What is Relativity?

Relativity is a motion and displacement plugin (applied via the Motion Options panel and/or the Object Properties panel) that allows the motions and/or deformations of one item (camera, light, object, or bone) to be related to the motions of another item via mathematical expressions. Below is a picture of the main Relativity motion plugin panel (accessed via the “Options” motion plugin button):
• **Current Item** - This menu allows you to bring up and edit the expressions of any item in LightWave that is currently using Relativity.

You cannot access Relativity morph expressions from the motion panel or vice versa.

• **Disable** - Turning this button on will disable this instance of Relativity for this object, allowing the original keyframed motion to play through untouched. This setting is saved in the scene file for this instance of the plugin as well.

• **Rot in Deg (Rotation in Degrees)** - Allows rotation values to be calculated in degrees, as opposed to the default radians that LightWave uses internally. For most, it is much easier to think in degrees.

• **Explicit WC’s** - This button forces Relativity to calculate the global position of a child object based on just the rotation, scale, and motion of each of its parents. LightWave normally supplies this information, but in the case of Full-Time IK and in cases where a motion plugin controls the motion of a parent or ancestor of the particular child object, LightWave supplies bogus world-coordinate information to a plugin. So, Relativity can go in and reconstruct the position explicitly using position, scaling and rotation data. Using this option can slightly increase the amount of time needed to perform an expression in Relativity.

• **Motion Channels** - It is ultimately the contents of these nine little fields that comprises all that Relativity really is. At its heart, you enter expressions defined in a simple expression language that end up relating the value of each motion channel to something else happening in LightWave. For instance, you could enter in the H slot the expression “X(light,t+2)”. What would happen then is that the expression interpreter would look at the light’s X position at 2 seconds before the time for the current frame and then pass that into LightWave as the heading rotation value for the camera. I could also make the expression 3*X(light,t+2) and then have the X position of “light” multiplied by 3 before being passed as the heading value for the camera.

• **Special Functions** - This slot is used for several special functions that Relativity can perform, like targeting and dynamic parenting.

• **Clear** - Clicking this button will clear everything out of the current panel, allowing you to start over. If you happen to do this accidentally, click Cancel to lose the changes and then come back in.

• **Copy/Paste** - Copies/Pastes the expression into/from an internal buffer. all settings and variables will be copied and pasted as well.

• **Copy to Descendants** - Copies the current set of expressions to the descendants of an object. An example would be in setting up a motion cycle for the thigh of a character. You could then take those same expressions and copy them to the descendants of the current object. NOTE: The descendants will need to already have Relativity applied to them via their own Motion Graph panels. Relativity is unable to do this automatically due to limitations in the LightWave plugin architecture.

• **Load/Save** - Loads and Saves the expression settings To/From a file. The file format is just text with all the parameters listed sequentially in it.

• **Search/Replace** - Search for instances of a text string, and replace it with another. The search can either be case-sensitive or not.

• **Comments** - Put in anything you darn-well please in these slots... they are there to help you remember why you set up an expression the way you did.
• Professors - For those who feel rather math-shy, there are a number of “Professors” that allow you to set up common types of expressions. Note that next to the expression slot for each motion channel, there is an “Ask a Professor” menu choice. For most professors, it is context sensitive (i.e. for the Gears professor, the rotation used in the expression will match that of the slot selected… so the bank slot would be filled with a “B(object,t)”, the heading slot with H(object,t) and so on.). The “Dr. Follow” professor will actually place data in any slot specified and is thus not context sensitive. To use a professor, select it from the choice menu and fill in the blanks. If you’ve filled in everything correctly, your expression will be automatically set up for you. You can then take the expressions created and tweak them further.

A detailed staff listing of the professors and what they do can be found later in this chapter.

• Variables Panel - Above is a picture of the variable panel, accessible via the “Variables” button on the main panel. Relativity has a number of variables available that you can use in your expressions. Basically, each variable can be a sub-expression which then gets dropped into the grand expression for each motion channel. For instance, I could define the sub-expression

\[ X(object_1, t) \times \sin(t \times 5) \]

as a sub-expression for variable “a”. This can then be substituted into the final motion channel expressions.

Say we put the expressions:

\[ 3 \times a \]

and

\[ 2 \times a \]

in the X and Y slots of Relativity… if you expanded things out, you’d end up with:

\[ 3 \times (X(object, t) \times \sin(t \times 5)) \]

and

\[ 2 \times (X(object, t) \times \sin(t \times 5)) \]

which are certainly more unwieldy to manage than putting the sub-expression in A and substituting in “a” where needed.

In addition, note the two buttons, “Shift Variables Up” and “Shift Variables Down” on the panel. What these buttons do is move the variables up or down in the slots, replacing the variable names in all expressions accordingly. The reason for these buttons is if you as the expression designer decide you need some extra slots at the beginning or end of a group of expressions… finding and replacing all references to variables within the expressions as necessary.

The Relativity Morph Plugin
Using the Relativity Morph plugin, found in the Object Properties panel Deform Tab, you can easily morph from one object to many potential targets, using complex expressions. You can even do additive, subtractive, multiplicative, and division-based morphs. There is a single expression slot “MORPH” into which you can enter any number of morph expressions (any other expressions in this slot will be ignored). Each morph expression has the following syntax:

```
<command character>MORPH(target,value)
```

where “command character” can be any of the following:

- &
  - replacement morph - this interpolates from the presently calculated morph into the target.

- +
  - additive morph - adds the values of the target points to the currently calculated morph

- -
  - subtractive morph - subtracts the values of the target points from the currently calculated morph.

- *
  - multiplicative morph - multiplies the currently calculated morph by the values of the target.

- /
  - division-based morph
- division morph - divides the calculated morph points by the point values of the current target

So, a typical morph expression might look like this:

&\text{MORPH} (\text{morph1}, \#a) + \text{MORPH} (\text{morph2}, \#b) - \text{MORPH} (\text{morph3}, \#a + \#b)

where \#a, \#b, etc. were derived from expressions in the variables panel.

You can enter any of the standard Relativity expressions in the variable slots, A through R, and then use these in the MORPH expressions. You can also place expressions directly in the morph value of the MORPH expression as well (e.g. "\text{MORPH} (\text{targ1}, \sin(t))")

You will also notice the options buttons at the top of the morph panel, labeled Evaluate Expression Point-by-point, Evaluate Point using World Coordinates, and Transform Morph Target(s).

- Evaluate Expression Point-by-Point - If this option is on, then the variables $x$, $y$, and $z$ are replaced with the value of each point in the original object, as they are moved. This is useful for creating partial morphs, like effector-morphs, gradient morphs and ripple morphs. If this option is off, then $x$, $y$, and $z$ use the keyframed motion values from the object itself for these variables.

- Evaluate Points using World Coordinates - If this option is on, then the world $x$, $y$, and $z$ values of each point is used in evaluating the morph. This will also cause the morph to be evaluated after bones have been applied. If this is off, then individual points are evaluated using their before-bones local coordinate.

- Transform Morph Target(s) - If this option is on (and it must go hand-in-hand with "Evaluate Points using World Coordinates"), the morph plugin will transform the target points by the motion of the morph target object, allowing you to have the morphed points move with their target.

It is highly recommended that you avoid using functions like

\begin{align*}
\text{DIST} \\
\end{align*}

and

\begin{align*}
\text{ODIST} \\
\end{align*}

in the expressions if you have any of the point-by-point options on... you will be waiting a long time for those expressions to finish with objects that have any significant point count.

There are several example scenes included in the latest example scene archive that show the Rel_Morph plugin in action.

In addition to the object morphing, the X, Y, and Z slots allow for expression based deformation on the X, Y, and Z coordinates of each point in your object.

### The Relativity Expression Language

#### Syntax

The Relativity language syntax is very similar to that of the NewTek “Math Motion” and “Math Morph” plugins, which ends up being a sort of LISP-ish type programming language. Basically, you enter a mathematical expression in a slot, and the plugin will evaluate it, applying the final value of that expression to that channel. In the slots on the variable panel, the expression is applied to that variable, whose value can then be placed in other expressions. Some simple expressions would be:

\begin{align*}
\cdot \sin(t) \\
\cdot \cos(x) \\
\end{align*}

- the sine of the current time value
- the cosine of the x value of the object's own keyframed motion

- adds the X value of blah1 and the Y value of blah2

- multiply whatever's in the "A" slot by the sine of t and add the X scale keyframe value of its own keyframed motion at the current time.

- gets the X motion value of object "something" where it was a half second ago (time is always in seconds)

The goal is to create an expression for each motion channel of interest. For motion channel slots left blank, the keyframed value for that channel will default through to the motion of the object.

Object Names

In order for relativity to grab object names, full names (case insensitive) need to be supplied for all functions, object and bone names also need to be unique in order for Relativity to find the correct object. Object names that end with the ".lwo" extension need to have their names specified in functions in Relativity with their extension.

Null objects should be specified with their exact names (typically, they do not have the ".lwo" extension on their names). There are also 3 predefined object names:

- camera - grabs motion info from the camera
- SELF - grabs information from the item's own keyframe data
- PARENT - grabs information from the object's parent.

Variable Substitutions

If you click on the "Variables" button on the main Relativity options panel, you will notice a panel with a number of additional slots: A through R. These are there to allow the build-up of complex multi-expressions that would become quite unmanageable very quickly if all strung into one string. Relativity will scan each string, looking for a "#" character followed by a variable name and substitute that value in. For example, if I had:

<table>
<thead>
<tr>
<th>X: 3*#i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y: 4*#i</td>
</tr>
<tr>
<td>I: sin(X(something,t))</td>
</tr>
</tbody>
</table>

the value of I would be evaluated and X would be essentially set to

3*sin(X(something,t))

and Y to

4*sin(X(something,t))
Note how using the variables also allows you to remove repetitive sub-expressions from your main expressions. In addition to that, each variable slot can take variables from the previous slots... so, we could have:

\[
\begin{align*}
I & : X(blah,t) \\
J & : Y(more,t) + \#i \\
K & : Z(amore,t) + \#i + \#j
\end{align*}
\]

etc. Relativity also has several special variables:

**#ex**

takes the numeric extension of the object’s name, either the “(1)”, “(2)”, etc.) of a cloned object’s name, or any numeric extension on an object’s filename (i.e. myobject001.lwo, myobject002.lwo). This can be useful in setting up a set of expressions that automatically offset clones and numeric duplicates of objects into their appropriate position when cloned (this was used in the “newtank.lws” example scene to make all the tread links fall into their correct position. A simple example would be:

\[
X: \#ex
\]

Now, anytime this object is cloned, the clones will each offset themselves by one meter in the X direction.

**#frm**

gets the number of the current frame.

**#fps**

gets the current frames-per-second value

**#def**

this passes through the default value for whichever motion channel it’s on.. So, for example, if you had an object keyframed at \(X=2, Y=-3, Z=5\), and then had the following expressions:

\[
\begin{align*}
X & : \#def+2 \\
Y & : \#def-6 \\
Z & : \#def*2
\end{align*}
\]

would change the coordinates of our object to 4, -9, 10.

In addition to the user-defined “slot” variables, Relativity has several predefined variables (these SHOULD NOT BE PRECEDED BY THE ‘#’ SYMBOL):
- the x position of the current item (in local coordinates)

- the y position of the current item (in local coordinates)

- the Z position of the current item (in local coordinates)

- the current time

Comments

You can put whatever you want into the comment fields, which will be saved with the expressions in the scene file. In addition to that, the user can end any expression line with a comment preceded by either a double-forward-slash "//", or a forward-slash followed by an asterisk "/*", Relativity will ignore anything after the comment character and will not send that part of the expression on to the parser/compiler/evaluator. An example would be the following, entered in the X field of the main panel:

X(object1,t) //here we’re getting the X value of an object

or

X(object1,t) /*here we’re getting the X value of an object

Functions

Relativity supports all of the standard math functions supported by LightWave. These can be found listed in the appendix in the Reference manual. In addition, Relativity supports the following functions (NOTE: the function names are case sensitive, X is not the same as x). It should also be understood that object names containing these functions in their proper case may confuse the Relativity expression parser... so, for example, don’t include an object's name as “myXObject”, instead, change the name to "myxobject" and all will be well with the world. IT IS HIGHLY RECOMMENDED THAT ALL OBJECT NAMES USED IN RELATIVITY EXPRESSIONS BE DONE IN ALL-LOWERCASE:

X(object, time expression)

Y(object, time expression)

Z(object, time expression)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>Gets the x, y, z, heading, pitch, bank, x scale, y scale, and z scale of “object”, respectively.</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>XS</td>
<td></td>
</tr>
<tr>
<td>YS</td>
<td></td>
</tr>
<tr>
<td>ZS</td>
<td></td>
</tr>
<tr>
<td>XW</td>
<td>Gets the world coordinate position of “object”.</td>
</tr>
<tr>
<td>YW</td>
<td></td>
</tr>
<tr>
<td>ZW</td>
<td></td>
</tr>
<tr>
<td>XL</td>
<td>Converts the global coordinate of “object” into the local coordinate space of the object to which we’re applying Relativity.</td>
</tr>
<tr>
<td>YL</td>
<td></td>
</tr>
<tr>
<td>ZL</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```
X(object,t)
```

This would get the X position of “object” at the current time.
These functions return a normalized vector representing the motion of the object. This can be useful, when combined with the vector parameters below, to gauge how much of the object’s motion is forward and how much is side-to-side.

Example:

\[ X: \text{MOTX}(\text{object},t) \]
\[ Y: \text{MOTY}(\text{object},t) \]
\[ Z: \text{MOTZ}(\text{object},t) \]

would cause our Relativity instance to “point” in the direction of motion of “object”.

\[ \text{UPX}(\text{object},t) \]
\[ \text{UPY}(\text{object},t) \]
\[ \text{UPZ}(\text{object},t) \]
\[ \text{FORX}(\text{object},t) \]
\[ \text{FORY}(\text{object},t) \]
\[ \text{FORZ}(\text{object},t) \]
\[ \text{RITX}(\text{object},t) \]
RITY(object, time expression)

RITZ(object, time expression)

These are the Relativity vector parameter functions, and return a normalized vector representing the orientation of “object”’s local X, Y, and Z axes in global space (UP = Y, FOR=Z, RIT=X).

OBJDIZ(object, time expression)

evaluates the dissolve factor of “object”, which can then be used in your expressions.

Example:

OBJDIZ(myobject,t)

will evaluate the dissolve amount for “myobject”.

BONERX(bone)

BONERY(bone)

BONERZ(bone)

BONERH(bone)

BONERP(bone)

BONERB(bone)

BONERL(bone)

Get the rest X, Y, Z, heading, pitch, bank, or rest length of “bone”.

Example:

BONERX(bone) + BONERY(bone)
adds the X and Y rest positions of “bone” together.

\[ \text{CAMZOOM}(\text{time expression}) \]

\[ \text{CAMHORZ}(\text{time expression}) \]

\[ \text{CAMVERT}(\text{time expression}) \]

\[ \text{CAMFL}(\text{time expression}) \]

\[ \text{CAMFD}(\text{time expression}) \]

\[ \text{CAMFSTP}(\text{time expression}) \]

\[ \text{CAMBLR}(\text{time expression}) \]

Gets the zoom factor, horizontal view angle, vertical view angle, focal length (in millimeters), focal distance, f-stop, and motion blur factors, respectively, for the camera at time “time expression”.

Example:

\[ Z: Z(\text{CAMERA,t}) + \text{CAMFD}(t) \]

would position a null right at the focal point of the camera, provided it’s looking down the Z axis and unrotated.

\[ \text{LIGHTCON}(\text{light},\text{time expression}) \]

\[ \text{LIGHTEDG}(\text{light},\text{time expression}) \]

\[ \text{LIGHTRED}(\text{light},\text{time expression}) \]

\[ \text{LIGHTGRN}(\text{light},\text{time expression}) \]

\[ \text{LIGHTBLU}(\text{light},\text{time expression}) \]
Gets the cone angle, soft edge angle, red color, green color, blue color, respectively of “light” at time “time expression”.

Example:

```
LIGHTCON(light, t)
```

would get the cone angle of “light” at time ‘t’.

```
XCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
YCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
ZCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
HCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
PCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
BCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
XSCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
YSCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

```
ZSCYCLE(object, driving expression, start time in seconds, end time in seconds)
```

Cycles the object’s motion from the start time to the end time… whenever the decimal portion of the driving expression is 0, the position of the object at the start time is copied… if it is .99999, the position at the end time is copied, if it is 0.5, the position halfway between the start and end times is returned, etc.
Example:

```
XCYCLE(SELF, t, 0, 0.333)
```

This would cycle the X values between 0 and 10 frames (frame 10 = 0.333 seconds), repeating the cycle each time the time value ticks off a second.

```
XMINPATH(object, following distance, time expression)
```

```
YMINPATH(object, following distance, time expression)
```

```
ZMINPATH(object, following distance, time expression)
```

```
HMINPATH(object, following distance, time expression)
```

```
PMINPATH(object, following distance, time expression)
```

```
BMINPATH(object, following distance, time expression)
```

```
XSMINPATH(object, following distance, time expression)
```

```
YSMINPATH(object, following distance, time expression)
```

```
ZSMINPATH(object, following distance, time expression)
```

Gets the motion value at a minimum path distance of “following distance” behind the object’s position at time t.

Example:

```
XMINPATH(leader, 1, t)
```

This would get the X coordinate of object “leader” 1 meter in path-length behind its current position.

```
TARGET(object, time expression)
```

Entered in the special expression field for the object, target will target the heading and pitch of the item toward the target object, pointing the local Z axis at the targeted object.
Example:

TARGET(object, t)

would target the position of “object” at time t.

MATCH(object, time expression)

This function will match an unparented object to correspond with the object listed in the MATCH function, no matter how deeply “object” is buried in a hierarchy (including bones and child bones). Useful for such things as attaching objects to bones properly. MATCH is a special function, and as such, should go in the Special Functions slot.

Example:

MATCH(footbone, t)

This would match the position of our object to that of “footbone” at time t.

GAP(object1, time expression1, object2, time expression2)

This function measures the gap between the global position of object1 at time “time expression 1” and object2 at “time expression 2”

DIST(object, time expression)

Gets the total path distance traveled (in global coordinates) from time 0 to the time specified.

Example:

DIST(whatever, t-0.5)

Gets the total path distance traveled by “whatever” from time 0 up to 15 frames ago.

ODIST(object, time expression)

This returns the oriented distance traveled by the object, with the orientation being determined by the direction of travel and where the object’s own local Z axis is pointing. Using this distance function, it is possible for an object to “walk” forward and backward, stepping properly as it does so. The original DIST function did not take object orientation into account and will return the total path distance traveled by an object regardless of the direction of the path motion with respect to its orientation.

Example:

ODIST(PARENT, t)

Gets the oriented distance traveled by the parent of our object.

MOVEVEC(velocity, time expression)
Entered in the Special Functions slot, this will move the object along the Z direction (wherever this happens to be pointing) at the specified velocity (in meters/second). Using this function, you can simply rotate your object and have it smoothly fly wherever you need it to go. This function is perhaps most useful for situations where you have the rotation of an object controlled by some sort of “virtual steering mechanism” (like a virtual joystick or steering wheel) and want the object to respond to the rotation changes by moving in the new direction.

Example:

\[
\text{MOVEVEC}(X(\text{controller},t), t)
\]

Moves the object by the velocity indicated by \(X(\text{controller},t)\), incorporating the rotations of our object up to the current time to determine its current position.

\[
\text{SUMX}(\text{object}, \text{time expression})
\]

\[
\text{SUMY}(\text{object}, \text{time expression})
\]

\[
\text{SUMZ}(\text{object}, \text{time expression})
\]

\[
\text{SUMH}(\text{object}, \text{time expression})
\]

\[
\text{SUMP}(\text{object}, \text{time expression})
\]

\[
\text{SUMB}(\text{object}, \text{time expression})
\]

\[
\text{SUMXS}(\text{object}, \text{time expression})
\]

\[
\text{SUMYS}(\text{object}, \text{time expression})
\]

\[
\text{SUMZS}(\text{object}, \text{time expression})
\]

This function adds the value of the object’s respective motion channel incrementally, once per frame. So, for example, if you have an object keyframed to \(X=0.5\), at frame 1,

\[
\text{SUMX}
\]

would equal 0.5, at frame 2,
SUMX

would equal 1.0, etc. To use the SUM functions as a pseudo-time function (for throttle type controls), you will probably have to multiply the SUM expression by 1/(frames per second rate), i.e. 1/30 or 1/24, etc.

Example:

```
SUMX(blah, t)
```

This would sum the X component of object “blah” from time 0 to the present time.

```
IF(expression1 <comparison character(s)> expression2)
```

the IF function will evaluate expression1 and expression2, compare them using the <comparison> character(s) and evaluate to 1 if the comparison is true, 0 if it is false. The following comparisons are supported by IF:

- `<` - less than
- `>` - greater than
- `<>` - not equal to
- `<=` - less than or equal to
- `>=` - greater than or equal to

Example:

```
IF(X(blah,t) < 0)
```

This will evaluate to 1 if the x coordinate of “blah” is less than 0, and will evaluate to 0 if the x coordinate of “blah” is greater than or equal to 0.

```
AND(expression1,expression2)
```

Performs a logical AND between two expressions, if expression 1 and expression 2 are BOTH not 0, then the AND function evaluates to 1, if either or both are 0, then the AND returns 0.

Example:

```
AND(IF(X(blah,t)<0),IF(Y(blah,t)>0))
```

would return 1 if the X value of “blah” is less than 0 and the Y value of “blah” is greater than 0, otherwise, the AND would return 0.

```
OR(expression1,expression2)
```

Performs a logical OR between two expressions, if either or both expression 1 and expression 2 are not 0, then the OR will return 1. If they are both 0, the OR will return 0.

Example:

```
OR(IF(X(blah,t)<0),IF(Y(blah,t)>0))
```

Would return 1 if either the X value of “blah” is less than 0, or the Y value of “blah” is greater than 0, or both. If neither is true, then the OR would evaluate to 0.
**NOT(expression)**

NOT will return 0 if the expression's value is not 0 and 1 if it is zero, i.e. a logical NOT operation. This is most useful for negating IF() expressions.

Example:

```
NOT(IF(x>0))
```

Will return 1 if the x value of our item is not greater than zero, and 0 if it is greater than 0.

**COND(expression <comparison character(s)> expression, trueresult, falseresult)**

This function implements a conditional expression, basically giving the user an IF... ELSE kind of logical decision-making. The first part is identical to the format of the IF statement, this is followed by an expression to evaluate to if the condition is true (i.e. “trueresult”). If the condition is false, then “falseresult” is returned instead.

Example:

```
Y: COND(X(self,t)<=0,1,Z(object,t))
```

In this example, if the X position of the object itself is less than or equal to zero, COND will evaluate to 1. If it is greater than zero, the COND expression will evaluate to the Z value of “object”.

**FT(framenum)**

Converts a frame into a time value in seconds (as Relativity equations use seconds to calculate the time). NOTE: framenum must be a constant value... it cannot be an expression.

Example:

```
FT(30)
```

Would convert to 1.0000, or 1 second, in the typical 30 fps scenario.

**EVTIME(expression)**

This function is a very powerful one, using it, you can set up an animation to play back when a specific trigger condition occurs. The way this function works is it will return a time value, indicating the last time that “expression” changed from 0 or less than zero, to greater than zero, most typically done by some form of IF statement. As long as “expression” remains greater than 0, EVTIME will keep ticking off time. Whenever the expression is 0, or less than zero, EVTIME will return 0. EVTIME will also return 0 if the expression starts out as true (i.e. greater than zero) without having switched from a false condition to a true condition. NOTE: EVTIME will only evaluate “expression” at frame boundaries, not at interframe time values in order to save calculation time. If “expression” changes from false to true at an interframe time, the time at the next frame will actually start the count.

Example:

```
Y(SELF,EVTIME(IF(Y(botfoot,t) < 0.1)))
```
This would start playing back the animation of the object itself once the Y value of botfoot gets reasonably close to zero. Something like this could be used to simulate camera shake whenever a bot’s foot hit the ground.

Another Example:

```
A: EVTIME(IF(GAP(object,t,sensor,t) < 1))

B: COND(#a <= FT(5), #a, FT(5))

C: EVTIME(IF(GAP(object,t,sensor,t) > 1))

D: COND(#c > 0,FT(5)+#c,#b)

X: X(SELF,#d)

Y: Y(SELF,#d)

Z: Z(SELF,#d)
```

In this example, say we have a door keyframed to open from frames 0 to 5 and keyframed to close from frames 5 to 10. In this example, an event trigger is thrown by A when object and sensor come within one meter of each other, causing the open animation to play. However, we don’t want the close animation to occur until our object gets far enough away. So, the B expression limits the playback of A to frames 0 through 5 by returning the time value for 5 frames if A gets bigger than that. The C expression is set up so that it’s triggered when the object moves further than a meter away. Finally, the D expression is set up to return the C value if it has a value, and the B expression otherwise. The net result of all this gibberish is that when “object” gets close to “sensor” the doors fly open. When it moves away from “sensor”, they fly shut again.

```
EVONE(expression)
```

This function is the same as EVTIME, except it doesn’t reset when the condition becomes false again. EVONE remains triggered no matter what happens to the tracking event.

```
VX(object,t)

VY(object,t)

VZ(object,t)

VH(object,t)
```
computes the directed velocity of an object along the appropriate motion channel. NOTE: This value is sensitive to direction, so an object moving in the positive direction at the same speed as one moving in the negative direction would have opposite X velocities, one positive and the other negative, respectively. NOTE: All velocities are in meters per second (except rotational velocities which are in degrees per second if the “Use Degrees” button is on, in radians per second if “Use Degrees” is off), if you need a different velocity measurement, such as miles per hour, you will need to multiply the velocity value by the appropriate constants.

Example:

\[ \text{VX(object, t)} \]

gets the X velocity of our object at time t (in meters per second)

\[ \text{SPEED(object, t)} \]

\[ \text{SPEEDW(object, t)} \]

Gets the speed in local coordinates or world coordinates, respectively, of the object at time t. Speed is irrespective of direction, and takes all axial motions into account.

Example (a simple speedometer):

\[ B: 180 \times \text{SPEED(car, t)} \]
computes the acceleration of an object along the motion channel. In this case, acceleration is direction insensitive, returning the change in the absolute value of the velocity on the channel in question.

Example:

```
AH(object, t-0.1)
```

this would get the heading velocity of “object” at .1 seconds behind the current time.

```
SPDCHNG(object, t)
SPDCHNGW(object, t)
```

gets the change in speed over time of an object at time t, in local or world coordinates, respectively. This can be considered akin to acceleration, but in all directions, rather than just one axis.

Example:

```
Y: 30*SPDCHNG(myobject, t-X(timecontrol, t))
```

This would measure the rate of speed change for object “myobject” at the current time minus the X value of “control” (in seconds).

```
ABSDEG(expression)
```
converts a rotation value in "expression" into an absolute degree value between 0 and 2\*PI radians, or between 0 and 359 degrees if the "Use Degrees" button is checked on. This can be useful when you have something continually rotating, and yet you want to create a behavior that works between a set of rotation values, like bones placed in a tire to do tire-bulge.

Example:

\[
\text{ABSDEG}(\text{P}(\text{obj},t))
\]

would have the following values with corresponding values for the pitch of "obj".

<table>
<thead>
<tr>
<th>Value of Pitch of obj</th>
<th>Value of ABSDEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 degrees</td>
<td>30 degrees</td>
</tr>
<tr>
<td>360 degrees</td>
<td>0 degrees</td>
</tr>
<tr>
<td>740 degrees</td>
<td>20 degrees</td>
</tr>
</tbody>
</table>

\[
\text{FRAC(controller object, # of frequencies)}
\]

gets a fractal noise value (a single value between 0 and 1.0) at the global position of an object, using the controller object to determine the position, rotation, and scale of the fractal texture. The way this works is as follows:

Relativity will grab the global position of an object and transform that coordinate into the coordinate space of the controller null. Using this information, and the # of frequencies value, Relativity will apply a Perlin fractal turbulence function to the coordinate and wrangle out a single value between 0 and 1 for that point.

Example:

\[
\text{X: X(SELF,t) + 0.5*FRAC(fracnull,3)}
\]

this will add a bit of fractal noise to the X position of our object

**Tips for using Fractal noise:**

1. Since the FRAC function will only return a single value for the coordinate of the item involved, you may want to use several fractal controllers to apply different fractal values to different motion channels of an item to shake things up a bit more.
2. The fractal controller null determines the size, position, and rotation of the fractal texture permeating throughout space. So, for example, you have an object sitting still, but you want it to wiggle a bit, no problem, move or rotate the fractal controller object(s) and the fractal texture at the object’s location will change. Another example: Say you want an object to bump around very quickly, changing values dramatically at almost every frame, just scale the fractal controller null to a very small size, and you'll reduce the size of the fractal "lumps" causing the texture to vary much more quickly from point to point. Conversely, larger scale values for the fractal controller will cause the variation to be much more smooth between points.

Remembering these tips can give you a high degree of control of the fractal randomness.

\[
\text{NOISE(expr)}
\]

returns a noise value based on the value of "expr". Basically, you feed it a value, and out pops a fractal noise value.

Example:

\[
\text{NOISE(t)}
\]

will return a noise value based on the current time.
NOISE(3*t) will return a more “jagged” noise value.

BLEND(value 1, at point 1, value 2, at point 2, current position)

TBLEND(value 1, at point 1, value 2, at point 2, current position)

Blend can be used to linearly blend between values 1 and 2, with the result of blend equaling “value 1” when the “current position” expression is at “at point 1”, and will return “value 2” when the “current position” expression is at “at point 2”. BLEND basically sets up a linear interpolation relationship between the values, and will over-interpolate if “current position” is less than “at point 1” or is greater than “at point 2”...

Examples:

BLEND(0,3,5,6,X(blender,t))

this would blend between the values 0 and 5, with 0 being returned by BLEND when X(blender,t) is 3, and 5 being returned when X(blender,t) is 6. Note: if X(blender,t) were 9, then the blend function would return 10 for its value, as it overinterpolates based on the value of X(blender,t). If this were a TBLEND instead, then the maximum value that the BLEND function would ever have would be 5, no matter how high X(blender,t) got.

BLEND(X(obj1,t),0,X(obj2,t),1,X(blender,t))

this would blend from the X motion channel of obj1 when X(blender,t) is 0 to the X motion channel of obj2 when X(blender,t) is 1. So, basically, X (blender,t) can be used as a toggle to switch the motion of our item from obj1 to obj2.

The Relativity Staff Directory

The following information will give you an overview of the professors in Relativity and what they do.

- Dr. Item Picker
- Dr. Wheel Rotator
- Dr. Oscillator
- Dr. Gear Grinder
- Dr. Motion Blender
- Dr. Dist Maintainer
- Dr. Camera Shaker
- Dr. Target
- Dr. MoveVec
- Dr. Equation Maker
- Dr. Follower
- Dr. Delayer
- Dr. Cycler
- Dr. Blend Machinist
- Dr. Event Maker
- Dr. Morph-o
- Dr. Matcher
- Dr. Snake Maker

Most of the professors in Relativity are accessed from the menu next to an expression slot on the Relativity panel. What they do, is allow the point-and-click automated set-up of many commonly used expression set-ups. It is quite likely that this list will expand in future releases of Relativity. One thing to note about the professors: some professors are context sensitive, they will only set-up the expression for one motion slot at a time, and may change how they set up the expression based on which motion slot they’re being run from. Other professors work on a more global level, and can set up multiple motion expression slots all at once. A few will even commandeer some variable slots. This will be noted as we go through the documentation, professors that set up multiple slots will have the statement “Multi-slot professor” before their description.
**Dr. Item Picker**

Dr. Item Picker is a simple professor panel that will blast an object name onto the end of whatever you are typing. So, for instance, if you were typing:

```
2+X(
```

and then forgot the name of the object you wanted, you could use Dr. Item picker to pick the name you’re interested in… so you would end up with:

```
2+X(mynull
```

and then you could add the “,t)” to finish off the expression. If you need to insert an object name into an existing expression, like:

```
2*sin(XW(blah,t))
```

and realize that you forgot your object name and need to replace “blah” with the correct name, use Dr. Equation Maker below.

**Dr. Equation Maker**

This professor is the most powerful, complex, and yet easy-to-use in your arsenal. It turns the task of hand-building an expression into an exercise in point-and-click. Now, whether or not your expression does what you expect, that’s up to you, but this professor will do all it can to help steer you in the right direction. On this panel, you have a total of eight menus that allow you to pick eight different possible elements that make up your expressions: LightWave’s math constants and functions, Relativity’s arsenal of expression functions, Rel’s special functions (for use only in the “Special Function” slot), Object names, bone names, light names, and special reserved Relativity objects (PARENT, SELF, camera, etc.).

To use this professor, simply select what you want from a menu and click it into place. As you select items, the text control at the top will give you a description of the choice you’ve made, to make sure you have what you’re looking for… in this sample image, X(obj,t) has been selected, and the description tells you what this Relativity function does. If you click **Add Choice to Expression**, the professor will append your choice to the current expression in the edit field.

You can also tell this professor that you want to insert a choice into the middle of your current expression, rather than just at the end. By placing an '@' character wherever you want an insertion point, you can make the professor insert your next choice where the '@' is, rather than at the end of the expression. So, we could start by selecting 'X(obj,t)' and clicking that into place. Then we could edit out the 'obj' and replace it with an '@'. Finally, we could select the actual object name we were interested in from the object menu, say it was called 'heavynull'. Once we click the **Add Choice to Expression** button, 'heavynull' will replace the '@' character, turning our expression into

```
X(heavynull,t)
```

**Dr. Wheel Rotator**

This professor has a simple task, making your object rotate as it tracks the motion of some other object. To use it, pick the object that will drive this rotation (this will most often be the direct parent, or an otherwise ancestor of your object). However, there is nothing that can stop you from letting the object’s own motion drive it’s own rotation. Here is a field-by-field break-down of the panel:

- **Object Driving the Rotation** - This is the object whose motion will cause the rotation of our current object to occur.
- **Pick Vehicle** - Brings up the object picker panel to get a point-and-click selection of the object you want.
- **Diameter** - Here you enter the diameter (i.e. the total distance across) of the wheel.

**Dr. Follower - Multi-slot professor**
This is the professor for setting up following motions of all kinds. You can set up many kinds of relationships, like a delayed following of another object, or have an object follow at a specified distance. You can even set up an offset-following motion where an object or group of objects will follow a leader, yet maintain a respectable distance from it. Here is a break-down of the fields:

- **Object to follow** - This is the object we want to follow… this is hopefully not a hard concept to grasp
- **Pick follow object** - allows point-and-click selection of object
- **Channels to follow** - allows you to determine which channels we’re going to follow on… this will set up multiple expression slots at one time.
- **Offset Motion by Difference at Frame 0** - If this button is clicked on, then this object will follow its leader in a “flocky” sort of way… so the follower object will move with the leader, but maintain its relative position with respect to the leader.
- **Follow at a specified distance** - This slot, if it contains a value, specifies that we want to follow at a specific path distance behind the lead object. This is useful for things like roller coasters, trains, etc. etc. where you need successive cars to maintain an exact distance behind the leader or behind the car in front of it. The distance will be maintained no matter how fast the lead object moves.
- **Follow with a constant time delay** - This slot, if filled, will cause the follower object to either lag behind or anticipate the motion of the lead object by a specified time delay.
- **Optional Time Delay Control Object** - This slot will allow you to specify an object (often a null) as a delay controller.
- **Channels to Control** - Select the channel(s) for the delay object to control. If you only select the X channel, the X channel of the object will control the delay on ALL THE MOTION CHANNELS of our follower object. Otherwise, the X channel of the delay control will determine the X delay, the Y channel will determine the Y delay. Unclicked channels will not have the delay controller specified in them.
- **Use Object Extension for Follow Delay** - This will multiply the object’s numeric extension or clone number (i.e. the (1) in ‘null (1)’ or the 3 in obj003.lwo) by the delay factor, either the time delay or the distance delay. This way, you could set up an expression for one object and copy it verbatim to other objects and have them lock into the right place behind the leader.
- **Extension Offset** - This is the offset for the extension… for instance, if you want object001.lwo to not have any delay, but object002.lwo to delay by one delay factor, you could make the offset 1, this would zero out any delay for object001 but turn it on for subsequent objects.

Dr. Oscillator - Multi-slot professor

This professor is the one for setting up any and all oscillating motions (i.e. objects swinging back and forth, bobbing up and down, wigging around, weaving side-to-side, etc.). The setup of this panel is quite simple:

- **Channels to wiggle** - Select any and all channels that you want to oscillate.
- **Wiggles per second** - This determines the frequency of the oscillation.
- **Frequency Controller** - You can optionally have a frequency controller instead, it’s set up so the X channel of the controller object controls the frequency (this can always be edited after the fact to a different channel).
- **Wiggle Intensity** - Determines the intensity of the oscillation.
- **Intensity Controller** - You can optionally have an intensity controller whose X channel controls the oscillation intensity.

Dr. Delayer - Multi-slot professor

Dr. Delayer allows you to delay an object’s own motion by an offset in time. This is useful for moving the dreaded “locked 0 keyframe” to another time, so cyclic motions can start off at frames other than 0. It’s also possible, using a delay controller, to soften acceleration and deceleration much more than when using just tension on your spline values. Here’s a breakdown of the fields on this professor:

- **Fixed Time Delay** - Enter a value (in seconds, or add a ‘F’ at the end to specify frames) and your motion will be delayed by that amount.
- **Delay Controller** - Choose an object to control the amount of the delay, which makes the delay factor animatable.
- **Channels of the controller object** - This specifies which single channel of the delay controller will control the delay. You can also choose “all channels” to get a channel-by-channel delay control.
- **Delay on which channels** - Pick the channels to set up a delay on.
- **Tie Delay to Object Extension** - Use the object number extension in the expression as a multiplier by the delay factor. This is useful for setting up a sequence of numbered objects to delay by specified amounts.
- **Extension Offset** - This value is subtracted from the object extension before applying it to the expression.

Dr. Gear Grinder

Dr. Gear Grinder is great for relating the rotation of one object to the rotation of another. You pick the controlling gear and then tell it the radius or diameter of each gear (note: you must use either radius or diameter values for both slots). If you are actually setting up gears, you could specify the number of teeth for each gear instead of radius.

Dr. Cycler

Dr. Cycler is the ultimate professor for setting up all sorts of nifty cyclic relationships in LightWave. Before you run the professor, though, you need to have a cyclic motion keyframed for your object. Dr. Cycler will take that cyclic motion and figure out a way to apply that motion over-and-over-again (i.e. cyclically) over the course of your animation. Here’s what all the fields mean:

- **Motion Channel to Cycle on** - The motion channel the cycle will be applied to.
- You can select any object as the source of the cycle. In many cases, you would select “SELF” to choose the object’s own keyframed motion as the basis for the cycle.
- First and Last frame of the Cycle - Again, this should be self-explanatory.
- Object driving the Cycle - This is the object whose motion is driving the cycle, unless you select “time” as the driving function, which makes the object irrelevant.
- Driving function - Choose what aspect of the object’s motion is going to drive the cycle. Most of the options should be clear, path distance means the absolute path distance traveled by the object with no sense of orientation. Oriented distance will grow larger as the object moves “forward” and grow smaller as the object moves “backward”. Oriented distance is probably what will be most often used. NOTE: if you happen to have constructed your object so that its “forward” end faces down the negative Z axis, you will need to go in by hand and edit the expression to have a negative ODIST value.
- Stride Length - Quite often a cycle will need to repeat over a specific distance (like 2 times the step distance of a character, or the distance around the path of a tank tread unit), the distance over which the cycle repeats is known as the “stride length”.
- Optional Length Delay - This field can be used to push the cycle back over its stride length, used to offset opposing legs in a walk cycle, or pushing one element of many back down its path a little bit.
- Tie length delay to object extension - Use the clone number (i.e. the 3 in “null (3)”, or the 10 in “obj10.lwo”) as a multiplier for the length delay. You can use this field to create an expression that should automatically tailor itself to clones of that same object.
- Extension offset - This value is subtracted from the object’s extension and can be used to turn the clone number of “null (1)” to 0, so the first object doesn’t have an offset along its path.

Dr. Motion Blender - Multi-slot professor

Dr. Motion Blender can be used to blend between the motion paths of two objects interactively during an animation. What applications that might have in your animations is entirely up to you. Here is a breakdown of the fields:

- Object 1 and Object 2 - Pick the objects whose motions you want to blend between. NOTE: if you put SELF in for one of the objects, it would blend this object’s own keyframed motion with something else.
- What Channels will we be blending between - Pick one or more motion channels for the motion blend.
- Constant Blend Percentage - Pick a constant blend percentage between the motions.
- Blend Control Object - If this field is filled, it negates the blend percentage field. Use an object to control the blend.
- Channels of the controlling object - You can choose a single channel of the controlling object to control the blend on all channels of our blend object (for instance, the Y value of our object would control the blend as it moved from 0 to 1). If you pick “All Channels”, then the X value of the control object will determine the X blend percent, the Y value will control the Y blend, etc.
- Truncated Blend - If the control object’s values get above 0 or 1, cut them off at 0 or 1, otherwise, you’ll get an “overblend”.

Dr. Blend Machinist

Dr. Blend Machinist is one of the most powerful professors. Using this panel, you can set up lots of interesting relationships between objects, where one channel of one object control the action on one or more channels of another object. For instance, imagine a foot where you need the toes to curl as it lifts off the ground. Or imagine a mechanical system where you need a piston to expand and contract with a rotation. Or imagine a key-state animation, where you can control the opening and closing of multiple bones/segments using a single control null (or group of nulls)… basically any kind of “reaction” to some slider value… Blend Machinist can do these and much more. Here’s a breakdown of the fields:

- Truncated Blend - This will cut off the blend as it reaches its min and max values, a non-truncated blend will continue the blend beyond the min and max values.
- Type of Blend - A blend between values will do just that, blend between two values. A time blend will push the time value back and forth and play back a keyframed motion. Quite often, a time-based blend is what’s required to properly blend-animate a motion that has more than two distinct keyframed values.
- First and Second Value - These can either be straight numerical values or expressions unto themselves… only used if we’re blending between two values.
- First and Second Frame - Used if we’re blending time, at the lower blend value, the current motion channel will go back to the start frame. As the blend value increases to the upper blend value, the current motion channel will play along to end at the second frame value when the blend value reaches the upper value.
- Track what value for the blend - You can either click on a motion channel to track, or enter an explicit expression by clicking on the Custom Expr button and designing an expression using Dr. Equation Maker from the Expression button.
- Expression - Used when the Custom Expr button is clicked.
- Lower/Upper Blend Trigger Values - Lower and upper values for the blend to track.

If you need to set up a blend for more than one motion slot, you will need to re-select the professor next to each slot that you want set up.

Dr. Dist Maintainer - Multi-slot professor

In order to understand how to use this professor, you need to understand the two modes it works in. One is a linear distance maintainer, the other is a spherical distance maintainer. For the first mode, it’s important that you choose wisely which motion slot you select the Dist Maintainer professor from, this will become the “sliding axis”. For instance, say you want an object to slide up and down the Y axis to try to keep its distance constant to some other object, you would then select Dr. Dist Maintainer next to the Y slot. Note that Dr. Dist Maintainer doesn’t make much
sense in the rotation or scaling slots. In spherical distance mode, the expressions generated will draw a line from the current object to the tracking object and measure the length of that line. It will then move the current object along that line until the distance is the value desired. Here’s a breakdown of the fields:

- Other object - This the object that you want to maintain a constant distance to.
- Distance to maintain - Distance to the maintainer.
- Spherical distance - Throw the professor into spherical distance mode, as described in the preceding paragraph. If this button is off, the distance formula works in sliding-axis, linear mode.

**Dr. Event Maker - Multi-slot professor**

Dr. Event Maker is one of the most powerful professors available in your arsenal. Using this professor, you can set up an animated event to play back when some particular action occurs: the object will play back its own keyframed motion in response. You can also set up a “closing event” for when the particular action undoes itself. A good example (indeed, an example scene), is a set of sliding doors, like on the turbo lift in Star Trek. As an object approaches the doors, they slide open, and then as the object gets far enough away, they slide shut again. But there’s lots more events than that… play around, experiment… see what automated responses you can come up with. Here’s what the fields on the panel do:

- Type of Event - If set to Resettable, the time of the event will reset every time the trigger event becomes false, and start ticking again while the trigger event is true. A One-time-only event will keep ticking even if the condition for the trigger event becomes false.
- Channels for Event: Pick the channels that you want this event to play back on. Note, you are selecting the object’s own keyframed motion for this event. If you want to play another object’s motion back instead, you will need to hand-edit the expressions generated to point to the object in question.
- Object to control the event - This is the object that we’re tracking.
- Look for Event - Here, you choose what event to look for. For now, it simply tracks whether a motion channel becomes greater than “>” or less than “<” a specific value. Notice if you choose “Dist >” or “Dist <”, the From object field becomes active.
- Value - This is the value that our object must hit to trigger the event. Once it falls above or below this value (depending on whether you chose a greater-than or less-than button), the event will be triggered.
- From what object - If you are tracking distance, there needs to be an object to track the distance from.
- Play Event From/To Frame - Pick the first and last frame that our object will play back in response to the event trigger.
- Closing Event - You can set up a closing event that happens when something else occurs. NOTE: if you are choosing to undo some action, like having a door close back shut, it’s important that you set up both events so that they are mutually exclusive… i.e. if you have a door open when an object gets closer than 1 meter, you should not have it close when the object gets out to 0.5 meters… otherwise, there will be a conflict between the two conditions.
- All the other fields are functionally equivalent to the original event fields.

**Dr. Camera Shaker - Multi-slot professor**

This professor is great for not only setting up a camera shake when an event occurs, but also for making stuff jiggle with impact, and other sorts of impact-related events.

- Channels to shake up - Pick what channels you want to set the impact on. Next to each channel selector is a power field... increasing this value will increase the intensity of the impact. Decreasing will do the opposite.
- Object that should shake the camera - This is the object that will make the camera shake occur.
- Look for Event - Look for either a motion channel or the distance of this object from another to either become less or greater than some value.
- Length of shaking - Determines the length of the shaking.
- # Shakes/second - This is the frequency of the shaking, higher values for this will make faster shakes, lower values will make slower shakes.
- Would you like shake to fall off with distance? - You can have the shake fall off with distance.
- Distance when shake is optional - The impact expression will be at full force at this distance; closer, and it will become a super-impact; further, and it will become a soft impact.
- Quadratic Falloff - Make the impact distance expression fall off more rapidly as the object gets further away. A non-quadratic falloff doesn’t decay down as rapidly as the object moves further from the camera.
- Would you like the shake to fall off with impact velocity? - Turning this on will cause the shake to change depending on how hard the object hits the ground.
- Velocity strength factor - Changing this to a higher value will make the shake more sensitive. Changing it lower will make it less sensitive.

**Dr. Morph-o**

Dr. Morph-o is the professor to use for setting up Relativity morphs. It should be noted that Dr. Morph-o currently only sets up morph expressions with either a morph controller or an effector. If you need some other expression to control a morph other than just the X value of a control null, you will need to hand-edit the expression to make it happen.

- Target - This is the target for the morph. NOTE: it doesn’t have to have the same number of points as the base object, but it would be quite advisable.
- Sign of morph - Determines how the morph is applied: replace is like LW’s traditional morph; add will add the values of the target points to the base object; subtract will subtract the points; difference is a cool morph, it morphs only points that are different between the base and target objects.
- **Type of morph** - Normal is like LW’s regular morph. Effector will apply a morph in an effector like fashion, in a spherical gradient around an effector object.
- **Object to control morph** - Choose an object to control the morph.
- **Morph effector** - This is the effector that controls an effector morph.
- **Min radius** - This is the area around the morph effector when everything is fully morphed.
- **Max radius** - In the area between the max and min radius, points close to the max will be barely morphed, while points close to the min will be almost fully morphed. Points outside the max radius are not morphed at all.
- **I want a min and max null** - Choose a min and max null to interactively animate the min and max values for the effector.
- **min null** - The minimum value for the null.
- **max null** - The maximum value for the null.

These last two don’t have to be nulls, but usually are.

**Dr. Target**

Dr. Target is used to set up a targeting relationship between the current object and a target.

**Dr. Matcher**

Dr. Matcher is used to match an unparented, free-floating object to an object or bone deep in a hierarchy. If you want to dynamically parent/unparent an object to another, here’s how to set it up:

1. Create a null and use Dr. Matcher to match to whatever child object/bone/whatever you’re interested in.
2. Set up a motion blend using Dr. Motion Blender between your unparented object and the MATCH’d null.
3. When you want the object to move from unparented to parented, quickly keyframe the blend null over one frame. If you need to have motion blur and not have your object do some sort of odd sub-frame rotation, you might want to change the expressions for the Motion Blend from:

\[
X = \text{BLEND}(X(\text{SELF}, t), 0, X(\text{matcher}, t), 1, X(\text{blah}, t))
\]

To

\[
X = \text{BLEND}(X(\text{SELF}, t), 0, X(\text{matcher}, t), 1, \text{IF}(X(\text{blah}, t) > 0))
\]

That will cause your motion blend to be instantaneous, even on a sub-frame level.

**Dr. MoveVec**

Dr. MoveVec sets up a really odd form of motion... basically, whichever way a MOVEVEC’d object is turned, it will move in that direction. This was originally done as a means of making an object move as if it were under the influence of a joystick... just set up an expression to make it rotate accordingly and the object will act like it’s in a video game.

**Dr. Snake Maker**

Dr. Snake Maker is a generic level plugin that can set up a scene for you with a bone-snake (for path-level deformations) stretching through an object. Here’s a breakdown of the fields:

- **Set up bones for object** - Picks your object.
- **How many bones do you want** - Tell it how many bones in the snake. Keep in mind, you want to balance between having enough bones for a smooth path deformation but not having so many that the scene becomes unwieldy.
- **How long is your object** - Measures your object and tell the nice professor how long it is. As an aside, the “head” of your object should rest at the origin (i.e. XYZ=0,0,0) and the “tail” stretch out into positive Z.
- **Bone name prefix** - Each bone will be named with whatever prefix you choose, followed by a number... i.e. SnakBon01, SnakBon02, etc.
- **Target name prefix** - The way a bone snake is set up in Relativity, a train of nulls follows your object around, and the bones target those nulls. This lets you choose the prefix for each target null.