Scripting Tutorial - Lesson 22: (3.2) Create Your Own Math Boxes

Download supporting files for this tutorial

Download this page in PDF format

Texas Instruments TI-Nspire Scripting Support Page

Create Your Own Math Boxes (3.2)

The 2D Editor certainly supports some exciting text-based opportunities, but for me, where it shines lies in the potential to make input and display of mathematical (and chemical) notation easy.

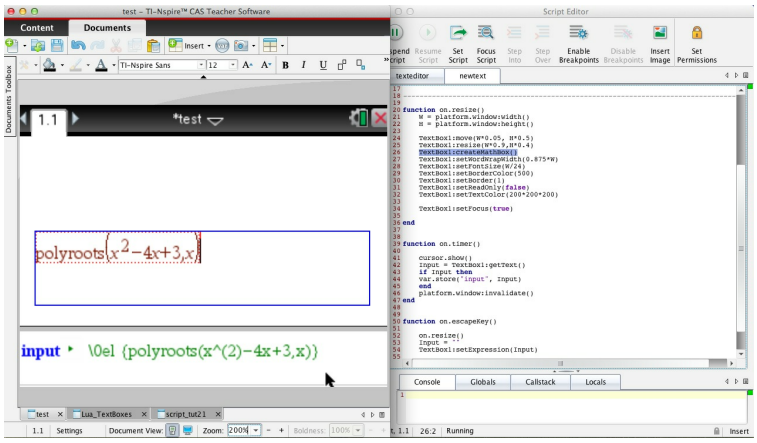
1. Setting Up Your MathBox or ChemBox

Top of Page

Begin with exactly the same script as in the last lesson - which creates an active text box and stores any input to a TI-Nspire variable called "input".

This time, however, add a single command: **TextBox1:createMathBox()** or **TextBox1:createChemBox()**. The magic is done!

Waiting for your input is a fresh math or chem box with cursor flashing, ready to go!



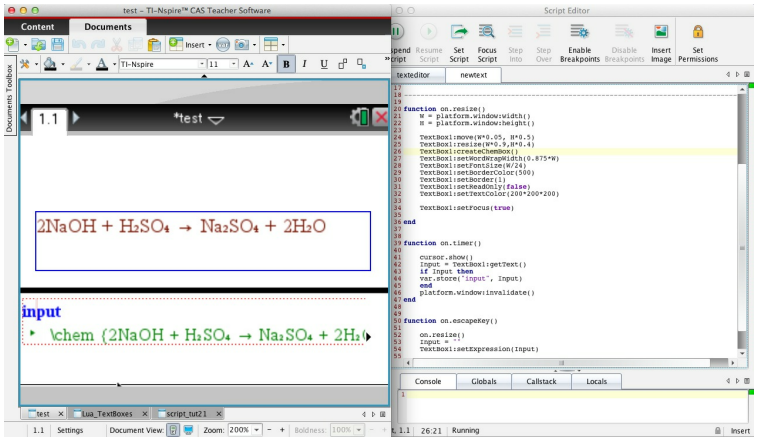

```
platform.apilevel = "2.0"

function on.construction()
	timer.start(0.4)
	TextBox1 = D2Editor.newRichText()
end

function on.resize()
	W = platform.window:width()
	H = platform.window:height()
	TextBox1:move(W*0.05, H*0.5)
	TextBox1:resize(W*0.9,H*0.4)
	TextBox1:createMathBox()
	fontSize = math.floor(W/32)
	fontSize = fontSize > 6 and fontSize
	or 7
	TextBox1:setFontSize(fontSize)
	TextBox1:setBorderColor(500)
	TextBox1:setBorder(1)
	TextBox1:setTextColor(200*200*200)
	TextBox1:setFocus(true)
end

function on.getFocus()
	TextBox1:setFocus()
end

function on.timer()
	cursor.show()
	Input = TextBox1:getExpression()
	if Input then
		var.store("input", Input)
```



```
        end
        platform.window:invalidate()
end
```

```
function on.escapeKey()
        TextBox1:setExpression(" ")
        platform.window:invalidate()
end
```

The chem box is basically just a presentation tool (which is great, of course) but the MathBox is live and fully functional. Type in a valid command, press enter and it evaluates – just like in Notes.

2. Playing with Math

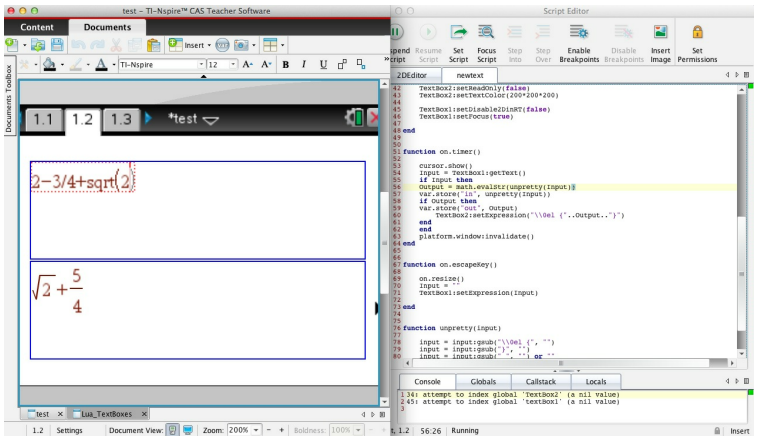
Top of Page

Look carefully at the output of each: the result is wrapped in a simple layer of code. For the MathBox, this is "\\0el {" my input and "}" while the ChemBox output is even simpler – "\\chem {" my input "}".

Hold on – can't see that extra "\" in the screen shots above? It is a reserved character in Lua and so to have it recognised, you need to add an extra one!! (Lucky I told you that, isn't it?)

So now for our challenge: once again, as shown, create two text boxes as you did in the last lesson. This time, we want whatever is typed into TextBox 1 to be evaluated, and the result displayed (correctly, of course!) in the second textBox.

Think about what will be needed for this to happen before moving on to the next section. You will need to **get** the expression from TextBox1. Yes, we can do that.



This expression will need to be evaluated – think back to lesson 17 and the **math.eval** command. But in this case, the output is likely to be a fraction, or an algebraic form, not just a string. So we will need a new 3.2 command – **math.evalStr**.

But hold on. The input is wrapped in that funny layer – TI-Nspire cannot evaluate it like that. So we need to strip away the wrapping – and for that we will define a neat little function which I call **unpretty**.

So strip away the wrapping and evaluate the result in Nspire to produce a variable we will call Output.

Now we wish to display this result in TextBox2. We can use the TextBox2:setExpression command. Easy.

But if we would like it to be presented in its own MathBox and correctly formatted, then we simply need to wrap that layer around it that we saw above. Pretty neat.

```
function unpretty(input)
        input = input:gsub("\\0el {", "")
        input = input:gsub("}", "")
        input = input:gsub(" ", "") or ""
        return input
end
```

```
function on.timer()
        cursor.show()
        Input = TextBox1:getExpression()
        if Input then
                Output =
                math.evalStr(unpretty(Input))
                if Output then
                TextBox2:setExpression("\\0el
                {"..Output.."}")
                end
        end
```

Once again so many possibilities present themselves.

We could easily create and correctly display questions in the first text box and have students enter answers

(also correctly presented) in the second.

We could create our own TI-Nspire-based calculator in Lua and try all sorts of neat things with the results, prototyping your own ideas for your students learning.

```
        platform.window:invalidate()

end
```

Whatever you choose to do, it seems to me to be so much easier than trying laboriously to create your own layout for fractions, algebraic terms and more using paint commands and strings. So have some fun!

Back to Top

Home ← TI-Nspire Authoring ← TI-Nspire Scripting HQ ← **Scripting Tutorial - Lesson 22**