values will not be changed), it is good style to use ALL_CAPS with optional underscores.

Later in this program, more text will be displayed using the same font and font size. It is better programming style to use constants like this than to repeat the string "Arial" and the number 20 throughout the program. One reason is that if you want to change the font or font size for some reason, there is only one place they need to be changed.

Next, notice the x-coordinate of the displayed text is *display.contentCenterX*. Since you now know about tables, you can recognize that *display* is a table (it is also a library) and *contentCenterX* is a key for that table. *display.contentCenterX* is a value stored with that key.

To find out about the *display* library, you can read about the *Lua* libraries in the Corona SDK at https://docs.coronalabs.com/api/library/index.html . This is a handy reference to know about.

The display library has a number of properties, like *display.contentCenterX*, as well as functions, like *display.newText()*.

display.contentCenterX gives the x-coordinate of the middle of the display area of the mobile device's display screen. Likewise *display.contentCenterY* gives the y-coordinate. So the program above places an "x" right in the middle of the screen.

3. Test the program (save *main.lua* and relaunch the app) and correct any errors.

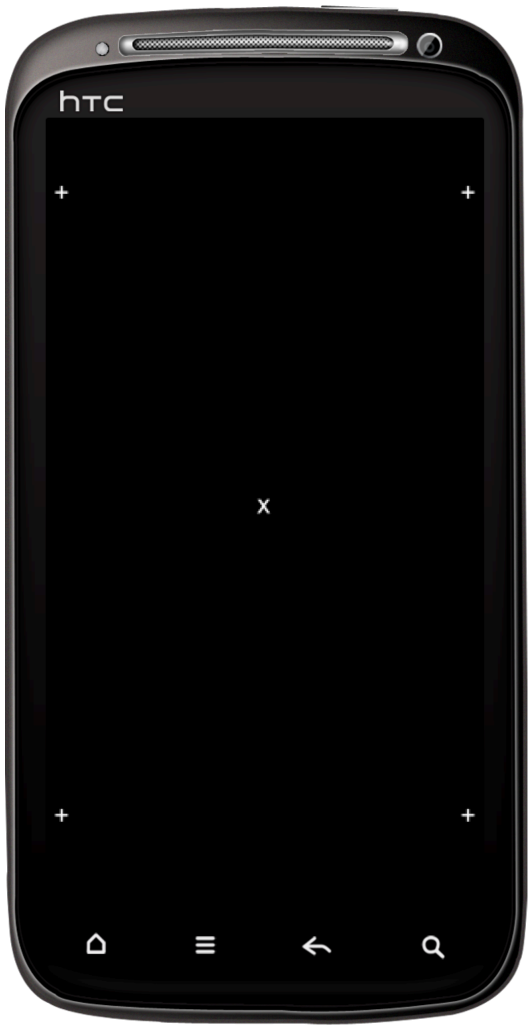
4. Add 4 more *display.newText()* function calls, placing a "+" near each corner of the display area. For example:

```
-- place a mark in each corner
display.newText("+", 10, 10, FONT, TEXT_SIZE)
-- add 3 more lines
```

will place a "+" 10 units from the left and 10 from the top of the display area.

Hint: you will have to use *display.contentWidth* and *display.contentHeight*, which contain the width and height of the display area of the screen, to figure out the right x and y-coordinates.

Your simulator window should end up looking something like this:

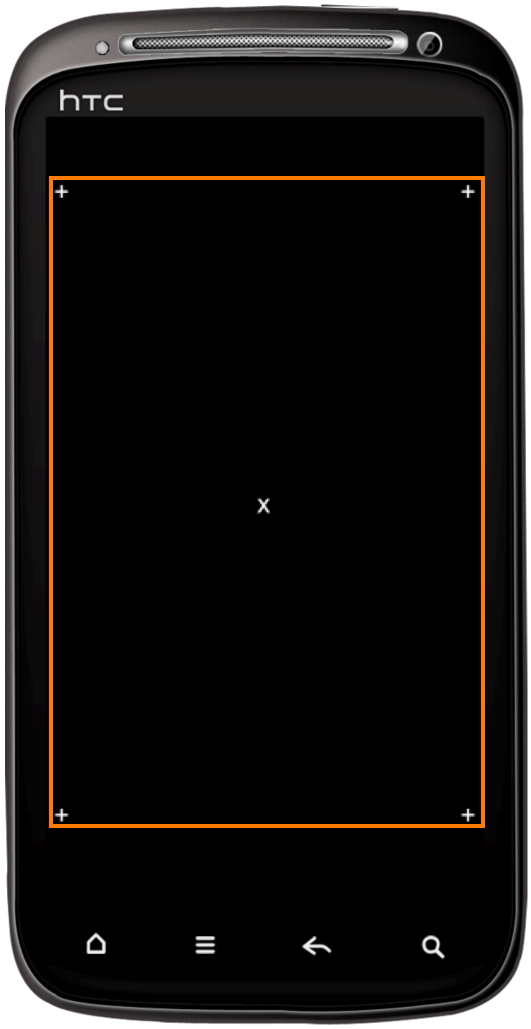


Notice that there is extra space at the top and the bottom of the display, so the top-left "+" sign appears to be too low and the bottom left one too high. This is because of the way Corona deals with different screen sizes on different devices.

In fact, *display.contentWidth* and *display.contentHeight* may not give you the actual width and height of the display. This sounds like a problem, but in fact it is a reasonable price to pay, so that you don't have to write special code for every possible screen size for every possible device.

Instead, you write code that assumes *display.contentWidth* and *display.contentHeight* are the correct width and height, and let the Corona libraries do scaling and make adjustments (like leaving a bit of blank space at the top and bottom) so that the result is acceptable on different screens.

To get the actual width and height, there are two extra properties, *display.actualContentWidth* and *display.actualContentHeight*. They work like this:



The orange rectangle above shows the display area that we draw into. The width of the rectangle is *display.contentWidth* and the height is *display.contentHeight*. When we draw text (like the "+") at location (10, 10) it will be drawn 10 units from the left and 10 units from the top of the orange rectangle. The actual height of the screen is a bit larger in this case, so *display.actualContentHeight* is different to *display.contentHeight*.

5. Let's find out what the actual width and height of the display are. We can print this information on the screen with the following code:

```
-- display info on display coordinates

currentY = 20 -- y coordinate for next text object to display

display.newText("Display:", display.contentCenterX, currentY, FONT, TEXT_SIZE)
currentY = currentY + TEXT_SIZE

-- these are the width and height set in config.lua
widthMessage = "W x H = "..display.contentWidth.."x"..display.contentHeight
display.newText(widthMessage, display.contentCenterX, currentY, FONT, TEXT_SIZE)
currentY = currentY + TEXT_SIZE

actualWidth = string.format("%.1f", display.actualContentWidth)
actualHeight = string.format("%.1f", display.actualContentHeight)
actualWidthMessage = "actual W x H = "..actualWidth.."x"..actualHeight

display.newText(actualWidthMessage, display.contentCenterX, currentY, FONT,
TEXT_SIZE)
currentY = currentY + TEXT_SIZE

aspectRatio = display.pixelHeight/display.pixelWidth
aspectRatioString = string.format("%.2f", aspectRatio)
display.newText("aspectRatio = "..aspectRatioString, display.contentCenterX, currentY,
FONT, TEXT_SIZE)
currentY = currentY + TEXT_SIZE
```

Add this code to the bottom of *main.lua*, save and relaunch. You should see something like this:

Let's review this code:

```
display.newText("Display:", display.contentCenterX, currentY, FONT, TEXT_SIZE)
currentY = currentY + TEXT_SIZE
```

The first line above just shows the heading "Display". We use the variable *currentY* to keep track of how far down the screen the next text item should be drawn, so we increase it by *TEXT_SIZE* after each time we draw some text.

Next we have:

```
-- these are the width and height set in config.lua
widthMessage = "W x H = "..display.contentWidth.."x"..display.contentHeight
display.newText(widthMessage, display.contentCenterX, currentY, FONT, TEXT_SIZE)
currentY = currentY + TEXT_SIZE
```

Here the second line is used to construct the text string that we want to display. Note the use of the concatenation operator (..) to join bits of text together.

Next:

```
actualWidth = string.format("%.1f", display.actualContentWidth)
actualHeight = string.format("%.1f", display.actualContentHeight)
actualWidthMessage = "actual W x H = "..actualWidth.."x"..actualHeight
```

Because *display.actualContentWidth* may not be a whole number, we are using the *string.format()* function from the *Lua* libraries to create a text string with 1 decimal place (otherwise we would get some default number of decimals, which may look ugly). You can read about the *string* library here http://lua-users.org/wiki/StringLibraryTutorial for example.

Finally, what about this bit:

```
aspectRatio = display.pixelHeight/display.pixelWidth
aspectRatioString = string.format("%.2f", aspectRatio)
display.newText("aspectRatio = "..aspectRatioString, display.contentCenterX, currentY,
FONT, TEXT_SIZE)
currentY = currentY + TEXT_SIZE
```

Here we are using *display.pixelHeight* and *display.pixelWidth* (you can read about these in the Corona online docs) to calculate the aspect ratio of the pixels on this display device.

6. See if you can add some more information yourself. Try to add some more code to *main.lua* to get this result:

Here we have used the *system* library to find out and display the *appName* (name of the app), *architecture* and *model* (of the device). Hint: use the *system.getInfo()* function, described on https://docs.coronalabs.com/api/library/system/getInfo.html .

7. The Corona simulator can simulate different devices. You can switch to another one using the Window/View As.. menu item in the Corona Simulator window. For example, here is what you get if you switch to the Samsung Galaxy 5:

Note that the *model* information has changed. If you run this app on an actual device, the *architectureInfo* value will also change (in the picture above it is x86-64 because I ran the app in the Corona Simulator on my Macbook Pro).

8. Let's try this app on a real device! To do so, you have to "build" the app, and transfer it to the device. In the Corona Simulator, use the File/Build menu item, and select Android. A "Build for Android" window will appear. Fill in the entries as described in your textbook on page 34.

Since your textbook has been written, a set of radio buttons labeled "After Build" has been added to the "Build for Android" window. Check "Copy to device and launch". Now connect your device to the computer via the USB connection, and click on "Build".

The next part might take a while : the computer will attempt to communicate with Corona Labs. It will send your source code, and if the build is successful, the compiled app will be sent back to the computer. This can be slow, and it may even fail with a network timeout. If so, try again.

Eventually your build should succeed, and the app will then be loaded onto the mobile device and launched, if everything goes right!

Another possibility is to choose “Show in finder” instead of “Copy to device and launch”. The build will go as before, but the built app (called *DisplayStrings.apk*) will be save on your computer. You can transfer this file to your device later, by a number of different methods. One way is to email the *.apk* file to yourself and read the mail on your mobile device. The mail app should know how to install your app.