Maya API Help Page

Naming Conventions

I tend to follow some basic naming conventions when I create my variables and objects.

```
Attributes:     I prefix with "a" -  aMyAttr, aMatrix, aStrength
MDataHandle:    I prefix with "h" -  hMyAttr, hMatrix, hStrength
MObjects:       I prefix with "o" -  oNode, oMat
MMatrix:        I prefix with "mat" -  matWorld, matLocator
                for the inverse I tack on "inv" -  matinvWorld,
matinvLocator
MPlug:          I prefix with "plug" -  plugMat, plugInput, plugStrength
Integers:       I prefix with "n" -  nSided, nCnt, nJnts
Floats:         I prefix with "f" - fStrength, fVal
Doubles:        I prefix with "d" - dStrength, dVal
```

Attribute Creation and Access

How do I create and access an Enum attribute?

```
        // For creation:
        //
        #include <maya/MFnEnumAttribute.h>

        // Then inside of the initialize function for your plugin:
        MFnEnumAttribute eAttr
        aMyEnumAttr = eAttr.create( "myEnum", "enu", 0 );  // can set
default 0, 1, 2, etc..
        eAttr.setStorable(true);
        eAttr.setKeyable(true);
        eAttr.addField("circle", 0) ;  // Here you define the string
label, and it's int value
        eAttr.addField("square", 1) ;
        eAttr.addField("diamond", 2) ;
        eAttr.addField("octo", 3) ;            // etc...


        // To access it from an MDataHandle inside of a compute or
deform call
        // where the MDataBlock is called data:
        //
        MDataHandle hMyEnumAttr = data.inputValue(aMyEnumAttr, &stat );
        int nMyEnum = hMyEnumAttr.asShort();  // Now get it as an SHORT
                // **IMPORTANT you must use "short" and not int here.
                // This is particularly critical if on Windows.
Otherwise
```

```
                // you may read extra bytes, and get a corrupted value
back.


        // To access it from an MPlug in a node:
        //
        MObject thisNode = thisMObject();      // Get this actual plugin
node's MObject...that's us!

        MPlug plugMyEnumAttr( thisNode, aMyEnumAttr );
        int nMyEnum = 0 ;
        plugMyEnumAttr.getValue( nMyEnum );
                // or
        plugMyEnumAttr.setValue( nMyEnum );
```

How do I create and access a Compound attribute?

Compound attrs are simply attributes that have other attributes as their children. It makes connectons easier since you could have 2 nodes that have similar compound attrs, but then only need the one connection from one to the other, rather than connecting each individual attr

WARNING: If you want to use compounds, then when you do the "addAttribute" call you should ONLY do so for the master compound, NOT the children. (I had this incorrct here previously.) Adding the children in, particularly first, will actually cause update issues on the node!

```
        // For creation:
        //
        #include <maya/MFnCompoundAttribute.h>

        // First define each aChildAttr here, ie: aChildAttr1 =
nAttr.create(...) ;
        // etc....  They should exist here.


        // Then inside of the initialize function for your plugin:
        MFnCompoundAttribute cAttr ;
        aCmpdData = cAttr.create( "cmpdData", "data") ;
        cAttr.addChild( aChildAttr1 ) ;
        cAttr.addChild( aChildAttr2 ) ;
        cAttr.addChild( aChildAttr3 ) ;
        cAttr.addChild( aChildAttr4 ) ;
        // etc...

        // Then add only the master compound not the children!!
        addAttribute( aCmpdData );
```

Note that for attributeAffects calls, you can still use both the master compound and the children, to ensure that everthing updates whatever your output is. One other note, you cannot have one child of a compound "affect" another child in the same compound. To do that, you'd need to have at least two separate compounds, one with input attributes and the other with output attributes. Or just one being in a compound and the other outside of a compound for example.

How do I create and access a MMatrix attribute?

```
        // For creation:
        //
        #include <maya/MFnNumericAttribute.h>

        // Then inside of the initialize function for your plugin:
        MFnMatrixAttribute  mAttr ;
        aMatrix = mAttr.create( "myMatrix", "m" );

        // To access it from an MDataHandle inside of a compute or
deform call
        // where the MDataBlock is called data:
        //
        MDataHandle hMatrix = data.inputValue(aMatrix, &stat );
        MMatrix mat = hMatrix.asMatrix();     // Now get it
        MMatrix invmat = mat.inverse();                // and the inverse
too.


        // To access it from an MPlug in a node, you must include
        // MPlug.h and MFnMatrixData.h and then do something as follows:
        //
        MObject thisNode = thisMObject();     // Get this actual plugin
node's MObject...that's us!

        MPlug plugMat( thisNode, aMatrix );   // find our attribute plug
on ourselves
        MObject oMat ;
        plugMat.getValue( oMat );             // For matrix plugs, have
to get as MObject
        MFnMatrixData fnMat( oMat );   // Then take that to a MfnMatrix
Data.
        MMatrix mat = fnMat.matrix() ;  // Then finally get the matrix
out from that.
        MMatrix invmat = mat.inverse() ; // and get inverse too...
```

How do I create and access an attribute that can take a shape, like a Nurbs Surface, Mesh or Subdiv?

```
        // For creation, you create a "Generic" attribute:
        //
        #include <maya/MFnGenericAttribute.h>

        // Then inside of the initialize function for your plugin:
        MFnGenericAttribute gAttr ;
        aSurface = gAttr.create( "inSurface", "surf", &stat );
        gAttr.addAccept(MFnData::kNurbsSurface);      // Accept nurbs
        gAttr.addAccept(MFnData::kMesh);                      // Accepts
Mesh
        gAttr.addAccept(MFnData::kSubdSurface);               // Accept
kSubdSurface
        gAttr.setStorable(false);
        gAttr.setCached(false);

        // To access it from an MDataHandle inside of a compute or
deform call
        // where the MDataBlock is called data:
        //
        MDataHandle hSurface = data.inputValue( aSurface, &stat );
        if( stat != MS::kSuccess ) return stat ;

        // What type of surf connected?
        MObject oSurf ;
        if (hSurface.type() == MFnData::kNurbsSurface)
            oSurf = hSurface.asNurbsSurface() ;
        else if (hSurface.type() == MFnData::kMesh)
            oSurf = hSurface.asMesh() ;
        else if (hSurface.type() == MFnData::kSubdSurface)
            oSurf = hSurface.asSubdSurface() ;
        else
            return stat ;

        // Now we have the basic MObject...now you can create one of
        // the following based on the type it was.  Only one of these
        // will work since only one will be the actual type.  You could
        // store a variable above to note which type it is, and then
create
        // the one needed.
        //
        MFnNurbsSurface fnNurbs(oSurf, &stat) ;
        MFnMesh fnMesh(oSurf, &stat) ;
        MFnSubd fnSubd(oSurf, &stat) ;
```

If I have a surface attribute as described above, how can I get an MItGeometry from it via an MPlug?

If you are in a call that passes in an MDataHandle, it's easy to create an MItGeometry from that data handle. However if you are trying to use an MPlug, you have to do a little more work.

```
        // The way this works is from the MPlug, we trace the connection
to the
        // "other side".  On this other side, we get the actual node
connected.
        // This would be an actual surface shape node.  From that we can
then get
        // an MFnDagNode and from that the dagPath, that the MItGeometry
constructor
        // can use.
        //
        MObject thisNode = thisMObject();      // Get this actual plugin
node's MObject...that's us!

        MPlug plugSurface( thisNode, aSurface ) ;     // Get our
attribute on our node
        MDagPath dpathSurf ;
        if (plugSurface.isConnected())
            {
            MPlugArray parr;
            plugSurface.connectedTo( parr, true, false, &stat ) ;     //
Now list connections on our attr
            MPlug plugSurfSide = parr[0] ;            // Get the plug
that is connected into our aSurface attr.

            MObject oSurf = plugSurfSide.node(&stat) ;        // Get the
actual node that is connected.
            MFnDagNode fnDagSurf( surfNode, &stat ) ; // Put that into a
Dag Node Function Set...
            stat = fnDagSurf.getPath( dpathSurf ) ;           // Now get
the dagPath of this node.
            }
        MItGeometry iter( dpathSurf, &stat) ; // Finally...create the
MItGeometry
        if (stat != MS::kSuccess)             // Make sure we didn't
fail.
            return stat ;
```

How can I easily set and access an array of data, like a array attribute of doubles?

There are two basic ways to do this. Since most plugins usually have a "compute" function or "deform" function with an MDataBlock passed in, you can use MDataHandle to access the array. However you can also access the array via an MPlug. The MPlug version is a lot easier. HOWEVER, Maya can be quite slow when using MPlugs and Logical Indexing. Inside of a compute or deform, you should always try to use the MDataBlock and MDataHandle methods. They will be faster and work better.

The MDataHandle mechanism is gets a little messy, since if you are setting (not reading) an array for output, then you will have to ensure that the actual elements exist first. That is, if you have an array attributes called "myDoubleArray" and you want to set the 12th element in MEL you'd do something like:

```
    setAttr myObj.myDoubleArray[12] 0.0 ; // set to 0
```

However...with an MDataHandle, you must actually ensure that the 12th element here already has been "created" in maya. To do that, the easiest way is to use the Array Data Builder class and then check the size of the array against what you need, and if required, grow the array out. This usually requires a separate function and is sort of a pain. In addition, you have to deal with MArrayDataHandle as well as MDataHandle classes. Check out some of the sample devkit files for examples on how to use the MArrayDataBuilder class.

One of the common things to do is to get weights in a format similar to the skinCluster node. This is an array of compounds, with an array of doubles inside. While the MFnSkinCluster base class has a function you can use to get those weights, sometimes you want to create and access your own weights this way. The following procedure shows a nice proc to do this. The proc takes a data handle, and the two arrays, one for the per point compound, and then one that is the child double array of weights inside of each of the previous elements. Note that if you wan't to set the data here, you'd instead want to use MArrayDataBuilder first to make sure all the needed elements exist.

```
        /*
         * getWeights() - Uses data block to nicely get weights
         *      for given attrs passed in.  This gets all weights for
the given listIdx.
         *
         *      Parameters:
         *                      data            - MDataBlock passed in
         *                      aWtList         - Attribute that is the
per point larger compound weight list
         *                      aWts            - Attribute that is the
double array of weights inside of the WtList array.
         *                      dArrWts         - Array of doubles to
values we are getting.
         *                      nTotal          - Total number of elements
we expect in the weights array.
         *                      listIdx         - What pt/list index for
the first list array do we want
         */
        MStatus getWeights(MDataBlock &data, MObject &aWtList, MObject
&aWts, MDoubleArray &dArrWts, int nTotal, int listIdx)
        {
                MStatus stat ;
                dArrWts.setLength(nTotal) ;    // first make sure array
can hold it all

                // Get handle to array wt list
                MArrayDataHandle hArrWtList =
data.inputArrayValue(aWtList, &stat) ;

                // Go to right pt/list index...
                stat = hArrWtList.jumpToElement( listIdx ) ;
                if (stat != MS::kSuccess)      // If that index doesn't
exist, our wts are 0..so just return.
                        return stat ;

                // Now we get the listIdx-th compound element handle...
                MDataHandle hCmpdWtListEle =
hArrWtList.inputValue(&stat) ;
                if (stat != MS::kSuccess)
                        {
                        cout << "Odd, can't get weight list compound
element handle!" << endl ;
                        return stat ;
                        }
                // Now from this compound, get the aWts weights array
child...
               MDataHandle hWtListChild = hCmpdWtListEle.child( aWts ) ;

                // Now get this child array as an MArrayDataHandle....
                MArrayDataHandle hArrWts (hWtListChild, &stat) ;
                if (stat != MS::kSuccess)
                        {
                        cout << "Odd, can't get weight list handle!" <<
endl ;
                        return stat ;
                        }

                // Whew!  Now we can finally just go through the weights
array and get the values
                // Since indexing is logical/sparse, not all elements
may exist...
                int idx=0;
                for (idx=0; idx < nTotal; ++idx)
                        {
                        stat = hArrWts.jumpToElement(idx) ;
                        if (stat != MS::kSuccess)              // If
element doesnt exist..wt is 0.0
                                dArrWts[idx] = 0.0 ;
                        else                    // get value from child
double
                                {
                                MDataHandle hWtEle =
hArrWts.inputValue(&stat) ;
                                if (stat != MS::kSuccess)              //
Strange, cant get handle.
                                {
                                        cout << "Odd, can't get weights
element handle!" << endl ;
                                        dArrWts[idx] = 0.0 ;
                                        }
                                else
                                       dArrWts[idx] = hWtEle.asDouble() ;
                                }
                        }

                return MS::kSuccess ;
        }
```

By comparison, the MPlug class is much easier and more straightforward to work with and is a lot closer to it's MEL counterpart. It also allows the automatic creation of the array indices if needed, when you use the "elementByLogicalIndex()" call. If you call that with an index that doesn't yet exist, Maya will make it for you, and you can safely and easily set and access that plug.

One thing to remember is tha a Plug is basically a "obj.attr" pair. The way arrays work is that the main "array" obj.attr is the kind of top level parent plug. Within that then there are the actual elements. In the API, these are element plugs, similar to a child plug. If you have an array with 10 elements in it, you will have the 1 main plug "obj.arrAttr" and then the 10 element plugs, "obj.arrAttr[0]", "obj.arrAttr[1]", "obj.arrAttr[2]", etc... In fact this is exactly how you work with it in the API. You get the first master plug, and then using elementByLogicalIndex() or elementByPhysicalIndex() obtain the proper element plug from which you can actually set or get data.

One nice thing to note too, is that you can have array plugs or compounds or any type as a child attribute. This means you can have an array attribute, whose elements are yet another array attribute, or even a compound. This is in fact how the skinCluster node stores it weights. There is a weightList[#].weights[#] array.

One other note is that if you are using MPlugs in a compute and you are going to set an array output attribute, you should still call the data.setClean() command on the MDataBlock to mark that output array clean, otherwise you will continue to get called for each element.

Finally note that I set usesArrayDataBuilder on, even though it is for use with the MPlug. I have seen cases where Maya will crash in it's own code, even after the plugin is done, because this was not used. It seems best to leave it on regardless.

```
        // For creation:
        //
        #include <maya/MFnNumericAttribute.h>

        // Then inside of the initialize function for your plugin:
        MFnNumericAttribute nAttr ;

        aMyArray = nAttr.create( "myDoubleArray", "arr",
MFnNumericData::kDouble, 0.0 );
        nAttr.setStorable(true);
        nAttr.setConnectable(true) ;
        nAttr.setArray(true);          // Set this to be an array!
        nAttr.setUsesArrayDataBuilder(true);          // Set this to be
true also!


       // Now later on, use MPlug's to access this array on your node...
        //
        MObject thisNode = thisMObject();     // Get this actual plugin
node's MObject...that's us!

        MPlug plugMyArray( thisNode, aMyArray) ;      // Get the MPlug
to the array...

        // Then you could do a loop with something like this:
        double dVal ;

        // Now if you know you have elements from 0...nEle and they all
exist
        // you can do the following.  The nice thing is if they don't
exist,
        // calling elementByLogicalIndex will automatically create them
for you.
        //
        for (idx=0; idx < nEle; ++idx)
            {
            MPlug plugElement = plugMyArray.elementByLogicalIndex( idx,
&stat) ;
            plugElement.getValue( dVal ) ;
                // or
            plugElement.setValue( dVal ) ;
            }


        // On the other hand, if you are not sure of the indices you can
use
        // elementByPhysicalIndex() instead so that you properly read
only the ones desired
        // Just remember...you cannot use "elementByPhysicalIndex" to
automatically
        // grow or expand the array like the "Logical" call can.
        //
        unsigned nEle = plugMyArray.numElements( &stat ) ;    // how
many elements?
        for (idx=0; idx < nEle; ++idx)
            {
            MPlug plugElement = plugMyArray.elementByPhysicalIndex( idx,
&stat) ;
            plugElement.getValue( dVal ) ;
                // or
            plugElement.setValue( dVal ) ;
            }


        // ...or you can actually get the indices that do exist into an
int array
        // and then just access those indices.
        MIntArray iarrIndexes ;        // array to hold each valid index
number.
        unsigned nEle = plugMyArray.getExistingArrayAttributeIndices
( iarrIndexes, &stat ) ;
        // Now the array has something like 0,1,4,6,7,8,12  assuming
those are valid indices
        //
        int i, idx ;
        for (i=0; i < iarrIndexes.count(); ++i)
            {
            idx = iarrIndexes[i] ;     // Get the i-th index.

            //  then once again...
            MPlug plugElement = plugMyArray.elementByLogicalIndex( idx,
&stat) ;
            plugElement.getValue( dVal ) ;
                // or
            plugElement.setValue( dVal ) ;
            }
```

OpenGL Drawing for MPxLocator Nodes

How can I determine what color to use?

You can use the "view.setDrawColor()" command to set a simple RGB color where each r,g,b value ranges from 0-1. Below shows a snippet of code to determine an override color, that matches the basic Maya default colors, or else draws in some other color.

```
        bool bColOverride = false ;    // override color?
        MColor col ;

        // And now a few overrides for selection, last selection or
        //      template... So make nice colors
        //
        switch (status)  // From M3dView::DisplayStatus status  in the
draw() command
            {
            case M3dView::kLead
                col= MColor(0.26, 1.0, 0.64)  ;               // maya
green
                bColOverride = true;
                break ;

            case M3dView::kActive
                col = MColor(1.0, 1.0, 1.0)  ;        // maya white
                bColOverride = true;
                break ;

            case M3dView::kActiveAffected
                col = MColor(0.78, 1.0, 0.78)  ;      // maya magenta
                bColOverride = true;
                break ;

            case M3dView::kTemplate
                col = MColor(0.47, 0.47, 0.47)  ;     // maya template
gray
                bColOverride = true;
                break ;

            case M3dView::kActiveTemplate
                col = MColor(1.0, 0.47, 0.47)  ;      // maya selected
template pink
                bColOverride = true;
                break ;

            default:
                col = MColor(0.1, 0.2, 0.7) ; // else set color as
desired
            }
        // ... after open gl has started and current_bit has been
pushed...
        //
        view.setDrawColor( col ) ;             // now switch to that
color
```

How can I tell if I should draw in shaded mode or not?

The following code shows how to use the M3dView::DisplayStyle style that gets passed into a draw() command.

```
        if ( style == M3dView::kFlatShaded ||
             style == M3dView::kGouraudShaded )
            {
            // Draw shaded things here like triangles, quads or polys
            }

        // Here you could do an else to only draw edges when needed, or
you could
        // simply always draw edges..
```

How can I draw points?

```
        view.beginGL();        // Start openGL

                        // Push the color settings
        glPushAttrib( GL_CURRENT_BIT | GL_POINT_BIT );
        glPointSize( 4 ) ;     // set point size..can be 1-64
        glDisable ( GL_POINT_SMOOTH );  // Draw square "points"
        glBegin( GL_POINTS );
        view.setDrawColor( MColor(0.1, 0.2, 0.7) );   // Set color as
desired


        // Now draw the points
        glVertex3f( 0.0, 0.0, 0.0) ;  // these are x,y,z positions of
point to draw
        // etc...


        glEnd();        // end of POINTS
        glPopAttrib();


        view.endGL();          // End openGL
```

How can I draw edges?

```
        view.beginGL();                // Start openGL

                       // Push the color settings
        glPushAttrib( GL_CURRENT_BIT );
        glBegin( GL_LINES );
        view.setDrawColor( MColor(0.1, 0.2, 0.7) );   // Set color as
desired

        // Now draw the edges, defined by a start and end point
        glVertex3f( 0.0, 0.0, 0.0) ;  // these are x,y,z positions of
point 1
        glVertex3f( 1.0, 0.0, 0.0) ;  // these are x,y,z positions of
point 2
        // etc...


        glEnd();       // end of LINES
        glPopAttrib();


        view.endGL();                  // End openGL
```

How can I draw faces?

```
        view.beginGL();                // Start openGL

                       // Push the color settings
        glPushAttrib( GL_CURRENT_BIT );
        glBegin( GL_TRIANGLES );
        view.setDrawColor( MColor(0.1, 0.2, 0.7) );   // Set color as
desired

        // First set normal for the tri we are about to draw, otherwise
lighting won't work!
        glNormal3f( 0.0, 1.0, 0.0 );
        // Now draw the triangle, defined by 3 points
        glVertex3f( 0.0, 0.0, 0.0) ;  // these are x,y,z positions of
point 1
        glVertex3f( 1.0, 0.0, 0.0) ;  // these are x,y,z positions of
point 2
        glVertex3f( 0.0, 0.0, 1.0) ;  // these are x,y,z positions of
point 3
        // etc...


        glEnd();       // end of LINES
        glPopAttrib();


        view.endGL();                  // End openGL
```

Note that for tris, polys, etc you MUST call glNormal to set the normal of the triangle before drawing it if you want it to work properly with lighting! Also note that you could actually insert the color call between each vertex to get shading on each individual vertex of the triangle.

What are some other useful glBegin modes?

The following is taken from the openGL help at:
http://www.mevis.de/~uwe/opengl/opengl.html

GL_POINTS

Treats each vertex as a single point. Vertex *n* defines point *n*. *N* points are drawn.

GL_LINES

Treats each pair of vertices as an independent line segment. Vertices *2n-1* and *2n* define line *n*. *N/2* lines are drawn.

GL_LINE_STRIP

Draws a connected group of line segments from the first vertex to the last. Vertices *n* and *n+1* define line *n*. *N-1* lines are drawn.

GL_LINE_LOOP

Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices *n* and *n+1* define line *n*. The last line however, is defined by vertices *N* and *1*. *N* lines are drawn.

GL_TRIANGLES

Treats each triplet of vertices as an independent triangle. Vertices *3n-2*, *3n-1*, and *3n* define triangle *n*. *N/3* triangles are drawn.

GL_TRIANGLE_STRIP

Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. For odd *n*, vertices *n*, *n+1*, and *n+2* define triangle *n*. For even *n*, vertices *n+1*, *n*, and *n+2* define triangle *n*. *N-2* triangles are drawn.

GL_TRIANGLE_FAN

Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. Vertices *1*, *n+1*, and *n+2* define triangle *n*. *N-2* triangles are drawn.

GL_QUADS

Treats each group of four vertices as an independent quadriliteral. Vertices *4n-3*, *4n-2*, *4n-1*, and *4n* define quadriliteral *n*. *N/4* quadriliterals are drawn.

GL_QUAD_STRIP

Draws a connected group of quadriliterals. One quadriliteral is defined for each pair of vertices presented after the first pair. Vertices *2n-1*, *2n*, *2n+2*, and *2n+1* define quadriliteral *n*. *N/2-1* quadriliterals are drawn. Note that the order in which vertices are used to construct a quadriliteral from strip data is different from that used with independent data.

GL_POLYGON

Draws a single, convex polygon. Vertices *1* through *N* define this polygon.

Where can I get more help on openGL?

The official openGL site with lots of good reference, tutorials, sample files and so on is at http://www.opengl.org/.

Compile, Link and Run-Time Errors

How come I suddenly get an error about "unresolved external symbol" when I add an attribute to my class?

The usual answer is that you declare your static attribute variables in your class, but because they are static they must actually be allocated somewhere. Therefore they must also be listed once outside of the class. This is why attributes have to be coded twice, and usually look similar to something like below. If you don't do this, you'll refer to your attribute variable, but they won't actually exist or be allocated anywhere, and so the linker won't be able to find it.

Another reason could simply be that you aren't linking in one of the required libraries. The error basically means that there is some function or variable that you are accessing, but that now it can't find when it actually links everything together.

```
/*
 * myDeformer Class
 */
class myDeformer : public MPxDeformerNode
{
public:
        myDeformer();
        virtual ~myDeformer();

        static void*    creator();
        static MStatus  initialize();

        // deformation function
        //
        virtual MStatus deform(MDataBlock& block,
                MItGeometry&    iter,
                const MMatrix&  mat,
                unsigned int    multiIndex);


        // Now these are the nodes attributes
        //
        static MObject          aMyAttr1 ;
        static MObject          aMyAttr2 ;
        static MObject          aMyAttr3 ;
} ;  // don't forget this semicolon!

// YOU MUST Redeclare static attrs here, so that they are actually
// created/allocated by the compiler.
//
MObject myDeformer::aMyAttr1 ;
MObject myDeformer::aMyAttr2 ;
MObject myDeformer::aMyAttr3 ;
```

Why does using my plugin crash Maya?

There are many reasons something like this might occur, however one of the common reasons is that you are using using an array class, but are going outside of the allocated range of the array.

For example if you have an MIntArray, you must remember to set a size for it either when it is created, or with the "setLength()" function.

Another reason is that even if you set the length, you might end up going outside of the array. One common mistake is to do something with an MItGeometry and use the iter.index() as the array index. This will usually work, but it's possible for the iter.index() to actually return a skipped value, this can happen when iterating over a nurbsSurface, where the [u][v] indices get converted into a single [w] index and could actually skip a value.

In that scenario you may do an iter.count() and see that there are 10 elements. But then iter.index() might return something like 0 1 2 3 4 7 8 9 10 11. If you set your array to the proper length of 10, then trying to access with the last two 10,11 indices may crash maya. To fix that, you either have to use your own counter, or grow the array if the element is greater.

For example this is BAD:

```
// DON'T DO THIS, YOU MAY CRASH:
MDoubleArray darrXPos ;
int nVerts = iter.count() ;
darrXPos.setLength(nVerts) ;   // this i ok, we'll have 0...nVerts
elements, one for each vert

// Now loop thru the iter verts...
for ( ; !iter.isDone(); iter.next() )
    {
    int idx = iter.index() ;   // UH-OH!  THIS WILL GET US IN TROUBLE
LATER!
    MPoint pt = iter.position() ;

    darrXPos[idx] = pt.x ;     // BAD BAD THIS MAY OR MAY NOT CAUSE A
CRASH
```

```
    }
```

The above code is bad, since the idx used is from iter.index() which may skip values and this may go above nVerts. Instead keep your own counter such as:

```
// THIS ONE IS SAFE:
MDoubleArray darrXPos ;
int nVerts = iter.count() ;
darrXPos.setLength(nVerts) ;   // this i ok, we'll have 0...nVerts
elements, one for each vert
int idx = 0 ;   // init our own idx counter.

// Now loop thru the iter verts...
for ( idx=0; !iter.isDone(); iter.next(), ++idx )
    {
    MPoint pt = iter.position() ;
    darrXPos[idx] = pt.x ;    // This is ok, we just increment idx in
the loop ourselves each time
    }
```

Another thing I have seen with arrays, even if using the MPlug method is that you may need to set "setUsesArrayDataBuilder" to true. Otherwise Maya might crash if it builds the attribute for you. At least in older versions of Maya I have seen this.

Making Arrays of Classes

I'd like to make an array class for my own classes or for those Maya doesn't provide like MMatrix. How can I generate my own MMatrixArray style of class?

The easiest way is to make a template class that will let you implement a dyanamic array with the same function prototypes that the Maya classes use. In fact I have already done this and have made it freely available. Just download my CArray.h template class file. Read the top of the header for more information on it.