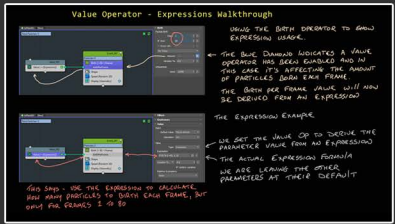
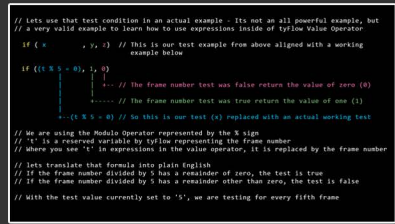
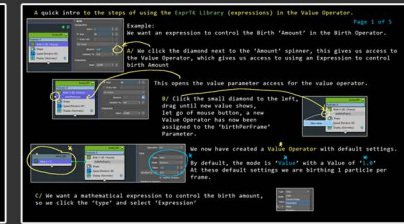
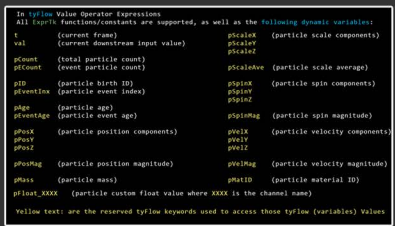






TyFlow - Value Operator Expressions (ExprTK) Reference Guide

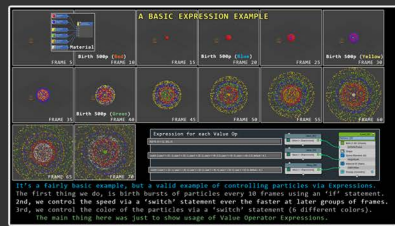
Presented by Robert Andersen aka (FireFlight Photo)

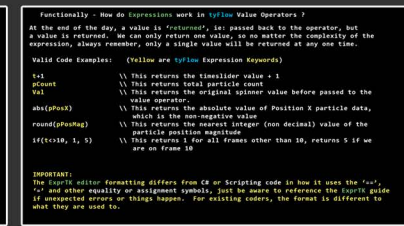


In tyFlow Value Operator Expressions
All ExprTk functions/constants are supported, as well as the following dynamic variables:

t	(current frame)	pScaleX	(particle scale components)
val	(current downstream input value)	pScaleY	
		pScaleZ	
pCount	(total particle count)		
pECount	(event particle count)		
		pScaleAve	(particle scale average)
pID	(particle birth ID)	pSpinX	(particle spin components)
pEventIdx	(particle event index)	pSpinY	
		pSpinZ	
pAge	(particle age)		
pEventAge	(particle event age)	pSpinMag	(particle spin magnitude)
pPosX	(particle position components)	pVelX	(particle velocity components)
pPosY		pVelY	
pPosZ		pVelZ	
pPosMag	(particle position magnitude)	pVelMag	(particle velocity magnitude)
pMass	(particle mass)	pMatID	(particle material ID)
pFloat_XXXX	(particle custom float value where XXXX is the channel name)		

Yellow text: are the reserved tyFlow keywords used to access those tyFlow (variables) Values





Functionally - How do Expressions work in tyFlow Value Operators ?

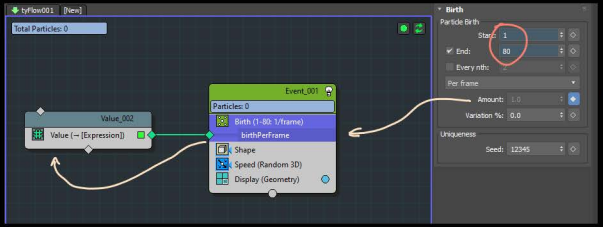
At the end of the day, a value is 'returned', ie: passed back to the operator, but a value is returned. We can only return one value, so no matter the complexity of the expression, always remember, only a single value will be returned at any one time.

Valid Code Examples: (Yellow are tyFlow Expression Keywords)

```
t+1                 \\ This returns the timeslider value + 1
pCount              \\ This returns total particle count
Val                 \\ This returns the original spinner value before passed to the
                       value operator.
abs(pPosX)          \\ This returns the absolute value of Position X particle data,
                       which is the non-negative value
round(pPosMag)      \\ This returns the nearest integer (non decimal) value of the
                       particle position magnitude
if(t<=10, 1, 5)     \\ This returns 1 for all frames other than 10, returns 5 if we
                       are on frame 10
```

IMPORTANT:
The ExprTK editor formatting differs from C# or Scripting code in how it uses the '=', '<' and other equality or assignment symbols, just be aware to reference the ExprTK guide if unexpected errors or things happen. For existing coders, the format is different to what they are used to.

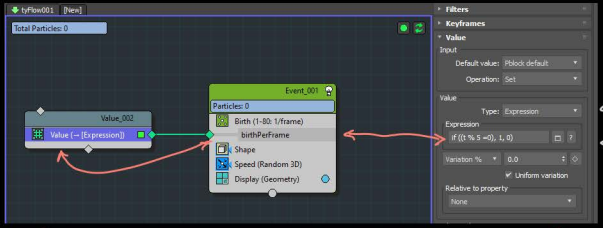
Value Operator - Expressions Walkthrough



USING THE BIRTH OPERATOR TO SHOW EXPRESSION USAGE.

← THE BLUE DIAMOND INDICATES A VALUE OPERATOR HAS BEEN ENABLED AND IN THIS CASE IT'S AFFECTING THE AMOUNT OF PARTICLES BORN EACH FRAME.

THE BIRTH PER FRAME VALUE WILL NOW BE DERIVED FROM AN EXPRESSION



THE EXPRESSION EXAMPLE

← WE SET THE VALUE OP TO DERIVE THE PARAMETER VALUE FROM AN EXPRESSION

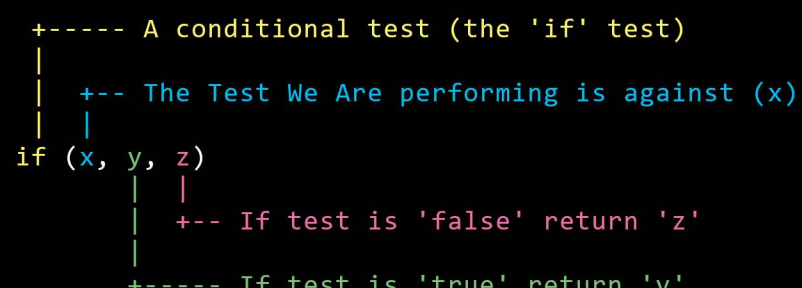
← THE ACTUAL EXPRESSION FORMULA

WE ARE LEAVING THE OTHER PARAMETERS AT THEIR DEFAULT

THIS SAYS - USE THE EXPRESSION TO CALCULATE HOW MANY PARTICLES TO BIRTH EACH FRAME, BUT ONLY FOR FRAMES 1 TO 80

// Value Operator - Understanding the Expression Toolkit

// In the control structures we get this example - which equates to a conditional test



Q: What does ‘return’ mean ?
A: This is the value that will be sent back to the operator, the value can be different based on a true or false test.

```
// Lets use that test condition in an actual example - Its not an all powerful example, but
// a very valid example to learn how to use expressions inside of tyFlow Value Operator

  if ( x         , y, z)  // This is our test example from above aligned with a working
                             example below

  if ((t % 5 = 0), 1, 0)
           |       |  |
           |       |  +-- // The frame number test was false return the value of zero (0)
           |       |
           |       +----- // The frame number test was true return the value of one (1)
           |
           +--(t % 5 = 0) // So this is our test (x) replaced with an actual working test

// We are using the Modulo Operator represented by the % sign
// 't' is a reserved variable by tyFlow representing the frame number
// Where you see 't' in expressions in the value operator, it is replaced by the frame number

// lets translate that formula into plain English
// If the frame number divided by 5 has a remainder of zero, the test is true
// If the frame number divided by 5 has a remainder other than zero, the test is false

// With the test value currently set to '5', we are testing for every fifth frame
```

```
// So when you look at this formula you can denote the following and alter it accordingly
   to get many different results

  if ((t % 5 = 0), 1, 0)

// The Above tests for every fifth frame, change that '5' to something else and get a
   different frame skip ie:

  if ((t % 2 = 0), 1, 0)  // now we are testing every 2nd ('2') frame
                     |  |
                     |  +-- // In this example if the test is false, we return '0' which means
                     |         birth '0' particles.
                     |         Now, this could be changed to some other number, and in that
                     |         way you could get creative with the results
                     |
                     +----  // The 2nd parameter, the 'y' in our original formula is currently
                               set to '1'.
                               meaning when the test is true return '1' which in this example
                               means birth 1 particle.
                               changing that value to ie: '2' or '5' or '10' would now mean if
                               the test was true birth 2,5,10 particles, whichever it was set
                               to.
```

```
if ((t % 2 = 0), 1, 0)
                  |  |
               +--+  +-----+
               |           |
```

// Something important to note here, both the ‘true’ and ‘false’ return conditions don't have to be static values, they can be variables calculated on the fly.

// You just have to be cautious to follow proper formatting rules, unfortunately tyFlow Value Operator, doesn't have a proper debugger to help find formatting issues.

// It only has a ‘valid’ or ‘non-valid’ expression, indicated at run time ie: when you enter the formula.

In tyFlow Value Operator Expressions
All ExprTk functions/constants are supported, as well as the following dynamic variables:

t	(current frame)	pScaleX	(particle scale components)
val	(current downstream input value)	pScaleY	
		pScaleZ	
pCount	(total particle count)		
pECount	(event particle count)	pScaleAve	(particle scale average)
pID	(particle birth ID)	pSpinX	(particle spin components)
pEventInx	(particle event index)	pSpinY	
		pSpinZ	
pAge	(particle age)		
pEventAge	(particle event age)	pSpinMag	(particle spin magnitude)
pPosX	(particle position components)	pVelX	(particle velocity components)
pPosY		pVelY	
pPosZ		pVelZ	
pPosMag	(particle position magnitude)	pVelMag	(particle velocity magnitude)
pMass	(particle mass)	pMatID	(particle material ID)

pFloat_XXXX (particle custom float value where XXXX is the channel name)

Yellow text: are the reserved tyFlow keywords used to access those tyFlow (variables) Values

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **Arithmetic & Assignment Operators**

OPERATOR	DEFINITION
+	Addition between x and y. (eg: x + y)
-	Subtraction between x and y. (eg: x - y)
*	Multiplication between x and y. (eg: x * y)
/	Division between x and y. (eg: x / y)
%	Modulus of x with respect to y. (eg: x % y)
^	x to the power of y. (eg: x ^ y)
:=	Assign the value of x to y. Where y is either a variable or vector type. (eg: y := x)
+=	Increment x by the value of the expression on the right hand side. Where x is either a variable or vector type. (eg: x += abs(y - z))
-=	Decrement x by the value of the expression on the right hand side. Where x is either a variable or vector type. (eg: x[i] -= abs(y + z))
*=	Assign the multiplication of x by the value of the expression on the righthand side to x. Where x is either a variable or vector type. (eg: x *= abs(y / z))
/=	Assign the division of x by the value of the expression on the right-hand side to x. Where x is either a variable or vector type. (eg: x[i + j] /= abs(y * z))
%=	Assign x modulo the value of the expression on the right hand side to x. Where x is either a variable or vector type. (eg: x[2] %= y ^ 2)

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **Equalities & Inequalities**

OPERATOR	DEFINITION
== or =	True only if x is strictly equal to y. (eg: x == y)
<> or !=	True only if x does not equal y. (eg: x <> y or x != y)
<	True only if x is less than y. (eg: x < y)
<=	True only if x is less than or equal to y. (eg: x <= y)
>	True only if x is greater than y. (eg: x > y)
>=	True only if x greater than or equal to y. (eg: x >= y)

These are comparison Operators, in the sense they compare the value on the left to the value on the right. They return 'true' or 'false' which also equates to 1 (true) or 0 (false).

Note:
If we write the following in the expression editor:
x = y; // This does not mean that you are assigning the value in y to x, it means you are testing if x and y are the exact same value, based on that test, return a 'true' or 'false' value.

example: 7 = 5; // this would return 'false'

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **General Purpose Functions** Page 1 of 3 (Some of these will be very useful in the expression editor)

FUNCTION	DEFINITION
abs	Absolute value of x. (eg: abs(x))
avg	Average of all the inputs. (eg: avg(x,y,z,w,u,v) == (x + y + z + w + u + v) / 6)
ceil	Smallest integer that is greater than or equal to x.
clamp	Clamp x in range between r0 and r1, where r0 < r1. (eg: clamp(r0,x,r1))
equal	Equality test between x and y using normalised epsilon
erf	Error function of x. (eg: erf(x))
erfc	Complimentary error function of x. (eg: erfc(x))
exp	e to the power of x. (eg: exp(x))
expm1	e to the power of x minus 1, where x is very small. (eg: expm1(x))
floor	Largest integer that is less than or equal to x. (eg: floor(x))
frac	Fractional portion of x. (eg: frac(x))
hypot	Hypotenuse of x and y (eg: hypot(x,y) = sqrt(x*x + y*y))
iclamp	Inverse-clamp x outside of the range r0 and r1. Where r0 < r1. If x is within the range it will snap to the closest bound. (eg: iclamp(r0,x,r1)
inrange	In-range returns 'true' when x is within the range r0 and r1. Where r0 < r1. (eg: inrange(r0,x,r1)

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **General Purpose Functions** Page 2 of 3 (Some of these will be very useful in the expression editor)

FUNCTION	DEFINITION
log	Natural logarithm of x. (eg: log(x))
log10	Base 10 logarithm of x. (eg: log10(x))
log1p	Natural logarithm of 1 + x, where x is very small. (eg: log1p(x))
log2	Base 2 logarithm of x. (eg: log2(x))
logn	Base N logarithm of x. where n is a positive integer. (eg: logn(x,8))
max	Largest value of all the inputs. (eg: max(x,y,z,w,u,v))
min	Smallest value of all the inputs. (eg: min(x,y,z,w,u))
mul	Product of all the inputs. (eg: mul(x,y,z,w,u,v,t) == (x * y * z * w * u * v * t))
ncdf	Normal cumulative distribution function. (eg: ncdf(x))
not_equal	Not-equal test between x and y using normalised epsilon
pow	x to the power of y. (eg: pow(x,y) == x ^ y)
root	Nth-Root of x. where n is a positive integer. (eg: root(x,3) == x^(1/3))
round	Round x to the nearest integer. (eg: round(x))
roundn	Round x to n decimal places (eg: roundn(x,3)) where n > 0 and is an integer. (eg: roundn(1.2345678,4) == 1.2346)

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **General Purpose Functions** Page 3 of 3 (Some of these will be very useful in the expression editor)

FUNCTION	DEFINITION
sgn	Sign of x, -1 where x < 0, +1 where x > 0, else zero. (eg: sgn(x))
sqrt	Square root of x, where x >= 0. (eg: sqrt(x))
sum	Sum of all the inputs. (eg: sum(x,y,z,w,u,v,t) == (x + y + z + w + u + v + t))
swap <=>	Swap the values of the variables x and y and return the current value of y. (eg: swap(x,y) or x <=> y)
trunc	Integer portion of x. (eg: trunc(x))

Note:
Not all functions and operators in the ExprTK library are usefull in tyFlow Expression Editor (Value Operators).

All are supported and can be used, it's up to the individual to decide.

Example:
Most of the Boolean Operators, probably don't have a big usage due to, they all, generally just return a 'true' or 'false' value . That's not to say they can't be used.

We don't use 'strings' (text) in tyFlow coding, so none of the string section really applies to tyFlow Expressions.

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **Trigonometry Functions** Page 1 of 2

FUNCTION	DEFINITION
acos	Arc cosine of x expressed in radians. Interval [-1,+1] (eg: acos(x))
acosh	Inverse hyperbolic cosine of x expressed in radians. (eg: acosh(x))
asin	Arc sine of x expressed in radians. Interval [-1,+1] (eg: asin(x))
asinh	Inverse hyperbolic sine of x expressed in radians. (eg: asinh(x))
atan	Arc tangent of x expressed in radians. Interval [-1,+1] (eg: atan(x))
atan2	Arc tangent of (x / y) expressed in radians. [-pi,+pi] (eg: atan2(x,y))
atanh	Inverse hyperbolic tangent of x expressed in radians. (eg: atanh(x))
cos	Cosine of x. (eg: cos(x))
cosh	Hyperbolic cosine of x. (eg: cosh(x))
cot	Cotangent of x. (eg: cot(x))
csc	Cosecant of x. (eg: csc(x))
sec	Secant of x. (eg: sec(x))
sin	Sine of x. (eg: sin(x))
sinc	Sine cardinal of x. (eg: sinc(x))

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **Trigonometry Functions** Page 2 of 2

FUNCTION	DEFINITION
sinh	Hyperbolic sine of x. (eg: sinh(x))
tan	Tangent of x. (eg: tan(x))
tanh	Hyperbolic tangent of x. (eg: tanh(x))
deg2rad	Convert x from degrees to radians. (eg: deg2rad(x))
deg2grad	Convert x from degrees to gradians. (eg: deg2grad(x))
rad2deg	Convert x from radians to degrees. (eg: rad2deg(x))
grad2deg	Convert x from gradians to degrees. (eg: grad2deg(x))

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **Control Structures** Page 1 of 2 (These are conditional tests and can be powerful in Expressions)

STRUCTURE	DEFINITION
if	If x is true then return y else return z. eg: 1. if (x, y, z) 2. if (x > y) z; 3. if (x <= 2*y) { z + w };
if-else	The if-else/else-if statement. Subject to the condition branch the statement will return either the value of the consequent or the alternative branch. eg: 1. if (x > y) z; else w; 2. if (x > y) z; else if (w != u) v; 3. if (x < y) { z; w + 1; } else u;

switch	The first true case condition that is encountered will determine the result of the switch. If none of the case conditions hold true, the default action is assumed as the final return value. This is sometimes also known as a multi-way branch mechanism. eg:
	`switch`
	`{`
	`  case x > (y + z) : 2 * x / abs(y - z);`
	`  case x < 3 : sin(x + y);`
	`  default : 1 + x;`
	`}`

while	The structure will repeatedly evaluate the internal statement(s) 'while' the condition is true. The final statement in the final iteration shall be used as the return value of the loop. eg:
	`while ((x -= 1) > 0)`
	`{`
	`  y := x + z;`
	`  w := u + y;`
	`}`

NOTE:
These are multi-line statements, tyFlow has a simple single line editor. We can still do multi-line structures like this, we just have to format them properly. We will show an example of this later.

Lets look at some of the more common operators available in the ExprTK and how to use them.

ExprTK: **Control Structures** Page 2 of 2 (These are conditional tests and can be powerful in Expressions)

STRUCTURE	DEFINITION

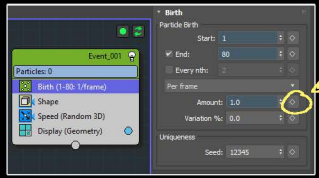
repeat/ until	The structure will repeatedly evaluate the internal statement(s) 'until' the condition is true. The final statement in the final iteration shall be used as the return value of the loop. eg:
	`repeat`
	`  y := x + z;`
	`  w := u + y;`
	`until ((x += 1) > 100)`

for	The structure will repeatedly evaluate the internal statement(s) while the condition is true. On each loop iteration, an 'incrementing' expression is evaluated. The conditional is mandatory whereas the initialiser and incrementing expressions are optional. eg:
	`for (var x := 0; (x < n) and (x != y); x += 1)`
	`{`
	`  y := y + x / 2 - z;`
	`  w := u + y;`
	`}`

NOTE:
There are more control structures than this available in the ExprTK library, but I think these are the main ones we would most likely use in a tyFlow Expression.

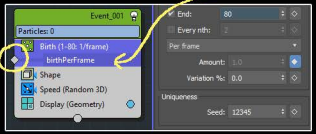
Thats most of the reference guide for ExprTK as it applies to tyFLow, but you can always resource the full documentation for anything left out here

A quick intro to the steps of using the ExprTK Library (expressions) in the Value Operator.



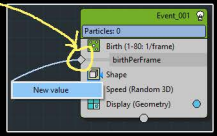
Example:
We want an expression to control the Birth ‘Amount’ in the Birth Operator.

A/ We click the diamond next to the ‘Amount’ spinner, this gives us access to the Value Operator, which gives us access to using an Expression to control birth Amount



This opens the value parameter access for the value operator.

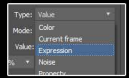
B/ Click the small diamond to the left, drag until new value shows, let go of mouse button, a new Value Operator has now been assigned to the ‘birthPerFrame’ Parameter.



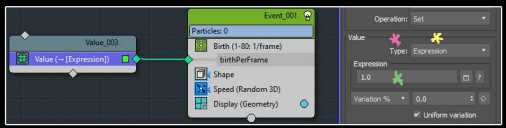
We now have created a Value Operator with default settings.

By default, the mode is ‘Value’ with a Value of ‘1.0’
At these default settings we are birthing 1 particle per frame.

C/ We want a mathematical expression to control the birth amount, so we click the ‘type’ and select ‘Expression’



D/ We can now create an ‘Expression’ to control the ‘birthPerFrame’ (# of birthed particles)



You can see Type, now says ‘Expression’

and you can see a default value of ‘1’ has been entered in the ‘Expression’ Editor

E/ Expression Editor Usage

Clicking the box to the right of the Expression, brings up a larger Editor window.

Clicking the question mark, brings up the Expression Editor help info.

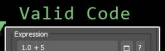
This is really just information on the tyFlow specific variables to access tyFlow parameters via code ie: ‘position’

It also informs you where to get ExprTK Help

And gives an example Expression

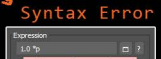
Regardless of which editor window we are using, it is a very basic one line editor with basic debugging.

Valid Code



✓

Syntax Error (problem with Expression)



✘

Clean code (valid) will just show the expression ✓

Bad Code (non valid) will show the error message ✘

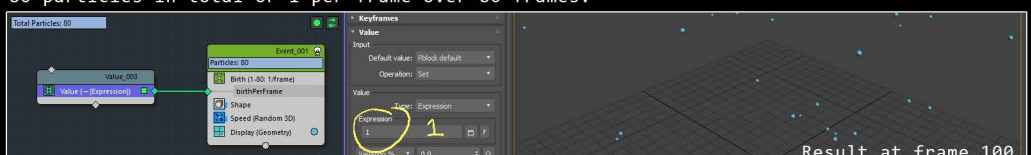
The MaxScript listener really isn’t helpful for debugging

F/ Let's see some example code and understand the single line Expression Editor

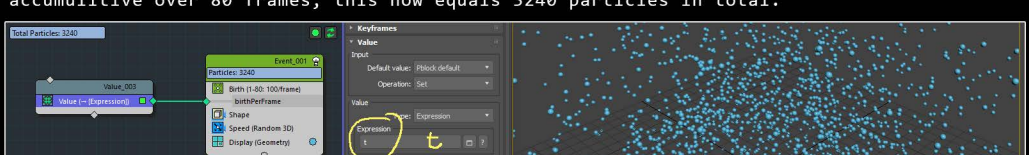
At this point, forget what can be done via standard input box options, we are trying to understand what can be done via 'expressions' and how we create those 'expressions'.

The 'Default' Expression is '1' (in the below example, this means birth 1 particle per frame)

Because we have the birth operator set to birth from frame 1 to frame 80, this expression results in 80 particles in total or 1 per frame over 80 frames.



Now, change the '1' in the expression editor and replace it with the tyFlow reserved variable name 't' which returns the value the time slider is at, on frame 1 that would be 1, frame 2 would be 2 etc. So now on each frame we are birthing an amount of particles equivalent to the frame number, accumulitive over 80 frames, this now equals 3240 particles in total.

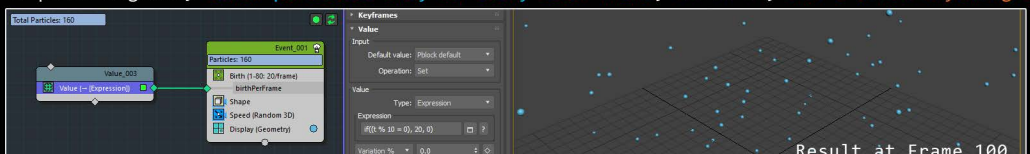


While very basic, we can see we can use expressions to control particle birth and we have also seen an example of accessing tyFlow reserved variables, in this case 't' for time slider count value.

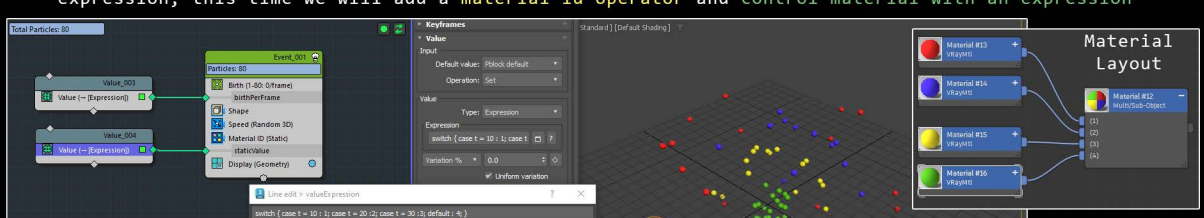
G/ Let's Get more complicated

At the beginning of this help we showed and broke down an 'if' conditional statement, lets adjust it and show what it looks like in max. The expression will be: if((t % 10 = 0, 20, 0)

In plain english, birth particles only on every 10th frame, birth 20, don't birth anything in between.



What we can mainly see here is the particle count matches the math, but unless you saw the timeslider actively moving, we can't tell they were born every tenth frame, lets visually change that with a new expression, this time we will add a material id operator and control material with an expression



The formula for the material ID: switch { case t = 10 : 1; case t = 20 :2; case t = 30 :3; default : 4;}

So if the time slider is 10 (color 1), 20 (color 2), 30 color 3, all others (default) color 4.

H/ Lets explain the ‘switch’ statement which is from the ExprTK library and how to implement multiline code in a single line editor.

```
switch { case t = 10 : 1; case t = 20 :2; case t = 30 :3; default : 4;}
```

If a test matches a case statement, return the value for that case statement, if no case statements match, then return nothing, if a default line is included, that will be the return value when no case statements match.

Lets show the case statement in a more readable multiline format, to visualize and explain further:

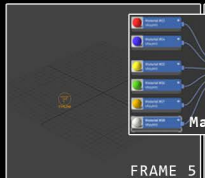
```
Switch
{
  case t = 10 : 1;     This line says, if the frame number is 10 set the material ID to 1
  case t = 20 : 2;     This line says, if the frame number is 20 set the material ID to 2
  case t = 30 : 3;     This line says, if the frame number is 30 set the material ID to 3
  default : 4;         This ‘default’ option in a switch statement, is like a catch all, if no case
}                      statements match assign this default color. In our case all particles birthed
                       after frame 30 will be given a default color of green.
```

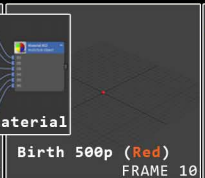
The semi-colon (;) At the end of each line, is how you define a line end in code.

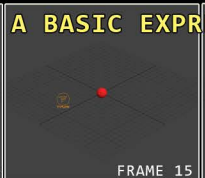
In plain english: ‘t’ is a reserved tyFlow keyword that returns the current frame number, so case t = 10, means if the timeline is at frame 10, return the value after the colon sign (:), In this case that value is 1, return 1.

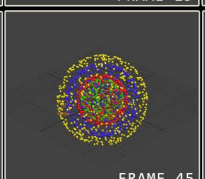
You can see, because of the simplicity of the Expression Editor (single line), code in it can quickly look more complicated than it is. Sometimes you might want to break down complex code in a separate document to understand it better before just coding directly in the Expression Editor

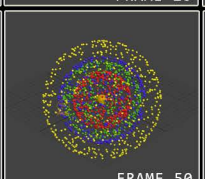
A BASIC EXPRESSION EXAMPLE

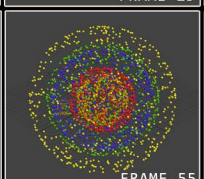


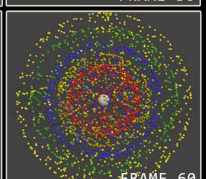


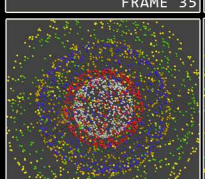


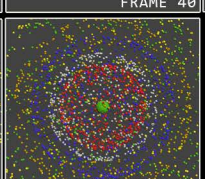






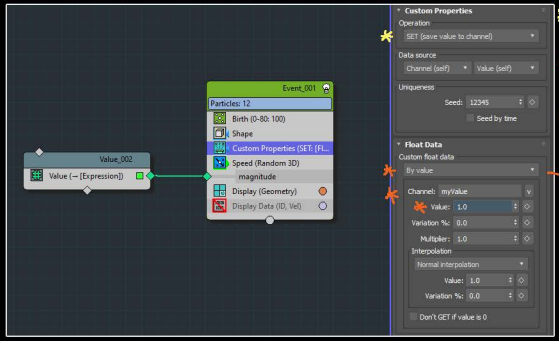


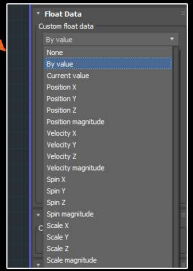


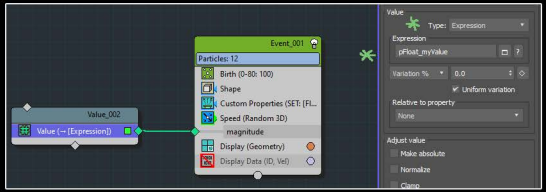


It's a fairly basic example, but a valid example of controlling particles via Expressions.
The first thing we do, is birth bursts of particles every 10 frames using an 'if' statement.
2nd, we control the speed via a 'switch' statement ever the faster at later groups of frames.
3rd, we control the color of the particles via a 'switch' statement (6 different colors).
The main thing here was just to show usage of Value Operator Expressions.

Float Talk - Using the tyFlow Reserved Variable ‘pFloat_XXXX’ To Access Custom Float Data







- When we look at the tyFlow Screen on the left, we see we are saving Custom Float Data to a channel called ‘myValue’.
- We can also see we are just setting a hard coded value of ‘1’

But, it doesn’t have to be by value, you can access the other values from the drop down list.

So, this example shows the way you access custom float data from within an Expression.

We can see to make it work, we replace the XXXX with the name of our channel (myValue)

- We can see in the expression operator, we have it set to expression and it’s pulling the value from a custom float called ‘pFloat_myValue’
- Obviously, this current example is not very dynamic, because we are setting a hard code value of ‘1’

Functionally - How do Expressions work in tyFlow Value Operators ?

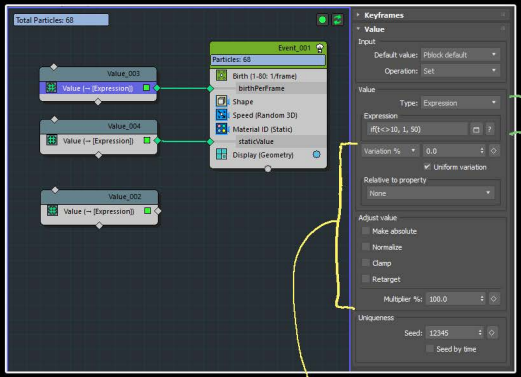
At the end of the day, a value is 'returned', ie: passed back to the operator, but a value is returned. We can only return one value, so no matter the complexity of the expression, always remember, only a single value will be returned at any one time.

Valid Code Examples: (Yellow are tyFlow Expression Keywords)

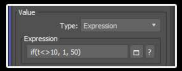
```
t+1                  \\ This returns the timeslider value + 1
pCount               \\ This returns total particle count
Val                  \\ This returns the original spinner value before passed to the
                        value operator.
abs(pPosX)           \\ This returns the absolute value of Position X particle data,
                        which is the non-negative value
round(pPosMag)       \\ This returns the nearest integer (non decimal) value of the
                        particle position magnitude
if(t<>10, 1, 5)      \\ This returns 1 for all frames other than 10, returns 5 if we
                        are on frame 10
```

IMPORTANT:
The ExprTK editor formatting differs from C# or Scripting code in how it uses the '==', '=' and other equality or assignment symbols, just be aware to reference the ExprTK guide if unexpected errors or things happen. For existing coders, the format is different to what they are used to.

We have covered Expression Basics - There is more to it!

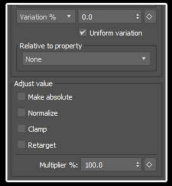


We covered the Expression part of it, which is what this coding tutorial was about. We left all other parameters at default, meaning only the expression return value was used as returned.



The ‘returned’ value can be altered by the settings below the expression editor.

Thats a topic for a different video, but making changes in any of the options eg: variance / relative to / absolute / normalize / clamp / retarget, will alter the Expression value returned.



TyFlow - Value Operator Expressions (ExprTK) Reference Guide

Presented by Robert Andersen aka (FireFlight Photo)

I have a tyFlow Course on Udemy Titled
'TyFlow Basics - The Missing Manual'

If you want to Support me, so I can create more content, that would be the way to do it. It is very affordable and I am constantly adding more content to the course.