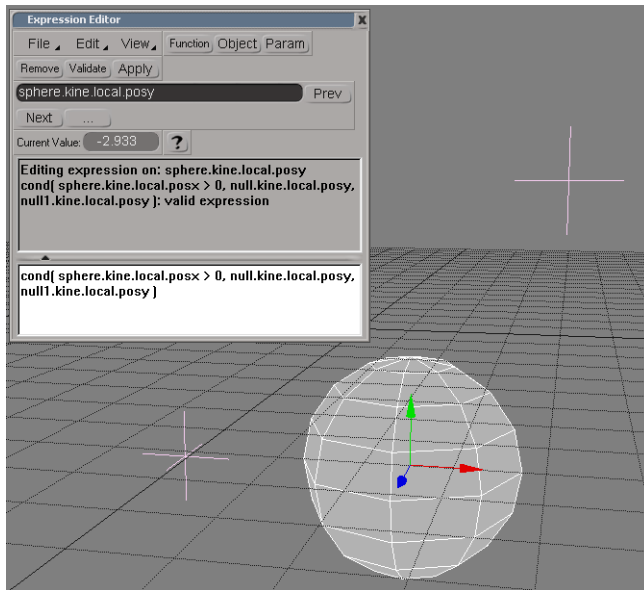


“Expressions for Dummies”

Expressions is the kind of topic that makes many animators say “That’s way too complicated for me!” However, expressions can be surprisingly easy to understand, once you get your feet wet. The best way to understand how to use expressions is to apply some and see what they do ... so let’s go!

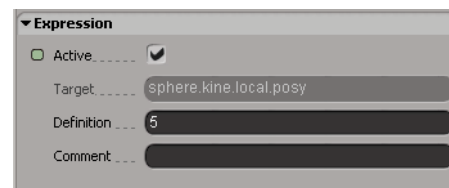
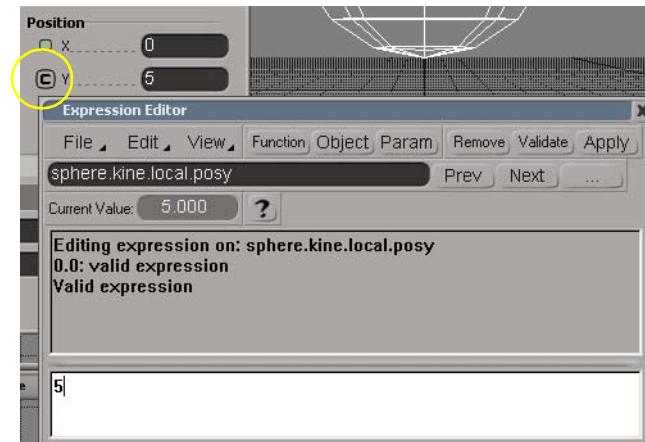


Jumping into Expressions

To understand expressions better, try the following simple exercises. Once you're comfortable with the process and the expression editor, you can go on to more complicated expressions and setups.

Setting Up a Constant Expression

1. Get a primitive sphere (or something equally as exciting).
2. Select the sphere and press Ctrl+k (this opens the sphere's Local Transform property editor).
3. Right-click on the animation icon (green box) of the local Position Y parameter and choose **Set Expression**.
4. In the expression editor, enter the value of 5 in the expression pane and click the **Apply** button, but keep the expression editor open to take a peek at what you just did:
 - At the top of the expression editor is the parameter (sphere.kine.local.posy) that you've set to be equal to 5.
 - Notice that the Position Y animation icon has the letter "C" in it to indicate that it's a constant expression (where the value is a number).
 - An Expression page is added to the Local Transform property editor so that you can easily edit this expression later, including adding comments.
5. Close the expression editor.



Congratulations, you’ve just set the simplest kind of expression known to humankind! The sphere is now fixed at 5 units on its Y axis. You won’t be able to translate it anywhere else in this direction, but you can still move it in X and Z.

Trying It Another Way

Here’s another way of creating a constant expression:

1. Get yet another sphere and set its Translation Y value to 5.
2. Again, press Ctrl+k, right-click on the Position Y parameter’s animation icon, and choose **Set Expression**.

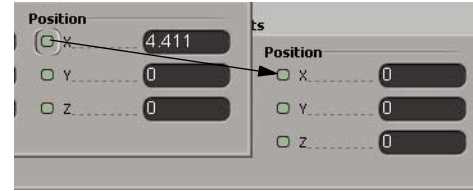
Notice the expression editor opens with the value of 5 already in it. You’ve just created a constant expression the same as in the first exercise.

Creating a Simple Equivalency Expression

You can also easily create simple $A = B$ expressions between parameters:

1. Open a new scene and get a sphere and a null.
2. Translate the null in X a little to offset it from the origin and the sphere.
3. Select the sphere and press Ctrl+k to open its Local Transform property editor. Lock this property editor to keep it open (click its keyhole icon).
4. Select the null and press Ctrl+k to open a Local Transform property editor for it.

5. Drag and drop the animation icon for the null's Position X parameter to the sphere's Position X parameter in the other property editor.



- The expression editor opens with the affected parameter, `sphere.kine.local.posx`. Its expression below is `null.kine.local.posx`, meaning that the sphere takes its Position X value from the null's Position X value.
- The sphere's animation icon for the Position X parameter now has an equal sign (=) in it to indicate the expression.



If there would have been animation on the animation icon being dragged, the animation would have simply been copied to the other parameter, but no expression would have been set.

6. Test the expression by translating the null in X: the sphere should follow in X.
7. Close the expression editor and the two property editors to keep the scene uncluttered.



One More Time

Here's another way to create an equivalency expression:

1. Select the sphere, press **Ctrl+k**, right-click on the animation icon for Position Y, and choose **Set Expression**.

2. In the expression editor, type `null.kine.local.posx` in the white expression pane below and click the **Apply** button.

Now as the null is translated in X, the sphere moves diagonally, getting both its X and Y Position values from the null's Position X value.

More Complex Expressions

Let the math begin! Here are a few more complex examples to try out.

Relational Expressions

You already saw how Parameter B can directly take the value of Parameter A. But you can also use the values from parameter A and modify them.

1. Create a sphere and a cube, just to keep things simple.
2. Animate the cube moving back and forth on its X axis.
3. Select the sphere and open its Local Transform property editor.
4. Set an expression on the Position X parameter.
5. Enter the following into the expression editor:

```
cube.kine.local.posx * -1
```

6. Validate and apply the expression and close the expression editor.

The animation icon for the Position X parameter now has an arrow in it indicating that the parameter is being controlled by a relationship.

7. Play the animation and notice that as the cube moves back and forth on the X axis, the sphere moves the same distance but in the opposite direction (the -1 value did this). If you plotted the Position X animation of the sphere, you would see that its fcurve is the same as the cube's Position X fcurve, except that it would be flipped vertically.

8. Edit the expression in the Local Transform property editor to the following:

```
cube.kine.local.posx * -0.5
```

Notice now the sphere moves in the opposite direction, but only half the distance. If you plotted its Position X fcurve, you would see that it's upside-down and half as tall as the cube's Position X fcurve.

By multiplying the value of the cube's Position X parameter, you're basically altering the shape of the source fcurve. You can make the fcurve taller by multiplying by numbers greater than 1, or shorter by dividing. Adding and subtracting values to the parameter shifts the resulting fcurve left and right.

9. With the sphere selected, choose **Create > Parameter > New Custom Parameter Set** from the Animate toolbar, and name the set *Control*.

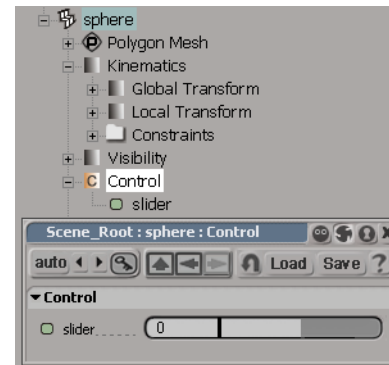
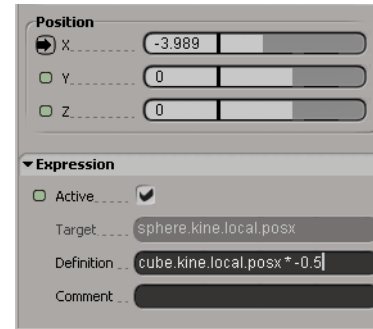
10. Choose **Create > Parameter > New Custom Parameter** (or press Shift+p) to create a custom parameter for the set and name it *slider*.

11. Set the **Minimum** value range to -10 and **Maximum** to 10, and leave everything else at default.

12. Edit the expression for the sphere's Position X parameter again, this time as follows:

```
cube.kine.local.posx * sphere.Control.slider
```

13. In the explorer, find the Control custom parameter set under the sphere's node and click on its icon to open up its property editor. Inside, there should be the parameter you created (slider).



14. Loop the animation of the cube while dragging the custom parameter's slider across its range of values.

Instead of multiplying the cube's Position X parameter by a number like -1 or -0.5, you multiplied it by the slider's value. You can see that this slider now controls the amplitude of how the sphere translates in X relative to the cube translating in X.

You can imagine that when the slider is set to 1, the sphere's Position X fcurve matches the cube's Position X fcurve precisely. As you increase the slider value, the sphere's Position X fcurve grows taller. As you decrease the value to zero, the fcurve flattens out, only to grow again in the opposite direction as you move the slider to negative values.

15. Change the expression so that instead of multiplying, you add the two parameters together:

```
cube.kine.local.posx + sphere.Control.slider
```

Now as you move the slider, the sphere shifts to the right or left.

This sort of mathematical manipulation of fcurves is exactly what's taught in math analysis or pre-calculus. Basically, it's drawing pictures with math equations. A pre-calculus text book can give you all the information you need to know about how to manipulate the shapes of functions using different math operations.

Conditional Expressions

If you're getting into it now, try this simple exercise using a conditional expression.

1. In a new scene, get a sphere.

2. Get two nulls and translate them so that one is above the sphere and the other is below (basically, you just need noticeably different Y positions).
3. Set an expression on the sphere's local Position Y parameter:

```
cond( sphere.kine.local.posx > 0,
      null.kine.local.posy, null1.kine.local.posy )
```

This is a conditional expression, which works according to this format:

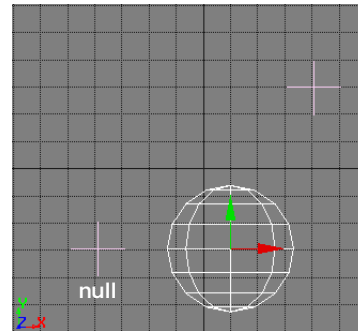
```
cond( True or False Condition, This Value if True, This Value if False )
```



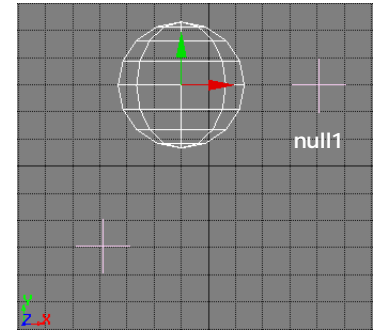
If you choose **Functions > Conditions >**

Condition in the expression editor, this basic formula is added for you in the expression pane so that you don't have to type it all out.

4. Translate the sphere on its X axis to see how it jumps when the value is greater than or less than zero. If it's greater than zero, the Position Y value from the first null is used; otherwise, the Position Y value from the second null (null1) is used.



Sphere's Position X value is **greater** than zero, so it takes the Position Y value of **null**.



Sphere's Position X value is **less** than zero, so it takes the Position Y value of **null1**.

Oscillation Using Cosine

And now for a simple oscillation using the cosine function and the scene's frame rate.

1. In a new scene, get a sphere.
2. Create a custom parameter set for it called *Control*.
3. Create two custom parameters for this set, called *amp* and *freq*, each with value ranges between -100 and 100.

4. Set the following expression on the sphere's local Position X parameter:

```
cos( T * 90 * sphere.Control.freq ) *  
sphere.Control.amp
```



You can also find the cosine function in **Functions > Trigonometry > Cosine** in the expression editor.

T represents Time and is dependent on the frame rate setting of your playback. If it is set to 30 frames per second, **T** equals 1 at frame 30.

5. Set the playback in a loop, play around with the custom **amp** and **freq** sliders (set the **freq** first).

Conclusion

This should be enough for you to understand the kinds of things one can do with expressions. Of course, this is just the tip of the proverbial iceberg. Through expressions, you can create all kinds of custom controls, rigs, animation control systems, and simulations. From here, it's just a matter of experimentation and experience.

For more information about the tools you used, see the following chapters in the XSI Animation user guide:

- Chapter 11: Custom and Proxy Parameters
- Chapter 12: Animating with Expressions

Original material written by Bradley R. Gabe, Technical Director / Quiet Man | NYC Graphics. Updated for XSI version 2.0 and revised by Edna Kruger, a member of the Softimage Education team.

© 2001 Avid Technology, Inc. All rights reserved.
SOFTIMAGE and Avid are registered trademarks and XSI
and the XSI Logo are trademarks of Avid Technology, Inc.
All other trademarks contained herein are the property of their respective owners.

Avid