Scripting Tutorial - Lesson 7: Quick Start: Working with Images

Download supporting files for this tutorial

Texas Instruments TI-Nspire Scripting Support Page

Lua has quite powerful but simple image manipulation commands, making it easy to import images into your Lua document and to actually "play" with these. While TI-Nspire 3.0 supports inserting images into TI-Nspire documents, there are as yet no options for interacting with these - making them change size, or even appear and disappear on the fly. This functionality is available using Lua.

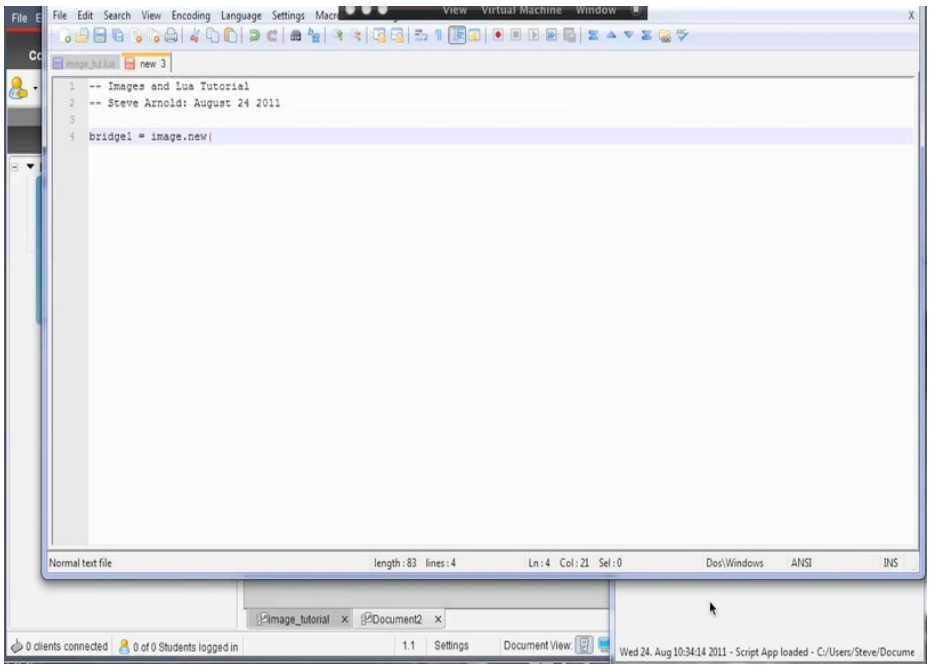
Lesson 7.1: Setting up and Displaying your Image

The first step in importing an image into Lua is to digitize it - to turn it into a series of characters which can be manipulated digitally. This may be done using TI's own Scripting Tools application, available from the link above. The accompanying video shows the steps in this process. Another option is to use a wonderful online image converter that will do the same job.

Click anywhere on this image for a video demonstration

As shown, these methods take an image (in most common formats), convert it and (in

the case of the TI Scripting Tool) place the digital code on the clipboard. This may then be pasted into your Lua script document. Note the way in which this digital code is defined as a variable in Lua using the **image.new** command.



bridge1 = image.new("..digital code for image..")

Now all we have to do is to tell the script to "paint" the image to the screen, using the now-familiar **on.paint** function, along with the simple **drawImage** command. If we have defined the image as a variable called **bridge1**, then displaying it is achieved using the code shown here:

```
function on.paint(gc)
    gc:drawImage(bridge1, 0, 0)
end
```

That is actually the heavy lifting done. Everything else just builds on previous lessons to "tweak" our image display – to ensure that it fits the window, for example, to center it in that window, and even to control scale and to change images dynamically!

Lesson 7.2: Fitting our Image to the Window

We encountered the window dimension commands way back in tutorial 1: **platform.window:width** and **platform.window:height.** In our example, these may be used to center the

```
function on.paint(gc)
    local w = platform.window:width()
    local h = platform.window:height()
```

image on the window, along with the equivalent dimension commands for images: **image.width(*image*)** and **image.height(*image*)**. Study the code and make sure everything makes sense.

Note particularly the way that the image is centered on the window.

```
        local imw =
        image.width(bridge1)

        local imh =
        image.height(bridge1)

        gc:drawImage(bridge1,
        (w - imw)/2, (h - imh)/2)

end
```

This is great if our image fits reasonably well to the window. But what if it is too big, or too small? For this we create a copy of the image that is scaled. **image.copy(*my_image*, scaleX * image.width(my_image), scaleY * image.height(my_image))** as implied creates a copy of the original image with the option for different scales for x and y dimensions. For our example, we will keep the same aspect ratio as the original image and define a single scale factor.

Note the double use of imw and imh: first for our original image (**bridge1**) and then again for the scaled version.

```
function on.paint(gc)

        local w =
        platform.window:width()

        local h =
        platform.window:height()

        local sc = 0.5

        local imw =
        image.width(bridge1)

        local imh =
        image.height(bridge1)

        local im =
        image.copy(bridge1, sc *
        imw, sc * imh)

        local imw =
        image.width(im)

        local imh =
        image.height(im)

        gc:drawImage(im, (w -
        imw)/2, (h - imh)/2)

end
```

Lesson 7.3: Making our Image Dynamic

So far so good. You could stop here and be able to insert an image and play around with the scale until it fits nicely. Nicer, though, would be to be able to change that scale on the fly – perhaps by using the arrow keys as we learned in the last lesson!

Initially it is a good idea to create the

```
function on.create()

        timer.start(1/5)

end

function on.timer()

        platform.window:invalidate()

end
```

variable that we will use in our TI-Nspire document - I normally insert a Geometry window and hide the scale. Then insert a slider called **scale**, set to run between 0 and 1 in steps of 0.1. We can probably dispose of this later since we will control the values using arrow keys - but if you wanted the document to be workable within the Player, then the arrow keys will be of no use and a slider would not go astray!
These functions can be defined prior to the **on.paint** function. As you can see, they check to see the value of the variable **scale** from the TI-Nspire symbol table (if it does not exist then the value is set to 0.5). With each press of an arrow key, then, this value is changed by 0.1, either up or down, and this new value is **stored** in the value of **scale** back in TI-Nspire.

Note that the only change to the **on.paint** function is to grab the current value of **scale** and define this as our new Lua variable **sc**. And it never hurts to refresh the screen after each change - hence the **platform.invalidate** commands!

```
function on.arrowUp()
    sc = (var.recall("scale") or
    0.5)
    var.store("scale", sc + 0.1)
end
function on.arrowDown()
    sc = (var.recall("scale") or
    0.5)
    var.store("scale", sc - 0.1)
end
```

```
function on.paint(gc)
    local w =
    platform.window:width()
    local h =
    platform.window:height()
    sc = (var.recall("scale") or
    0.5)
    imw = image.width(bridge1)
    imh =
    image.height(bridge1)
    im = image.copy(bridge1,
    sc * imw, sc * imh)
    imw = image.width(im)
    imh = image.height(im)
    gc:drawImage(im, (w -
    imw)/2, (h - imh)/2)
end
```

Now you can control the scale using the up and down arrow keys!

Lesson 7.4: Swapping Images on the Fly

Our final tweak is to define several images and use the left and right arrow keys to switch between these! back to the Scripting Tool and digitize a couple of other images – I am using the stock bridge images that are installed in your TI-Nspire Images folder when 3.0 is installed. I define these as **bridge2** and **bridge3**.

Create a slider variable, called **choose** in your Geometry window, and set it to run from 1 to 3.

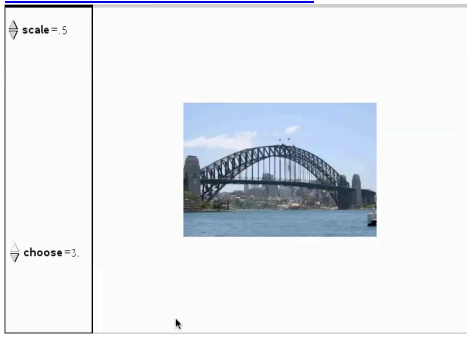
As we just did for the scale, we define the left and right arrow keys to increment the values of our variable **choose** and refer to it in the Lua script as **ch**.

```
function on.create()
    timer.start(1/5)
end
function on.timer()
    platform.window:invalidate()
end
function on.arrowLeft()
    ch = (var.recall("choose") or 1)
    var.store("choose", ch - 1)
    platform.window:invalidate()
end
function on.arrowRight()
    ch = (var.recall("choose") or 1)
    var.store("choose", ch + 1)
    platform.window:invalidate()
end
```

Within our paint function, then, once again recall the current value of **choose** and then define the image using an if..then..elseif..end loop.

You should now be able to switch between images using the left and right arrow keys!

Click anywhere on this image to view a short video demonstration




```
function on.paint(gc)
    local W = platform.window:width()
    local H = platform.window:height()
    sc = (var.recall("scale") or 0.5)
    ch = (var.recall("choose") or 1)
    if ch == 1 then
        show = bridge1
    elseif ch == 2 then
        show = bridge2
    else
        show = bridge3
    end
    local imw = image.width(show)
    local imh =
```

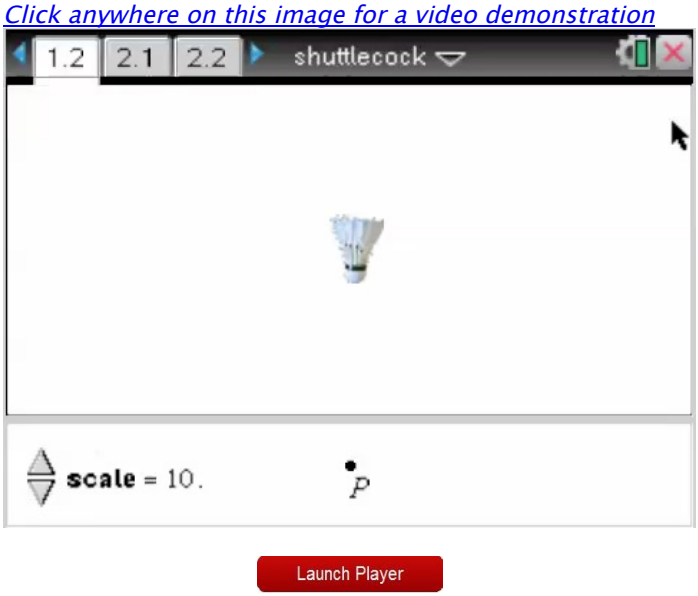
```
image.height(show)

local im =
image.copy(show, Scale *
imw, Scale * imh)

local imw = image.width(im)

local imh =
image.height(im)

gc:drawImage(im, (W -
imw)/2, (H - imh)/2)

platform.window:invalidate()

end
```

Congratulations! You now have all the ingredients you need to digitize, define, insert, scale and manipulate images using Lua. Our final lessons in this sequence will look at creating your own graphics using Lua.

Lesson 7.5: Controlling the Image Position

What if you want to move the image around on the screen? While this can be done entirely within Lua using the arrow commands as described previously, another neat option might be to split the window, place a Graph window underneath, and control the image by moving a point around on the Graph window. To do this, store the x- and y-coordinates of the point (say **px** and **py**) and then use these with your arrow commands in Lua to move the image.

Click anywhere on this image for a video demonstration



This is demonstrated in the **shuttlecock.tns** file that is included with this lesson's downloads. Included too you will find the image that was used and the Lua script for you to study. In problem 2 of this TNS file, the point P is linked to a parabola so that the motion of the shuttlecock can be modelled and even

animated!

NOTE: One advantage of adding sliders and draggable points is that the document is then able to be used with the Player. If we just rely on arrow keys and keyboard controls, then the Player is not an option.

Lesson 7.6: A Simpler Approach

Using arrow keys and general keyboard controls (including enterKey, escapeKey and tabKey) are great ways to ensure that a document you create will work seamlessly on the handheld. I cannot stress this enough, because it may be a simple thing but it is a significant way in which Lua-based documents can be written to be much more user-friendly than "native" documents.

Even though the applications shown in the lessons all involve using the arrow keys to grab a variable from TI-Nspire, make some change and then store back to that variable, this is not the only way to set up keyboard controls. In fact, it is much simpler to just use Lua's own variables and not have to reply on "outside" ones at all.

Suppose you have created the digital versions of the three bridge images referred to in Lesson 7. so at the start of our document, we have definitions for bridge1, bridge2 and

```
function on.resize()
    W = platform.window:width()
    H = platform.window:height()
    Scale = 1
    Choose = 1
end
function on.arrowUp()
    Choose = Choose + 1
end
function on.arrowDown()
    Choose = Choose - 1
end
```

bridge3.

Then we could do the following for the remainder of our script:

Define two variables for scale and choose: these days I tend to do this in a "resize" function, since this always gets called when the page is created, and also when any change in size occurs – like switching between handheld and computer view. Here I am going to start a good habit of naming Global variables with a Capital letter.

Do you see how much simpler this approach is?

So why go to all the trouble of using var.recall and var.store to move variables back and forth between Lua and TI-Nspire? One reason is that, while keyboard controls are great for the handheld, they are actually useless if you want your document to run on the Player, since it only supports grabbing and dragging things. No keyboard input.

So for the moment, we are leaving TI-Nspire sliders in place and linking to them as well as adding keyboard commands. This allows our document to be used anywhere.

Later (lessons 11-15) you will learn how to enable mouse controls, so that you can create your own controls and use these instead of TI-Nspire sliders. Put these together with our keyboard controls that you have just learned and you have an optimal document. Have a look at the sample documents on the Scripting HQ page

```
function on.arrowLeft()
    Scale = Scale - 0.1
end
function on.arrowRight()
    Scale = Scale + 0.1
end
function on.paint(gc)
    if Choose == 1 then
        show =
        bridge1
    elseif Choose == 2 then
        show =
        bridge2
    else
        show =
        bridge3
    end
    local imw =
    image.width(show)
    local imh =
    image.height(show)
    local im =
    image.copy(show, Scale *
    imw, Scale * imh)
    local imw = image.width(im)
    local imh =
    image.height(im)
    gc:drawImage(im, (W -
    imw)/2, (H - imh)/2)
    platform.window:invalidate()
end
```

where you access these tutorials and you will see examples of exactly this.

Next we learn how to use Lua's own graphics capabilities to create our own images.