# Dark Lua

## *Introduction*

DBPro Lua implementation now has separate states that can be used for each script.
This means globals will be on script scope basis (as in they are not shared between scopes, so a global *Health* in one state is different to a *Health* in another state).

You can also use a state like a single scope, using Load Lua on the same state to retain previous globals that exist - so you can choose to use Dark Lua like unity, or utilise multiple scopes.

*Pros of Multi State:*
You can assign the same AI script to 50 enemies and their data will persist in Lua and not get overwritten by someone elses. Also the script does not need to be reopened constantly, just loaded and left open as long as needed. You can name your functions in each script whatever you choose and the game goes for any variables.

*Cons of Multi State:*
There are no globals shared between states since they are seperate, so you will have to set any you need for each one (player position for example). But then you aren't having to set all the variables for the script each time you run it.

Solution:
You can use Dark Lua as single or multi state or a mix of the two.

*Daves thoughts on multi vs single state*
World of Warcraft uses Lua for all of their GUI and allows user man addons so I was interested to find out which method they picked. They use a single state so everyone can pick up the globals provided, but they have to name their functions *MyUniqueNamedAddon_Function name* and variables *MyUniqueNamedAddon_Variable*.

After having a good play and digging around I think moving forward with FPSCR, for the AI side we should use one scope also. Sure it imposes naming conventions but then the pool of data is available for others to use. This way, much like WoW does we can allow users to create their own addons and scripts knowing they have access to the variables and structures provided, since they won't be able to pass them across like the main DBPro app will be able to.

I think the GUI could be a separate state, particle affects another state etc.

## *Commands*

**Note** that although Lua passes 1 for an error and 0 for success, DBPro (and most other things) do the opposite. To keep DBPro friendly, Dark Lua returns 1 for success and 0 for failure

Because Dark Lua allows multiple states, most commands have an additional ID parameter to specify the state to use.

For the Lua Table commands you can use "." as well as "->". The "." support was added to make it more DBPro like and because I think it looks neater.

**Note** You can now omit the state in all commands. A default Lua state of 1 is used unless overridden by SET LUA STATE iID

***void SET LUA STATE iID***
Sets the default state to be used when a state iD is omitted in any commands (The default state is set to 1 initially and returns to 1 after Lua Reset).

***int LOAD LUA ( "script.lua" , iID )***
***int LOAD LUA ( "script.lua" )***
iID is the state, returns 1 for success, message box if an error. Creates a new state if one does not exist

***void CLOSE LUA ( iID )***
***void CLOSE LUA ( )***
Closes state passed in. message box if an error (NOTE this command was not part of Unity, but since we have multiple states we need the option to close one)

***void LUA RESET***
Silent reset, closes all lua states, all variables will be gone, message queue is emptied and the default state is set back to 1.

***int LUA EXECUTE ( "MyVar = 1" , iID )***
***int LUA EXECUTE ( "MyVar = 1" )***
Execute a string like as a script. iID is the state, 1 for success, message box if an error

***int LUA GET INT ( "MyVar" , iID )***
***int LUA GET INT ( "MyVar" )***
Returns the integer value of the variable

***float LUA GET FLOAT ( "MyVar" , iID )***
***float LUA GET FLOAT ( "MyVar" )***
Returns the float value of the variable

***string LUA GET STRING ( "MyVar" , ilD )***
***string LUA GET STRING ( "MyVar" )***
Returns the string value of the variable

***void LUA SET INT( "MyVar" , iValue , ilD )***
***void LUA SET INT( "MyVar" , iValue )***
Variable name, integer value , lua state id

***void LUA SET FLOAT( "MyVar" , fValue , ilD )***
***void LUA SET FLOAT( "MyVar" , fValue )***
Variable name, float value , lua state id

***void LUA SET STRING( "MyVar" , string , ilD )***
***void LUA SET STRING( "MyVar" , string )***
Variable name, string value , lua state id

***int LUA ARRAY INT ( "soldier.head.armor.rating" , ilD )***
***int LUA ARRAY INT ( "soldier.head.armor.rating" )***
Returns an array (Lua Table) field. can use "->" or "." (Numbered index arrays not supported)

***float LUA ARRAY FLOAT ( "soldier.head.armor.rating" , ilD )***
***float LUA ARRAY FLOAT ( "soldier.head.armor.rating" )***
Returns an array (Lua Table) field. can use "->" or "." (Numbered index arrays not supported)

***string LUA ARRAY STRING ( "soldier.head.armor.name" , ilD )***
***string LUA ARRAY STRING ( "soldier.head.armor.name" )***
Returns an array (Lua Table) field. can use "->" or "." (Numbered index arrays not supported)

***void LUA SET FUNCTION "FunctionName" , ilD, iNumberOfParams, iNumberOfResults***
***void LUA SET FUNCTION "FunctionName" , iNumberOfParams, iNumberOfResults***
Prepare function to be called

***void LUA CALL***
Execute function

***int LUA RETURN INT ( ilD )***
***int LUA RETURN INT ( )***
Pops an int off the stack, if the stack is empty returns 0

***float LUA RETURN FLOAT ( ilD )***
***float LUA RETURN FLOAT ( )***
pops a float off the stack, if the stack is empty returns 0

***string LUA RETURN STRING ( ilD )***
***string LUA RETURN STRING (  )***
Pops a string off the stack, if the stack is empty returns NULL

***void LUA PUSH INT integer, ilD***
***void LUA PUSH INT integer***
Push integer onto the stack

***void LUA PUSH FLOAT float , ilD***
***void LUA PUSH FLOAT float***
Push float onto the stack

***void LUA PUSH STRING string , ilD***
***void LUA PUSH STRING string***
Push string onto the stack

## *The Message Queue*

Messages are separate from any lua states and are global.
The amount of messages sent is only limited by memory.
There are new Lua commands added to make it easy to pass messages to dbpro.
Message queue is FIFO.

***int LUA NEXT()***
Grabs the next message and returns 1 if there is one, 0 if not, removes the message from the queue

***string LUA MESSAGE DESC()***
Returns current message description

***int LUA MESSAGE INDEX()***
Returns current message index

***int LUA MESSAGE INT()***
Returns current message integer

***float LUA MESSAGE FLOAT***
Returns current message float

***string LUA MESSAGE STRING***
Returns current message string

## *New Lua Commands (Used from within a Lua Script)*

***SendMessage( string Descriptor )***
Sends a message to the message queue, descriptor only. You can actually send many of these at the same time, for example: SendMessage ( “Hello” , “Another message”, “Yet another message” );

***SendMessageI( string Descriptor, integer Value )***
***SendMessageI( string Descriptor, Index, integer Value )***
Sends a message to the message queue with a descriptor string and integer value.

***SendMessageF( string Descriptor, float Value )***
***SendMessageF( string Descriptor, Index, float Value )***
Sends a message to the message queue with a descriptor string and float value.

***SendMessageS( string Descriptor, string Value )***
***SendMessageS( string Descriptor, Index, string Value )***
Sends a message to the message queue with a descriptor string and string value.