MAYA operations

Anzong Zheng

January 1, 2015

Contents

Chapter 1

Mel

1.1 Basic types

1.1.1 int

1.1.2 float

1.1.3 string

1.1.4 vector

```
// default:
<<0.0,0.0,0.0>>

// usage:
.x .y .z
```

1.1.5 array

dynamic array, use size to get length, the first index is 0.
use clear to clear.

1.1.6 matrix

```
matrix $mat[2][4] = <<1,2,3,4;5,6,7,8>>;
```

1.2 usage of mel

1.2.1 add to path or put in default folder

This method doesn't work even if I add corresponding paths to `MAYA_SCRIPT_PATH`

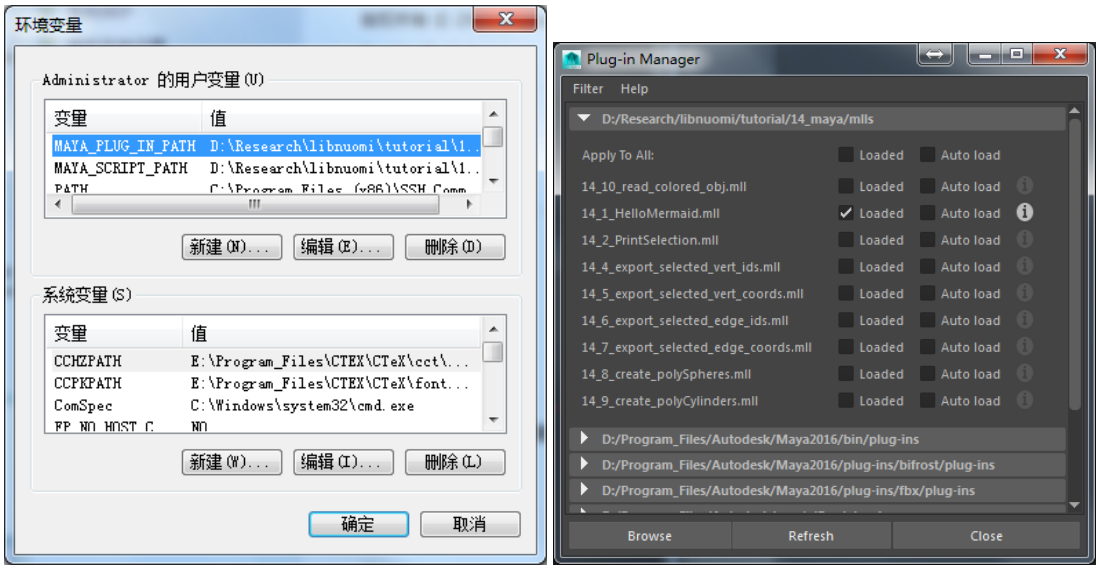
```
# locate Maya.env
about -environmentFile;

# add user script path in Maya.env
MAYA_SCRIPT_PATH = D:/Git/libnuomi/tutorial/14_maya/scripts

# show MAYA_SCRIPT_PATH
getenv MAYA_SCRIPT_PATH
getenv MAYA_PLUG_IN_PATH
```

1.2.2 system environment

Finally, I put `MAYA_SCRIPT_PATH` and `MAYA_PLUG_IN_PATH` into system variables, then **Source** and **Mel** commands can work.



(a) system environments (b) plug ins

In Mac

```
sudo vi ~/.bash_profile
setenv MAYA_SCRIPT_PATH=/path/to/scripts
setenv MAYA_PLUG_IN_PATH=/path/to/plugins
```

1.2.3 Source

Can be used to update modified mel files.

```
source "e:/kfq/mel/mel_test.mel";
test_proc;
```

1.3 rename

1.3.1 Synopsis

```
    rename [-ignoreShape] [object] string
```

rename is undoable, **NOT queryable**, and **NOT editable**.

Renames the given object to have the new name. If only one argument is supplied the command will rename the (first) selected object. If the new name conflicts with an existing name, the object will be given a unique name based on the supplied name. It is not legal to rename an object to the empty string.

When a transform is renamed then any shape nodes beneath the transform that have the same prefix as the old transform name are renamed. For example, ”rename nurbsSphere1 ball” would rename "nurbsSphere1|↩ nurbsSphereShape1" to "ball|ballShape".

If the new name ends in a single ’#’ then the rename command will replace the trailing ’#’ with a number that ensures the new name is unique.

Notes

If the name has an absolute namespace part, it will be considered. Namespaces that do not exist will be created automatically as needed. If the name has a relative namespace part, it will be ignored. In that case, the object will be put under the current namespace. (see example below).

Return value

string The new name. When undone returns original name.

Flags

-ignoreShape(-is) Indicates that renaming of shape nodes below transform nodes should be prevented.

Mel Examples

```
// create two namespaces under the root namespace and create
// a sphere under the root namespace and a sphere under one
// of the new namespaces.
namespace -set ":";
sphere -n sphere1;
namespace -add nsA;
namespace -add nsB;
namespace -set nsA;
sphere -n sphere2;
namespace -set ":";

// change name of sphere1
rename sphere1 spinning_ball;
// result: spinning_ball //

// change name of spinning_ball back to sphere1
select -r spinning_ball;
rename sphere1;
// Result: sphere1 //

// move sphere2 to namespace nsB
rename nsA:sphere2 nsB:sphere2;
// Result: nsB:sphere2 //

// move sphere2 back to namespace nsA when not in the root namespace
// Note the ":" appearing in front of the new name to indicate
// we want to move the object to namespace nsA under the root namespace.
namespace -set nsB;
rename nsB:sphere2 :nsA:sphere2;
// Result: nsA:sphere2 //

// Let's try this without the leading ":" in the new name.
// Since we are namespace nsA, in affect, what we are trying to do
// is rename :nsB:sphere2 to :nsA:nsB:sphere3. Since there isn't a
// nsB namespace under the namespace nsA, the namespace specification
// on new name is ignored and a warning is issued.
namespace -set ":nsA";
rename nsA:sphere2 nsB:sphere3;
// Warning: Removing invalid characters from name. //
// Result: nsA:sphere3 //

// rename an object when not in the root namespace
// and move the object to current namespace
namespace -set ":nsB";
rename nsA:sphere3 sphere4;
// Result: nsB:sphere4 //

// rename an object with an absolute name to move it into a new namespace.
// The namespace does not exist so will be created.
namespace -set ":nsB";
rename nsA:sphere3 :nsC:sphere4;
// Result: nsC:sphere4 //
```

1.4 polyInfo

Synopsis polyInfo is NOT undoable, NOT queryable, and NOT editable.

Command query's topological information on polygonal objects and components. So, the command will require the following to be specified: - selection list to query

Return value
string Components
Keywords
query, polygons, information, topology
Flags

-nonManifoldVertices(-nmv) Find all non-manifold vertices in the specified objects.

-nonManifoldEdges(-nme) Find all non-manifold edges in the specified objects.

-laminaFaces(-lf) Find all lamina faces in the specified objects.

-edgeToFace(-ef) Returns the faces that share the specified edge. Requires edges to be selected.

-vertexToFace(-vf) Returns the faces that share the specified vertex. Requires vertices to be selected.

-faceToEdge(-fe) Returns the edges defining a face. Requires faces to be selected.

-faceToVertex(-fv) Returns the vertices defining a face. Requires faces to be selected.

-edgeToVertex(-ev) Returns the vertices defining an edge. Requires edges to be selected.

-vertexToEdge(-ve) Returns the Edges connected to a vertex. Requires vertices to be selected.

-faceNormals(-fn) Returns face normals of the specified object. If faces are selected the command returns the face normals of selected faces. Else it returns the face normals of all the faces of the object.

MEL examples

```
// To find all non-manifold edges on a polygonal object called pPlane1
polyInfo -nme;
// Result: pPlane1.e[74] //
// To find all non-manifold vertices on a polygonal object called pPlane1
polyInfo -nmv;
// Result: pPlane1.vtx[38] pPlane1.vtx[49] //
```

1.5 Using system()

Maya is rarely used on its own, there are normally a whole host of command line utilities associated with the development of 3D assets. At some point or other, we may find that we need one of those external programs to be called from inside Maya. To do this we use the system call.

1.5.1 using DIR/ls

This simple example uses system to read the contents of the current working directory. Please don't use this for reading directories though, use the portable mel command getFileList instead.

```
{
    // for windows
    string $files[] = system( "DIR" );

    // for linux
    string $files[] = system( "ls" );

    // loop through the returned files and print
    for( $file in $files )
    {
        // print file
        print( $file + "\n" );
    }
}
```

1.5.2 spawning GUI based App's

A big problem with system is that mel will wait for it to finish execution of the command before control will be returned to the script. This is fine for simple programs such as ls or pwd where we actually want the returned results. However an application with a GUI normally doesn't terminate for a long time after the initial system call was made.

In effect, we need to spawn GUI based App's in the background so that our script can continue running. Under linux we need to redirect the output of the application (normally to dev/null). Under Windows you need to use either shell or start before the executable name.

```
{
    // spawning fcheck as a background process under WIN32
    system("start C:/aw/Maya/bin/fcheck.exe " + $filename);

    // spawning fcheck as a background process under linux
    system("fcheck " + $filename + " >/dev/null 2>&1 &");
}
```

Chapter 2

User Interface

2.1 Windows

mel is primarily used within Maya to control every aspect of the User Interface. There are a great number of user interface elements available to you, however i shall only deal with 3 general groups; **windows**, **layouts** and **controls**.

The following example demonstrates the purpose of each GUI element type.

2.1.1 A simple GUI

The mel command *window* allow us to create (suprisingly) a window. Within the window we will want to place controls, however first we must tell Maya how we want the controls to be laid out. In this case I am using *columnLayout* to place all controls in a column. There are a number of other Layouts available which can be used to arrange your GUI's far better than this simple example ;)

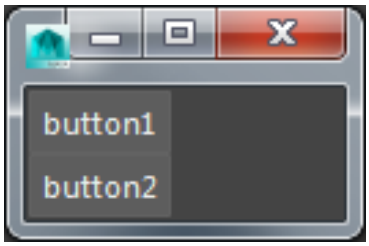


Figure 2.1: A simple window

```
{
    // create a window
    window -title "Mermaid" -widthHeight 300 200 winMermaid;

        // define the layout of controls added to the window.
        columnLayout;

        // create a couple of buttons
        button -label "button1";
        button -label "button2";

    // show the window we last created
    showWindow winMermaid;
}
```

To make sure only one instance of the window is opened: Use the "deleteUI" command to remove a window (or even a control).

```
if (`window -exists myWindow`) deleteUI myWindow;
window -title "new window" -widthHeight 300 200 myWindow;
showWindow myWindow;
```

Maya will always remember the size and position of a window if the user changes it. If you want to ensure the window opens at the original size, use the ”windowPref” command:

```
window -title "new window" -widthHeight 300 200 myWindow;
    if (‘windowPref -exists myWindow‘) windowPref -remove myWindow;
    columnLayout;
    button;
    button;
showWindow myWindow;
```

2.1.2 Changing GUI Colours

we can make use of the -bgc (background colour) flag to change the colours of the user interface elements (only works under Win32 i believe). This can be really useful to colour co-ordinate your GUI elements for different parts of the pipeline. (green for exporter, blue for sound plugins, yellow for modelling etc)

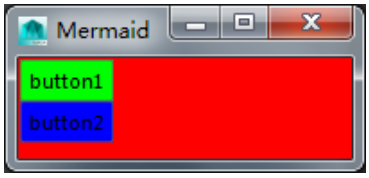


Figure 2.2: change background color of window

```
{
// create a window
window -title "Mermaid";

// define the layout of controls added to the window.
columnLayout -bgc 1 0 0;

// create a couple of buttons
button -bgc 0 1 0 -label "button1";
button -bgc 0 0 1 -label "button2";

// show the window we last created
showWindow;
}
```

2.1.3 Commands and deleting the GUI

To delete any user interface element is simply a case of calling

```
deleteUI "uiElementName" ;
```

If we delete a window, all of it’s child controls will be deleted also.

In addition, we can also assign commands to most control types. In this example we will use the -command flag to specify some mel code to be executed whenever the button is clicked. The command that we will execute will simply delete the window that was created.

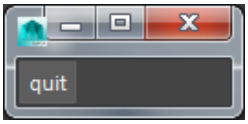


Figure 2.3: delete window

```
{
// create a window and store the name
```

```
$win = `window`;

// define the layout of controls added
// to the window.
columnLayout;

// create a command to delete the window
$command = ("deleteUI " + $win);

// create a couple of buttons
button -label "quit" -command $command;

// show the window we last created
showWindow;
}
```

2.2 Layouts

When placing User Interface controls into your window's, need need to specify a layout for those items. Maya contains a number of different layout controls for different purposes. This page will examine a few of the basic types.

2.2.1 columnLayout

he simplest of all layouts is the columnLayout. This simply places all controls in a vertical column.

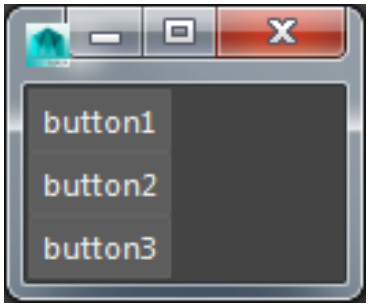


Figure 2.4: columnLayout

```
{
    // create a window
    window -title "new window" -widthHeight 300 200 myWindow;;

        // define the layout of controls added
        // to the window.
        columnLayout;

         // columnLayout -adjustableColumn true;
        columnLayout -columnAttach "both" 5 -rowSpacing 5 -columnWidth 100;
        button -label "button1";
        button -label "button2";
        button -label "button3";

    // show the window we last created
    showWindow myWindow;
}
```

2.2.2 rowLayout

The rowLayout simply places all controls in a horizontal row. We need to specify the number of columns to be used in the row (ie, how many controls we wish to place underneath it). In this example the width of the 3 colums is also specified

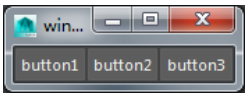


Figure 2.5: rowLayout

```
{
    // create a window
    window -title "new window" -widthHeight 300 200 myWindow;
        rowLayout
            -numberOfColumns 3
            -columnWidth3 120 120 120
            -columnAlign3 "center" "center" "center";

            button -label "make sphere" -command "sphere";
            button -label "button2";
            button -label "button3";

    // show the window we last created
    showWindow myWindow;
}
```

2.2.3 grid layout

```
{
    window -title "new window" -widthHeight 300 200 myWindow;
        gridLayout -numberOfColumns 2 -cellWidthHeight 80 20;
            button -label "make sphere" -command "sphere";
            button -label "make cone" -command "cone";
            button -label "make cube" -command "polyCube";
            button -label "make circle" -command "circle";
    showWindow myWindow;
}
```

2.2.4 frameLayout

A frameLayout creates a collapsable layout. The frame can only hold a single user interface item. Therefore if you want more than one control inside the frame, you will need to nest another layout inside the frame layout.

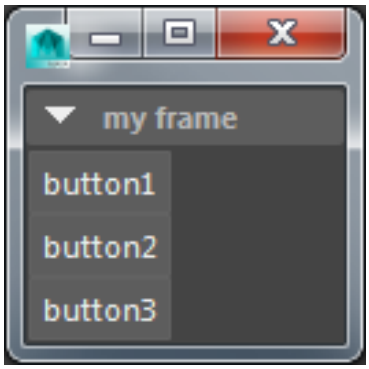


Figure 2.6

```
{
    // create a window
    window;
        // create a collapsible frame layout
        frameLayout -collapsable true -label "my frame" -borderStyle "etchedIn";

            // define the layout of controls added
            // to the window.
            columnLayout;

            // create a couple of buttons
            button -label "button1";
            button -label "button2";
            button -label "button3";

    // show the window we last created
    showWindow;
}
```

2.2.5 setParent

As soon as we create a layout, all new GUI elements will automatically be added underneath the new layout. Often though we want to change the default layout underwhich controls get parented.

The following example places 2 frame layouts underneath a columnLayout. Within each frame layout is a columnLayout which contains some controls. So, having finished adding controls to frame 1, we need to set the highest columnLayout as the current parent so that frame2 will be added underneath that rather than frame1.

to do this, we could either use `setParent $ui_name` or we can use setParent .. which takes us up a single level in the GUI.

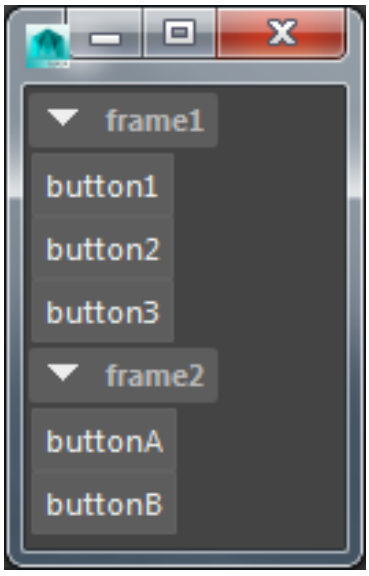


Figure 2.7: setParent

```
{
    // create a window
    window;

    // two frame layouts will be laid out in a column
    columnLayout;

        // create a collapsible frame layout
        frameLayout -collapsable true -label "frame1";
```

```
10
11              // define the layout of controls added
12              // to the window.
13              columnLayout;
14
15              // create a couple of buttons
16              button -label "button1";
17              button -label "button2";
18              button -label "button3";
19
20              setParent ..;
21          setParent ..;
22
23          // create a collapsible frame layout
24          frameLayout -collapsable true -label "frame2";
25
26              // define the layout of controls added
27              // to the window.
28              columnLayout;
29
30              // create a couple of buttons
31              button -label "buttonA";
32              button -label "buttonB";
33
34              setParent ..;
35          setParent ..;
36
37      // show the window we last created
38      showWindow;
39  }
```

2.3 Controls

2.3.1 Simple Text

The mel command text allows us to specify some simple text within the user interface.

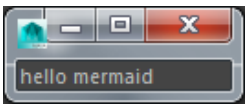


Figure 2.8: Simple Text

```
1   {
2       // create a window
3       window;
4
5           // define the layout of controls added
6           // to the window.
7           columnLayout;
8
9           // create some text
10          text -label "hello mermaid";
11
12      // show the window we last created
13      showWindow;
14  }
```

2.3.2 Simple Button

Buttons are the most basic of all user interface controls. This example simply attachs a command to the button.

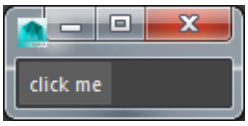


Figure 2.9: Simple Button

```
// a function to be called when the button gets clicked.
proc func()
{
    print("button clicked\n");
}

// create a window
window;

    // define the layout of controls added
    // to the window.
    columnLayout;

    // create a button
    button -label "click me" -command "func";

// show the window we last created
showWindow;
```

2.3.3 Symbol Button

Symbol buttons work in the same way as normal buttons except that they display an image instead of a text label. Use the -image flag to specify the image to use; you may use either bmp or xpm images (gimp can create xpm images).

```
    // create window
    window;

        columnLayout;

        // create three symbol buttons with related mel command
        symbolButton -image "circle.xpm" -command "circle";
        symbolButton -image "sphere.xpm" -command "sphere";
        symbolButton -image "cube.xpm" -command "polyCube";

    showWindow;
```

2.3.4 Simple Checkbox

Checkboxes simply store an on/off value. You can either set up commands to run when the state changes, or you can manually query the value from the checkbox.

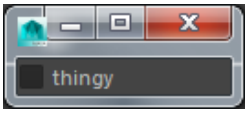


Figure 2.10: checkbox

```
// a function to be called when the checkbox gets checked.
proc on_func()
{
    print("checkbox on!\n");
}

// a function to be called when the checkbox gets unchecked.
proc off_func()
{
    print("checkbox on!\n");
}

{
    // create a window
    window;

        // define the layout of controls added
        // to the window.
        columnLayout;

        // create a checkbox
        $c = `checkBox -label "thingy"
        -onCommand "on_func"
        -offCommand "off_func"`;

        // show the window we last created
    showWindow;

    // to get the current value of the checkBox, use the -query flag
    $value = `checkBox -query -value $c`;

    print("check_box value = "+ $value +"\n");
}
```

2.3.5 Radio Button

Radio Buttons require the use of both the radioButton and radioCollection mel commands. First we create a radioCollection and then create the radioButtons that need to be part of the collection. The following example creates two radio collections.....

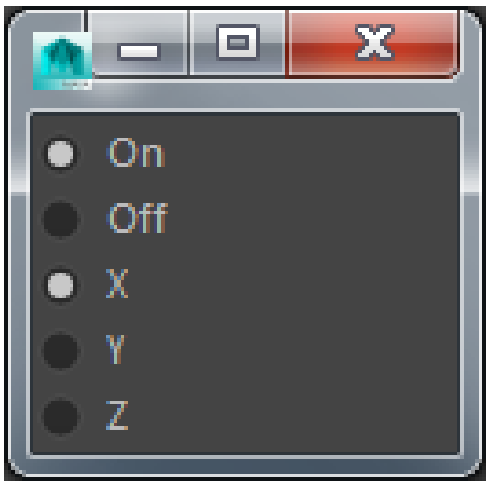


Figure 2.11: radioButton

```
{
    window;
```

```
        columnLayout;

        // create the first radio collection
        $radio1 = `radioCollection`;

        // add some radio buttons to the collection
        $on  = `radioButton -label "On"`;
        $off = `radioButton -label "Off"`;

        separator -w 50 -style "single";

        // create the second radio collection
        $radio2 = `radioCollection`;

        // add some radio buttons to the collection
        $X = `radioButton -label "X"`;
        $Y = `radioButton -label "Y"`;
        $Z = `radioButton -label "Z"`;

        // edit the radio collections to set the required radio button
        radioCollection -edit -select $on $radio1;
        radioCollection -edit -select $X $radio2;

    // now show the window
    showWindow;

    // If you need to query the selected radio button, use...
    $selected = `radioCollection -query -select`;
}
```

2.3.6 Float Fields

Float fields can be used to provide numerical text input into your user interfaces. For integers you will need an intField, for strings you can use the textField control.

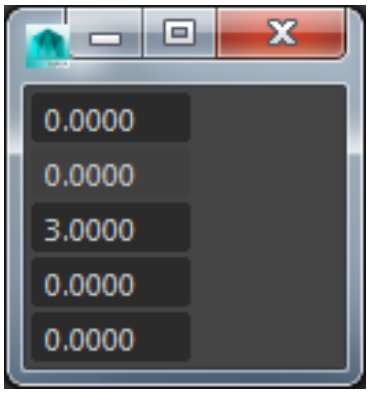


Figure 2.12: float fields

```
// Global Data and Functions. The callbacks need
// to be global for the user interface command to be
// able to call them. Any user interface event will occur
// AFTER the script has finished executing. Any commands
// you call must therefore be available within any scope,
// ie they must be global.

// the name of the float field
global string $floatFieldName="";
```

```
10
11 global proc floatValueChanged()
12 {
13 global string $floatFieldName;
14
15 // query the value from the float field
16 $value = `floatField -query -value $floatFieldName`;
17
18 // print value
19 print("newValue="+ $value +"\n");
20 }
21
22 // window creation scoped to prevent unesseccary
23 // globals being defined
24 {
25 string $window = `window`;
26 columnLayout;
27 floatField;
28 floatField -editable false;
29 floatField -minValue -10 -maxValue 10 -value 3;
30 floatField -precision 4 -step .01;
31 $floatFieldName =
32 `floatField -changeCommand "floatValueChanged();"`;
33 showWindow $window;
34 }
```

2.3.7 Int Fields

Int fields can be used to provide numerical text input into your user interfaces. For floats you will need a floatField, for strings you can use the textField control.

Figure 2.13: intFields

```
1 // Global Data and Functions. The callbacks need
2 // to be global for the user interface command to be
3 // able to call them. Any user interface event will occur
4 // AFTER the script has finished executing. Any commands
5 // you call must therefore be available within any scope,
6 // ie they must be global.
7 //
8 //  There are a couple of ways we can get around having
9 // a global variable, but more on that later...
10 global string $intFieldName="";
11
12 global proc intValueChanged()
13  {
14     global string $intFieldName;
15
16     // query the value from the float field
17     $value = `intField -query -value $intFieldName`;
18
19     // print value
20     print("newValue="+ $value +"\n");
```

```
}

// window creation scoped to prevent unesseccary
// globals being defined
{
    string $window = `window`;
        columnLayout;
        intField;
        intField -editable false;
        intField -minValue -10 -maxValue 10 -value 3;
        intField -changeCommand "intValueChanged();";
    showWindow $window;
}
```

2.3.8 Text Fields

Text fields can be used to provide text input into your user interfaces. For floats you will need a floatField, for int's you can use the intField control. If you require larger multi-line text input then look at the Scroll Field example.

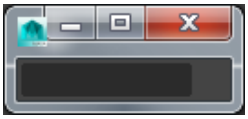


Figure 2.14: text fields

```
// a procedure to be called when the text
// value changes in the textField
global proc textValueChanged(string $control)
{

    // query the value from the float field
    $value = `textField -query -value $control`;

    // print value
    print("newValue="+ $value +"\n");
}

{
    window;
        columnLayout;

        // create the text field
        $field = `textField`;

        // sets the command to call when the text changes.
        // Note that we add the name of the control directly
        // into the command string. This negates the need
        // for a global variable to store the textField name
        textField -edit
        -changeCommand ("textValueChanged("+$field+")")
        $field;
    showWindow;
}
```

2.3.9 Scroll Fields

The scroll field defines a large mulit-line text edit control. For example, Maya's script editor is composed out of two scroll fields; one for the results, one for the code being edited.

```
{
```

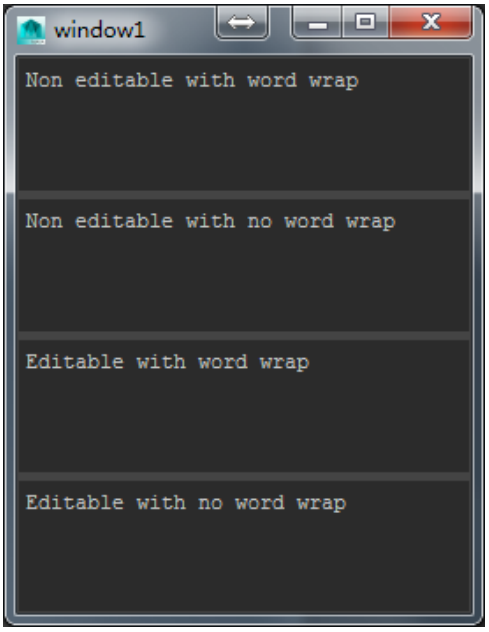


Figure 2.15: scroll fields

```
    // create the window
    window;

        // create a pane layout to hold the 4 scroll fields
        paneLayout -configuration "horizontal4";

        // create 4 scroll fields
        scrollField -wordWrap true
        -text "Non editable with word wrap" -editable false;
        scrollField -wordWrap false
        -text "Non editable with no word wrap" -editable false;
        scrollField -wordWrap true
        -text "Editable with word wrap";
        scrollField -wordWrap false
        -text "Editable with no word wrap";
    showWindow;
}
```

2.3.10 Float Sliders

Float Sliders tend to be used less than the float slider group controls available in mel. This is mainly because a simple slider with no label has *limited* uses.

Figure 2.16: float sliders

```
{
    window;
        columnLayout -adjustableColumn true;
        floatSlider;
        floatSlider -min -100 -max 100 -value 0 -step 1;
    showWindow;
```

```
}
```

2.3.11 Int Sliders

Int Sliders tend to be used less than the int slider group controls available in mel. This is mainly because a simple slider with no label has *limited* uses.

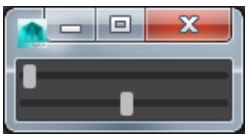


Figure 2.17: int sliders

```
{
    window;
        columnLayout -adjustableColumn true;
        intSlider;
        intSlider -min -100 -max 100 -value 0 -step 1;
    showWindow;
}
```

2.3.12 Float Slider Group

The floatSliderGrp mel command creates a float slider with a text label and a float field input box. This tends to be more useful than the floatSlider on it's own.

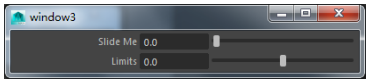


Figure 2.18: float slider grp

```
{
    window;
        columnLayout;
        floatSliderGrp -label "Slide Me" -field true;
        floatSliderGrp -label "Limits" -field true
            -fieldMinValue -50 -fieldMaxValue 50
            -minValue -10 -maxValue 10 -value 0;
    showWindow;
}
```

2.3.13 Int Slider Group

The intSliderGrp mel command creates an int slider with a text label and an int field input box. This tends to be more useful than the intSlider on it's own.

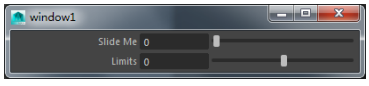


Figure 2.19: int slider group

```
{
    window;
        columnLayout;
        intSliderGrp -label "Slide Me" -field true;
```

```
        intSliderGrp -label "Limits" -field true
            -fieldMinValue -50 -fieldMaxValue 50
            -minValue -10 -maxValue 10 -value 0;
    showWindow;
}
```

2.4 A simple menu

This little example creates a menu and attaches it to a window.

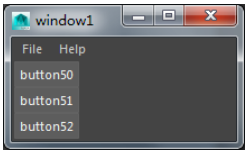


Figure 2.20: menu

```
{
    // create a window with a menu bar
    window -menuBar true -width 200;

        // create a tearOff menu
        menu -label "File" -tearOff true;

        // add the menu items
        menuItem -label "New";
        menuItem -label "Open";
        menuItem -label "Save";
        menuItem -divider true;
        menuItem -label "Quit";

        // add a help menu
        menu -label "Help" -helpMenu true;
        menuItem -label "About Application...";

        // add the other controls
        columnLayout;
        button; button; button;

    showWindow;
}
```

2.5 Attribute Controls

There are a set of user interface controls that are designed to directly manipulate attributes contained within nodes. This means that you don't have to provide callbacks for them to update your attribute values.

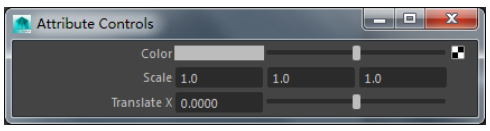


Figure 2.21: Attribute editor

```
{
    // create a sphere and a shading node
    $sphere = `sphere`;
```

```
    $phong = `shadingNode -asShader phong`;

    // create a window to hold the controls
    window -title "Attribute Controls";
    columnLayout;

    // attach a colour control to the phong colour
    attrColorSliderGrp -at ($phong+".color");

    // attach a field group to the scale attribute
    attrFieldGrp -attribute ($sphere[0] + ".scale");

    // attach a float slider to the translate X of the sphere
    attrFieldSliderGrp -min -10.0 -max 10.0 -at ($sphere[0]+".tx");
    showWindow;
}
```

2.6 Dialogs

2.6.1 Colour dialog

```
// create a colour editor dialog box
colorEditor;

// query the result
if (`colorEditor -query -result`)
{
    float $rgb[], $alpha;

    // get RGB
    $rgb = `colorEditor -query -rgb`;
    print("Colour = "+ $rgb[0] +" "+ $rgb[1] +" "+ $rgb[2] +"\n");

    // get alpha
    $alpha = `colorEditor -query -alpha`;
    print("Alpha  = "+ $alpha +"\n");
}
else
{
    print("Editor was dismissed\n");
}
```

2.6.2 Confirm dialog

The mel command confirmDialog brings up a simple little yes/no; ok/cancel type of dialog box. Simply check the return argument from the command to see what was chosen.

```
// create a colour editor dialog box
colorEditor;

// query the result
if (`colorEditor -query -result`)
{
float $rgb[], $alpha;

// get RGB
$rgb = `colorEditor -query -rgb`;
print("Colour = "+ $rgb[0] +" "+ $rgb[1] +" "+ $rgb[2] +"\n");

// get alpha
```

```
$alpha = `colorEditor -query -alpha`;
print("Alpha  = "+ $alpha +"\n");
}
else
{
print("Editor was dismissed\n");
}
```

2.6.3 Prompt Dialog

The prompt dialog in mel allows you to bring up a small window into which can be used to request a specific value from the user.

```
{
    string $text;

    // create a prompt dialog to request the users name
    string $result = `promptDialog
    -title "Hello Window"
    -message "Enter Name:"
    -button "OK" -button "Cancel"
    -defaultButton "OK" -cancelButton "Cancel"
    -dismissString "Cancel"`;

    // if OK pressed
    if ($result == "OK")
    {
        // query the entry typed by the user
        $text = `promptDialog -query -text`;
        print("HELLO to "+ $text +"\n");
    }
    else
    {
        print("fine. I won't say hello then :(\n");
    }
}
```

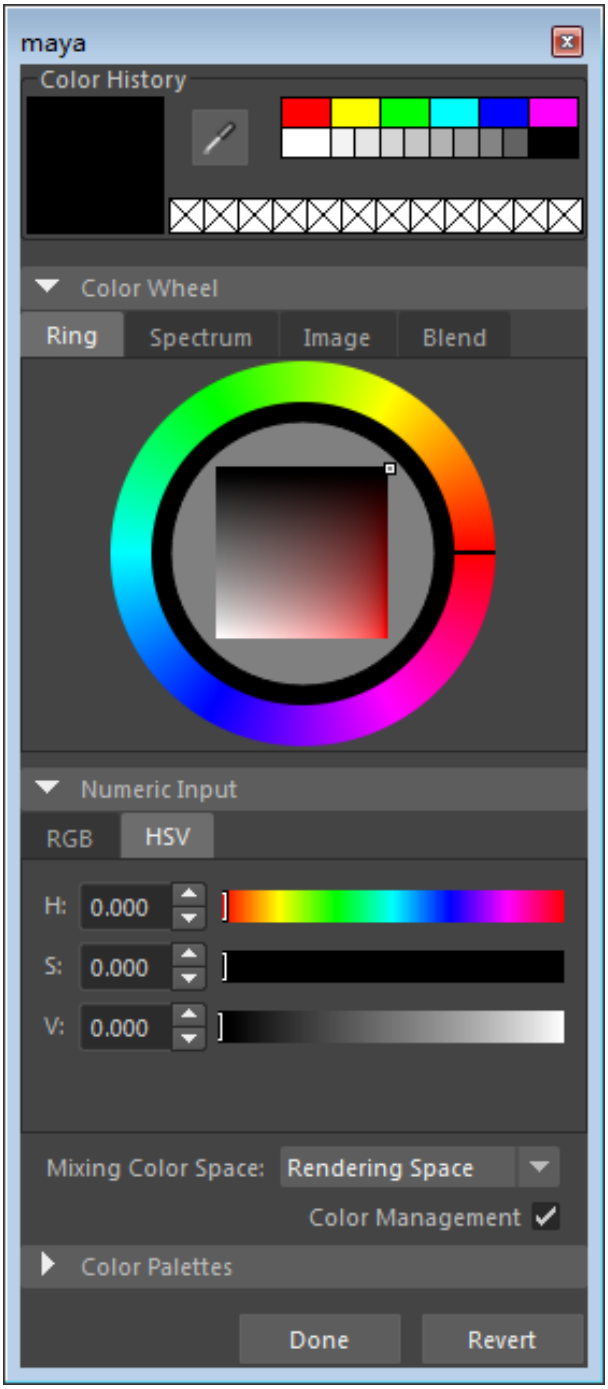


Figure 2.22: colour dialog

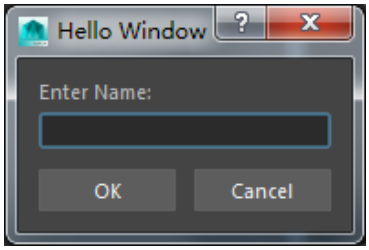


Figure 2.23: prompt dialog