

INTO THE new
BREW 2007 CONFERENCE

Trig Development with Lua

Kevin Hunter
Director, Technical Marketing
QUALCOMM Incorporated





Trig Development with Lua – Contents

- Why add a scripting language to uiOne™?
- Why did uiOne choose Lua?
- Adding Lua to the Trig
- The TrigML® Document Object Model
- Using timers in Lua
- Extensibility



Why add a Scripting Language?

- TrigML is very powerful, but...
 - Markup is great for describing *appearance*
 - For relatively straightforward UIs, markup works well for describing *behavior* too
 - But beyond a certain complexity, using markup to describe *all* behavior can feel cumbersome
 - > Challenging to avoid code duplication
 - > Difficult to cleanly implement state machines
 - > Hard to separate behavior from appearance
 - Previously, the remedy has been to embed complex application behavior logic in an Actor, requiring C coding skills



What does a Scripting Language bring?

- Another way to code the UI's behavior
 - Functions, if-else, loops, data-driven programming
 - > Fewer verbose TrigML expressions
 - Enhanced readability, maintainability
 - Let the mark-up concentrate on layout
- New ways to store and manipulate state
 - Sophisticated, efficient data structures
 - Avoid over-using /var
- Reduced need to put application logic in C-coded actors
 - Shorter development time, easier learning curve



Trig Development with Lua - Contents

- Why add a scripting language to uiOne?
- Why did uiOne choose Lua?
- Adding Lua to the Trig
- The TrigML Document Object Model
- Using timers in Lua
- Extensibility

What is Lua?

- Lua is a powerful, fast, light-weight, embeddable scripting language
 - Originated in 1993 at PUC-Rio, now developed by Lablua
 - Has gained widespread popularity as a general purpose scripting and “glue” language
 - A leading scripting language in the games industry
 - See www.lua.org



Getting familiar with Lua

```
function hello ()  
    print("Hello, world")  
end
```

```
function factorial (n)  
    local ret = 1  
    while n > 1 do  
        ret = ret * n  
        n = n - 1  
    end  
    return ret  
end
```



Why did uiOne choose Lua?

- Small
 - ILUA, the core runtime module, adds less than 100kbyte to the phone's code footprint
- Fast
 - Many benchmarks rate it as one of the fastest scripting languages available
- Well documented
- Already part of the BREW[®] world
 - ILUA v1.0 in use internally within QUALCOMM
 - ILUA v1.1 will ship as part of uiOne SDK 2.0
 - > Based on Lua 5.1
 - > Wide string support



Some Advantages of Lua

- Shares many features familiar from other scripting languages
 - Dynamically typed, lexically scoped
 - Garbage collection
 - All values are first-class citizens
 - > Functions can be returned by functions, stored in variables
 - Basic data types (boolean, string, number, etc)
 - One powerful way to structure data: the *table*
 - > An associative array with optimizations for numeric indexes
 - > Used for arrays, maps, sets, trees, structures, classes...



Trig Development with Lua - Contents

- Why add a scripting language to uiOne?
- Why did uiOne choose Lua?
- Adding Lua to the Trig
- The TrigML Document Object Model
- Using timers in Lua
- Extensibility



Adding Lua to the Trig

- New script resource type in Trigbuilder
 - Lua functions are entered into script resources
 - Syntax-highlighting Lua editor pane
 - Easy to write and edit Lua code without using a separate editor
- New TrigML `<import>` element
 - `<import res="script/myfuncs"/>`
 - Loads a Lua script from a VFS path
 - Automatically prevents duplicate imports
 - The Lua script can define functions, initialize variables, load extensions, etc



Initializing Attribute Values with Lua

- Lua function calls in TrigML attributes
 - Evaluated once, when the element is loaded
 - The return value of the function becomes the initial value for the attribute

```
<text text="myFunction()" .../>
```

```
...
```

```
function myFunction()  
    return "Hello BREW 2007 Delegates"  
end
```



Initializing Attribute Values with Lua (cont.)

- Notes
 - The function may perform arbitrary computations and refer to all accessible Lua variables
 - It may inspect existing elements in the tree
 - > Example: base the characteristics of a new element on an existing one
 - It may *not* inspect itself or other elements that are being loaded during the current <load> operation
 - > Their attributes haven't yet been resolved or their layout computed



The “ui” Table

- All Trig state is accessed by Lua scripts through “*ui*.”
 - The “ui” table is the main global Lua object provided by TrigPlayer
 - Available to all Lua functions executed on behalf of the Trig
 - Contains data items associated with the Trig’s current state
 - Provides functions to manipulate the Trig’s state



Using Lua to Respond to Events

- New listening element `<call>`
 - Causes a Lua function to be run in response to a TrigML event.
 - The function can alter existing elements, throw further events, load new TrigML fragments etc
 - Anything a combination of `<load>` `<throw>` `<att>` and `<anim>` elements could do – and more besides!

```
<text text="Hello" ...>  
  <call when="_key" exec="changeText()"/>  
</text>
```

...

```
function changeText()  
  ui.this.text = "Goodbye"  
end
```



During the Event Handler

- “ui.this” is the visible parent of the <call>
 - Changing attributes of the “this” element is easy: assign new values to ui.this.x, ui.this.text, ui.this.res, etc
- “ui.focus” is the current focus element
 - ui.focus.bgcolor = “#FF0000”
- “ui.event” contains the event type and parameters
 - Possible to handle multiple event/parameter combinations with just one <call> element
 - Reduces need for multiple listeners

Using <call> for Fewer Listeners

```
<group ...>  
  <att when="_key[_key1]" name="bgcolor" value="#FF0000"/>  
  <att when="_key[_key2]" name="bgcolor" value="#FF00FF"/>  
  <att when="_key[_key3]" name="bgcolor" value="#FFFF00"/>  
  <att when="_key[_key4]" name="bgcolor" value="#FFFFFF"/>  
  ...  
</group>
```



```
<group ...>  
  <call when="_key" exec="changeColor()"/>  
</group>
```

```
function changeColor()  
  ui.this.bgcolor = computeColorFromKeycode(ui.event.key)  
end
```



Example – Basics

In this example we will see the basics of adding Lua to a Trig

- *Adding a Lua script resource*
- *Controlling TrigML layout attributes with Lua*
- *Responding to events with the <call> element*



Trig Development with Lua - Contents

- Why add a scripting language to uiOne?
- Why did uiOne choose Lua?
- Adding Lua to the Trig
- The TrigML Document Object Model
- Using timers in Lua
- Extensibility



The TrigML Document Object Model

- An object tree that exposes the runtime TrigML element tree
 - One DOM object per active TrigML element
 - DOM objects are inserted into and removed from the tree by <load>
 - DOM objects are exposed to Lua as tables with field names corresponding to TrigML attribute names
 - > x, y, res, frames, visible, animate, etc
 - The script can assign new values to DOM attributes to alter the appearance of the element



More Ways to Access DOM Objects

- We've seen `ui.this` and `ui.focus...`
 - `ui.element(id)`
 - > Look up a named element to obtain its DOM object
 - > The name is the "id" attribute of the element in TrigML
 - > `ui.element("popup").visible = true`
 - Functions to explicitly walk the DOM tree
 - > Get first child
 - > Get sibling
 - > Get parent



Trig Control

- Control the UI flow from Lua script
 - ui.load (target, path [, parameters])
 - > Load a TrigML fragment - equivalent to <load>
 - > Target can be an element ID or an element's DOM object
 - > `ui.load("popupLayer", "trigml/alerts/confirmExit")`
 - ui.throw (target, event [, parameters])
 - > Throw a TrigML event – equivalent to <throw>
 - > Target can be an element ID, an element's DOM object or a VFS path
 - > Parameters passed in a Lua table
 - > `ui.throw ("/actor/clock",
"setTime",
{time="13:45:00", date="6/6/2007"})`



Virtual Filesystem Access

- Lua API to the VFS, including
 - `ui.vfsRead(path)`
 - > Read the resource from the specified VFS path
 - > `myAppName = ui.vfsRead("text/appName")`
 - `ui.vfsWrite(path, value [, persist])`
 - > Write the resource at the specified VFS path
 - > `ui.vfsWrite("/var/name", "John", false)`
 - `ui.vfsObserve(path, function)`
 - > Call the function when the resource changes
 - > `ui.vfsObserve("/actor/clock/time", myClockCallback)`
 - `ui.exists(path)`
 - > Test whether a resource exists at the given path



Trig Development with Lua - Contents

- Why add a scripting language to uiOne?
- Why did uiOne choose Lua?
- Adding Lua to the Trig
- The TrigML Document Object Model
- Using timers in Lua
- Extensibility



Using Timers in Lua

- Drive timed aspects of your UI from Lua scripts
 - `ui.setTimer(myTimerCallback, timeout)`
 - > Request that a Lua function be called after *timeout* milliseconds
 - `ui.cancelTimer()`
 - > Cancel a pending timer
 - `ui.getCurrentTime()`
 - > Read the current millisecond resolution time
 - > Use it to lock your animation to real-time



Using timers in Lua

- Useful for implementing complex animation effects
 - The timer callback can adjust one or more attributes of one or many elements
 - *Much* more powerful than <anim> !
 - > Alter several things in one place
 - > Avoid pre-computed lists of co-ordinates
 - > Synchronize the animation of multiple elements
 - > Easily model gravity, bouncing, acceleration etc.



Example – Timers

In this example we will use a Lua timer to add dynamic behavior to a Trig

- *Setting a timer*
- *The timer callback*



Trig Development with Lua - Contents

- Why add a scripting language to uiOne?
- Why did uiOne choose Lua?
- Adding Lua to the Trig
- The TrigML Document Object Model
- Using timers in Lua
- Extensibility



Extensibility

- Packages
 - Lua code bundles
 - Namespacing, implementation hiding
 - Single import
 - > `require("mypackage")`
- Libraries
 - C code libraries linked as BREW modules
 - Implement a Lua veneer to an existing C API
 - Implement speed-critical functions



Functionality Provided as Extensions

- File System Access
 - Create and remove directories
 - Copy, move and delete files
- Networking
 - HTTP transactions
 - Network utilities
 - > URL encode/decode
- Utilities
 - Base64 encode/decode; compute SHA-1 hash
 - Zip and gzip file decompression
 - Get/set device config items
 - Send/Post URL



Example – HTTP

In this example we will see how Lua extensions make it easy to connect your Trig to the network

- *Loading the “Inet” extension library*
- *Making an HTTP request*
- *Reading the HTTP response*
- *Displaying the received data in the UI*



Thanks for listening!



Any Questions?