

# Scripted controllers and object references in 3dsmax R6

By Martin Breidt ([martin@breidt.net](mailto:martin@breidt.net)) – v1.1 - 30.10.2003

*[This document is based on a great discussion with Larry Minton (discreet), for which I would like to say thank you! again]*

This document as well as some example scenes can be found at my MAXscript webpage <http://www.breidt.net/scripts/index.html>

## **Intro**

Scripted controllers (SCs) can be a powerful alternative to expressions, wire controllers or reactor controllers because you can place an entire MAXScript inside a controller.

If you are using references to other objects inside a SC, one common problem arises when you rename these referenced objects. So, if you calculate the SC return value based on another object's property, you have to tell the SC which object that is.

This document is about these object references inside scripted controllers.

## **Pathnames**

So you might be using something like "`$upperArm.pos/2`" in your SC in order to use the position of object "upperArm" and divide it by two. That works fine at first glance, but has a few issues:

The most obvious is the fact that renaming that object to something like "hero\_upperArm" will break your script. Likewise, when you merge one scene into another, you might have duplicate object names, so you'll want to rename one set of objects. Again, the above SC expression will not handle that.

Other problems arise when you use Scene XRef, since that will also cause an error due to not finding the node name used in the script.

A smaller problem with using this kind of object names is the fact that MAXScript has to do a case-insensitive string comparison against all nodes in the scene. In other words: Every time you use something like "`$Teapot01`", the system has to check which object you mean by comparing that string against all object names. So depending on where in the internal scene structure your object is and how many other objects are around in that scene, this might take up some serious CPU cycles.

## **Node Handles**

Another way of addressing objects in the scene is using node handles, introduced in 3dsmax R4: "`$Teapot01.inode.handle`" and "`maxOps.getNodeByHandle <number>`" respectively. This is basically an integer number assigned from a counter that gets incremented each time an object is created. Note that it does not decrease when an object is deleted. Also, after each Reset, it will start over with a value = 1.

The problem here is that this handle number is inconsistent across scenes. So for operations like merge and XRef, this will not work.

## **This**

3dsmax R6 introduced a new local variable 'this' for scripted controllers. While many have waited a long time for it, it is not exactly what you might expect (at least I initially expected it to be something different).

'this' will allow an SC to access itself (the controller) directly, without the need for any fancy object addressing. What it will not do is give you direct access to the geometric object that uses this controller. As I learned, this is something that conflicts with the basic concepts of 3dsmax. But, as we will see in the next two sections, it will help us with this problem as far as the 3dsmax architecture allows.

## **Refs.dependents**

In order to create scripted controllers that are independent of absolute node references, you can also use "refs.dependents obj". This will give you an array of all objects that are dependent on object obj. So in a SC context, you could say "refs.dependents this" and get all nodes that depend on the scripted controller.

Here's a very compact function that Larry cooked up:

```
fn getFirstRefNode obj = (
  local res = undefined
  for ref in refs.dependents obj      -- look at each dependent
    while res == undefined          -- while this condition is true
      where isValidNode ref         -- run expr. if this condition=true
      do res = ref
  res
)
```

And this is what your script controller text could look like:

```
o = getFirstRefNode this
if o != undefined then (
  o.wireColor = if (o.pos.x < 100.0) then color 255 0 0 else color 0 255 0
)
```

This script will change the wirecolor of whatever object it is assigned to to red if its x position is less than 100 units and to green otherwise. You should not use this in a position, rotation or scale script controller, since it will create a cyclic reference: it needs the position to determine the color, but (if used in PRS) also is required by the PRS controller to calculate the position.

Drawbacks of this approach: The `getFirstRefNode` function needs to be declared. For that, you could put it into a global function script that 3dsmax runs upon startup, but then you have to handle two script files in different locations. Also, the SC will be rather slow since it has to do the `refs.dependents` call each time the controller is evaluated. And finally, one has to be aware that one controller can be instanced from multiple objects (of potentially different class) and thus `getFirstRefNode` would need enhancements to cover these cases. Right now it just uses the first object that returns true for the function `isValidNode`.

## **Custom Attributes**

Since 3dsmax R5.1, every MAXWrapper object can carry custom attributes (CAs). CAs can store all kinds of custom data associated with an object and additionally carry a user interface description. That UI is currently accessible directly only in the Command panel (CAs attached to base objects or modifiers) and the Material Editor (CAs attached to materials), but the UI can be manually displayed otherwise as you will see later.

The interesting part of CAs (for this document) is that they come with a parameter block that can carry node pointers and is fully integrated into the 3dsmax structure. So renaming an object will not matter. Also merging and XRef'ing will work. CAs will be saved with the object and travel with it.

As mentioned above, we now (3dsmax R6) have a special local variable 'this' within scripted controllers. That variable points to the controller itself. And as such, provides a mean to access the properties of the controller. You see where this is aiming?

We can now attach a CA to the controller itself. And that CA can carry the reference to an object, just like a persistent variable. And, thanks to Larry Minton's enhancements to R6, we now have relative access to the controller object itself from within the script using the 'this' variable, removing the need for any absolute addressing.

So here's an example script that demonstrates the basic approach, taken from discussions with Larry Minton:

```
-- create the custom attribute definition that does nothing but store a single node
reference
nodeStoreCA = attributes nodeStore attribID:#(0x5df52ec4, 0x76fcba8b)
( parameters main ( node type:#node ) )

-- create some objects for testing
s1 = sphere()
t1 = teapot pos:[-50,50,0]
-- create and assign a position script controller
sc = s1.pos.controller = Position_Script()
-- add the custom attribute to the controller and store the reference to the sphere
inside the CA
custAttributes.add sc nodeStoreCA
sc.nodeStore.node = t1
-- assign the script text to the controller
sc.script = "local target = this.custAttributes[#nodeStore].node; if
isValidNode target then target.transform.pos/2 else [0,0,0]"
-- the script controller will look up the referenced node in it's custom attribute and,
if there is a valid node, use that node's position
```

(Note that there are different ways for accessing a CA: `obj.nodeStore.node` and `obj.custAttributes[#nodeStore].node`)

Try out this example and afterwards select the sphere, then open the Motion tab. Note that the position script controller suddenly has a sub-item (click on the plus sign) called 'nodeStore'. Cute, isn't it? My immediate wish was to click on that entry in order to assign a new node, but unfortunately that is not yet possible. But here's a thing you can do: You can simply add an UI (rollout) definition, just as with other CAs. Since 3dsmax will not show that UI by itself, you also need a function to invoke it. Replace the custom attribute definition in the example above with this:

```
nodeStoreCA = attributes nodeStore attribID:#(0x5df52ec4, 0x76fcba8b) (
  parameters main rollout:params
  ( node type:#node ui:pNode)
  -- here comes the new stuff: the UI definiton
  rollout params "Pick Node" (
    pickbutton pNode ""
    on pNode picked obj do pNode.text = obj .name
    on params open do pNode.text = if isValidNode node then node.name
    else "<<none>>"
  )
  -- we also include a function for displaying that UI, otherwise we won't see it
  fn openDialog = createdialog params
)
```

You now can open the UI using `"s1.pos.controller.nodeStore.openDialog()"` and then click on the button to select a new object for this controller. Aint' that cool?

So, while the controller setup is a bit more complicated than in the other approaches, this has several advantages: It will automatically cause the controller to update as soon as the referenced object changes (no need for 'dependsOn' anymore, since the parameter block creates that dependency itself). According to Larry Minton, this will also be a fair bit faster than using pathnames. And most importantly (I think), it will finally allow you use script controllers and still freely rename objects, since it does not use object names.

## ***Speed***

Here's a quick comparison of the absolute computation time it takes for evaluating the controller for each frame 0...100 a hundred times in a scene containing about hundred objects.

Animated Position XYZ controller	125 ms
Empty script controller	125 ms
Expression controller using another position controller	172 ms
Script controller using pathnames	938 ms
Script controller using CA to store node ref	375 ms

As expected, the script controller is a fair bit slower than an expression controller. But using CAs can save a lot of computation time in your rig.