

MAXScript Tips and Techniques

Larry Minton
Software Engineer
Autodesk Media & Entertainment Division

MAXScript Tips

Make it run faster!

Rule: Do not use return, break, exit, or continue - I

Return, break, exit, continue, and throw are implemented using C++ exceptions.

C++ exceptions are SLOW!

Test cases:

```
fn test1a v = (if v==true do return 1;0)
fn test1b v = (if v==true then 1 else 0)
```

For 100000 iterations:

test1a true	15890 msec
test1a false	78 msec
test1b true	47 msec
test1b false	62 msec

Rule: Do not use return, break, exit, or continue - I

Test cases:

```
fn test2a =  
(  
    local res  
    for i = 1 to 1000 do  
        if i == 10 do (res = i; break;)  
    res  
)  
fn test2b =  
(  
    local notfound = true, res  
    for i = 1 to 1000 while notfound do  
        if i == 10 do (res = i; notfound = false;)  
    res  
)
```

For 100000 iterations:

test2a	84265 msec
test2b	1359 msec

Rule: Only calculate things once

MAXScript does not optimization of code. You must do your own optimizations.

Test cases:

```
fn test4a inVal =  
(  
    local res = 0  
    for i = 1 to 100 do  
        res += ((inVal *10) + i)  
    res  
)  
fn test4b inVal =  
(  
    local res = 0  
    local tmp = inVal *10  
    for i = 1 to 100 do  
        res += (tmp + i)  
    res  
)
```

For 100000 iterations:

test4a 0	20562 msec
test4b 0	17797 msec

Rule: Cache frequently used functions and objects - I

Test case:

```
ep = converttopoly (mesh())           -- node
ep_bo = ep.baseobject                 -- editable poly
polyop_getvert = polyop.getvert       -- structure method
IEditablePoly = ep_bo.EditablePoly    -- FPS interface
IEditablePoly_GetNumMapChannels =     -- FPS function
    IEditablePoly.GetNumMapChannels
```

For 100000 iterations:

```
polyop.getvert ep 1                   470 msec
polyop_getvert ep 1                   79 msec
polyop.getvert ep_bo 1                48 msec
polyop_getvert ep_bo 1                0 msec
ep.EditablePoly.GetNumMapChannels()   76532 msec
ep.GetNumMapChannels()                72141 msec
ep_bo.EditablePoly.GetNumMapChannels() 36688 msec
ep_bo.GetNumMapChannels()             34219 msec
IEditablePoly.GetNumMapChannels()     6454 msec
IEditablePoly_GetNumMapChannels()     0 msec
```

Rule: Cache frequently used functions and objects - II

Test case:

```
gs=geosphere()  
gs_bo = gs.baseobject  
theBend = bend()  
addmodifier gs theBend
```

For 100000 iterations:

gs.radius	718 msec
gs.baseobject.radius	828 msec
gs_bo.radius	688 msec
gs.modifiers[1].angle	719 msec
gs.modifiers[#bend].angle	1187 msec
theBend.angle	609 msec

Rule: Preallocate arrays when size is known.

Used to be very slow to append large number of items to an array due to large number of array memory reallocations.

Now increasing array size by:

```
int new_size = (data_size < 16) ? 16 : (int)(ceil(data_size * 1.5));
```

Test case:

```
fn test3 v =  
(  
    local myArray = #()  
    local m = 500  
    if v do myArray.count = m  
    for i = 1 to m do myArray[i] = i  
)
```

For 100000 iterations:

test3 false	72969 msec, data_size = 620, 10 reallocs
test3 true	71235 msec, data_size = 500, 0 reallocs

Rule: Use `StringStream` to build large strings - I

If building strings, use a `StringStream` value to accumulate the string, and then convert to a string.

Each string addition creates a new string.

```
a = "AAA"
```

```
b = a + a + a + a + a + a
```

Creates 6 strings of length 3, 6, 9, 12, 15, 18

```
a = "AAA"
```

```
b = (a + a + a) + (a + a + a)
```

Creates 6 strings of length 3, 6, 9, 6, 9, 18

For dynamically creating rollout definitions for evaluation, see `stdscripts\baseLib\rolloutCreator.ms`

Rule: Use StringStream to build large strings - II

Test cases:

```
fn test5a =  
(  
    local a = ""  
    for i = 1 to 100 do a += (i as string)  
)  
fn test5b =  
(  
    local ss = stringstream ""  
    for i = 1 to 100 do format "%" (i as string) to:ss  
    ss as string  
)  
fn test5c =  
(  
    local ss = stringstream "", fmt = "%"  
    for i = 1 to 100 do format fmt (i as string) to:ss  
    ss as string  
)  
fn test5d =  
(  
    local ss = stringstream "", fmt = "%"  
    for i = 1 to 100 do format fmt i to:ss  
    ss as string  
)
```

For 100000 iterations:

test5a()	58875 msec, 505 MB
test5b()	54672 msec, 39.2 MB
test5c()	41125 msec, 29.2 MB
test5d()	13532 msec, 10.0 MB

Rule: Use name values instead of strings when possible

Each use of a string literal requires creating a new string value.

Problem case:

```
Fn test() = append "A" "B"
```

```
Test() → "AB"
```

```
Test() → "ABB"
```

Test case:

```
fn test6 v = ()
```

For 100000 iterations:

test6 "A"	125 msec
test6 #a	16 msec

Rule: Be careful of undo system - I

MXS commands run from Listener, Script Editor, or Macro Script are run within an undo on context

MXS commands run from scripted UI controls are not run within an undo on context.

Do not delete nodes with undo off, unless you also created the nodes with undo off, and didn't do anything to the node with undo on.

When working with meshes, you typically want to use meshop methods as they support undo/redo. But typically you don't want to store undo records for all operations in a loop, just the first and last.

When working with epoly, you want to store undo records for all operations in a loop.

Rule: Be careful of undo system - II

Test Case:

```
em = mesh()
meshop_setvert = meshop.setvert
fn test7 holdAll =
(
    local nVerts = getnumverts em
    for i = 1 to nVerts do
        with undo (holdAll or (i == 1 or i == nVerts))
            meshop_setvert em i ([1,1,1]*i)
)
```

For 100000 iterations:

test7 true	85313 msec, 229 MB
test7 false	7609 msec, 11 MB

Rule: Don't use execute function

Very few cases require use of execute function.

MAXScript has many introspection methods and properties:

```
getPropNames <maxobject>  
getProperty <maxobject> <prop>  
setProperty <maxobject> <prop> <val>  
isProperty <maxobject> <prop>  
getInterfaces <maxobject>  
<maxobject>.custattributes  
<rollout>.controls  
getNodeByName <string>
```

Valid cases:

```
dynamic rollout creation  
dynamic scripted CA creation
```

MAXScript Techniques

Make it run better!

Don't use persistent global variables

A good idea gone bad.

Initially added to support scripted controllers and xrefs.

Problems:

- No control over whether they are loaded

- Overwriting previous values

Solutions:

- New scripted controllers

- Use scripted CAs for persistent data storage

Using scripted CAs for persistent data storage - I

Can place a scripted CA on the scene root

```
sceneDataCADef = attributes sceneDataCADef version:1 attribID:#{0x61e9ff5f, 0x63784819}
(
parameters main rollout:params
( note type:#string ui:et_note default:"" )
rollout params "Scene Data Parameters"
( edittext et_note "Note: " )
)
```

```
thescene = (refs.dependents rootnode)[1]
rootNodeDataCA = undefined
if (custattributes.add rootnode sceneDataCADef) do
    rootNodeDataCA = rootnode.custAttributes[rootnode.custAttributes.count]
sceneDataCA = undefined
if (custattributes.add thescene sceneDataCADef) do
    sceneDataCA = thescene.custAttributes[thescene.custAttributes.count]
rootNodeDataCA.note
sceneDataCA.note
rootNodeDataCA.note = "rootnode"
sceneDataCA.note = "thescene"
```

Using scripted CAs for persistent data storage - II

If save and reload file, can access the CAs:

```
rootNodeDataCA = undefined
if (rootnode.custAttributes.count != 0) do
    rootNodeDataCA = rootnode.custAttributes[rootnode.custAttributes.count]
sceneDataCA = undefined
if (thescene.custAttributes.count != 0) do
    sceneDataCA = thescene.custAttributes[thescene.custAttributes.count]
```

If bring the file in as an xref scene, can access the CAs:

```
xr = xrefs.getXRefFile 1
xr_root = xr.tree
xr_rootNodeDataCA = undefined
if (xr_root.custAttributes.count != 0) do
    xr_rootNodeDataCA = xr_root.custAttributes[xr_root.custAttributes.count]
```

Build UI on script/expression controller

By placing a scripted CA on a script or expression controller, can provide a custom UI for the controller.

Define the CA to contain animatable parameters and rollout for UI.

Create a script controller.

Apply the CA to the script controller.

Apply the script controller.

Create variables in the script controller.

Assign Target values to the variables, specifying the CA parameters as the targets.

Display the CA's UI using `createDialog` or `newRolloutFloater` & `addrollout`

Build UI on script/expression controller - II

```
floatDataCADef = attributes floatDataCADef version:1  
(  
  parameters main rollout:params  
  (prop1 type:#float ui:s_prop1 )  
  rollout params "Parameters"  
  ( spinner s_prop1 "Prop1: " )  
  fn getRollout = params  
)
```

```
sc = float_Script()  
custattributes.add sc floatDataCADef  
floatCA = sc.custattributes[1]  
sc.addTarget "prop1" floatCA[#prop1]
```

```
displayControlDialog sc ""
```

Build UI on script/expression controller - III

```
mscas = for ca in sc.custattributes where isMSCustAttrib ca collect ca
rollouts = for ca in mscas where isproperty ca #getrollout collect ca.getrollout()
if rollouts.count == 1 then
    createdialog rollouts[1]
else if rollouts.count > 1 do
    (
        width = rollouts[1].width
        for ro in rollouts do
            width = amax width ro.width
        rof = newrolloutfloater "" (width+13) 0 0 0
        for ro in rollouts do addrollout ro rof
        height = 6
        for ro in rollouts do
            height += ro.height + 24
        rof.size = [rof.size.x,height]
    )
```

End of Presentation
Slide