Literals, Temps, Locals, Params and Attributes.

An exploration of some of the "data holders" used to store and manipulate numbers.

Literals

A "literal" is a fixed number, for example, 0, 24, -16

As SimAntics is based around 16-bit values, values can range from binary 0000000000000000 to binary 1111111111111111 or hex 0x0000 to hex 0xFFFF.

However, as SimAntics uses signed integer values, these represent the decimal whole numbers +32,767 to -32,768.

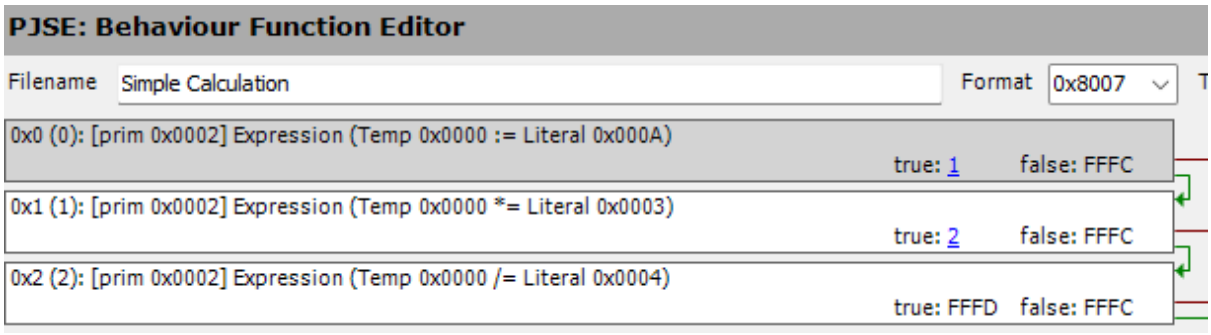
Temps

Literals on their own are not very useful, typically we want to be able to perform maths on numbers, so we need somewhere to store the intermediate calculations and the result.

In higher level languages than SimAntics we can write things like "x = 10 * 3/4" to calculate three-quarters of 10 and put the answer into something we're calling "x", but in SimAntics this needs to be a sequence of operations.

SimAntics provides "data holders" to store things in - what most languages call "variables". One type of data holder is the "temporary values", of which there are eight, commonly referred to as "Temp 0" thru "Temp 7" (or more succinctly as "T0" thru "T7")

So, to perform the above calculation we would need to assign the literal value 10 into "Temp 0" and then manipulate "Temp 0", first multiplying it by 3 then dividing by 4.



Whole Numbers Only Please

So, what's in T0 after the last line is executed?

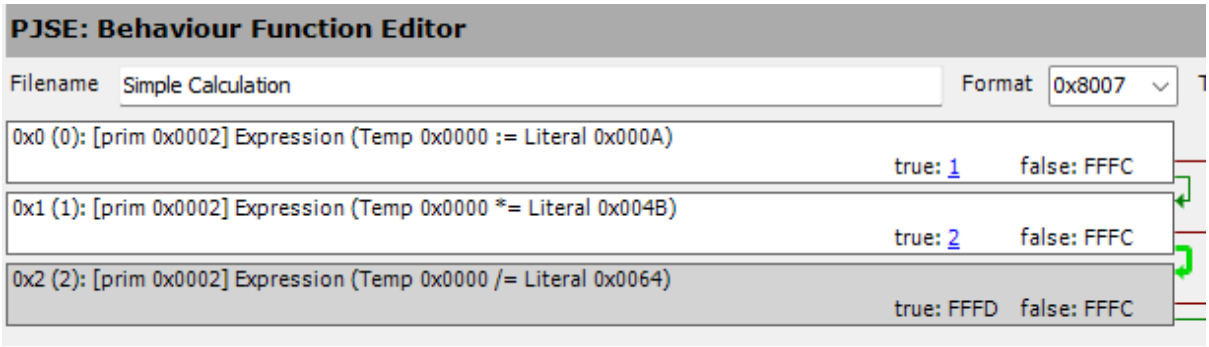
It is NOT "seven and a half" as we're dealing with whole numbers. 10 times 3 is 30, and 4 goes into 30 seven times (remainder two), so the value in T0 is 7

Note, if we perform the calculation the other way around (10 divided by 4 times 3) the value in T0 will be 6

In general, you should always multiply before dividing.

But like all rules, we need to understand the exceptions.

Let's change the code to calculate 75% of a value instead of three-quarters.

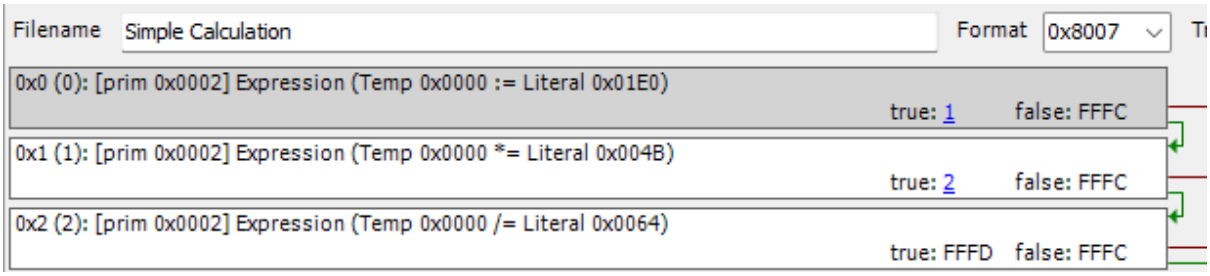


(Aren't hex numbers annoying! 0x0048 is 75 and 0x0064 is 100)

10 times 75 is 750, and 750 divided to 100 is 7

Overflow Issues

So, what's 75% of 480?



With the code above, it is NOT 360 – it will either cause an error or be -295

WTF?

480 times 75 is 36,000 which in hex is 0x8CA0 or 1000110010100000 binary, but that first 1 tells SimAntics that this is the NEGATIVE number -29,536 and that divided by 100 is -295

So perhaps we should always divide first – 480 / 100 * 75 = 300 (only 60 out!)

But then 10 / 100 * 75 = 0

You need to be aware of what range of numbers you are expecting as inputs and adapt your code accordingly.

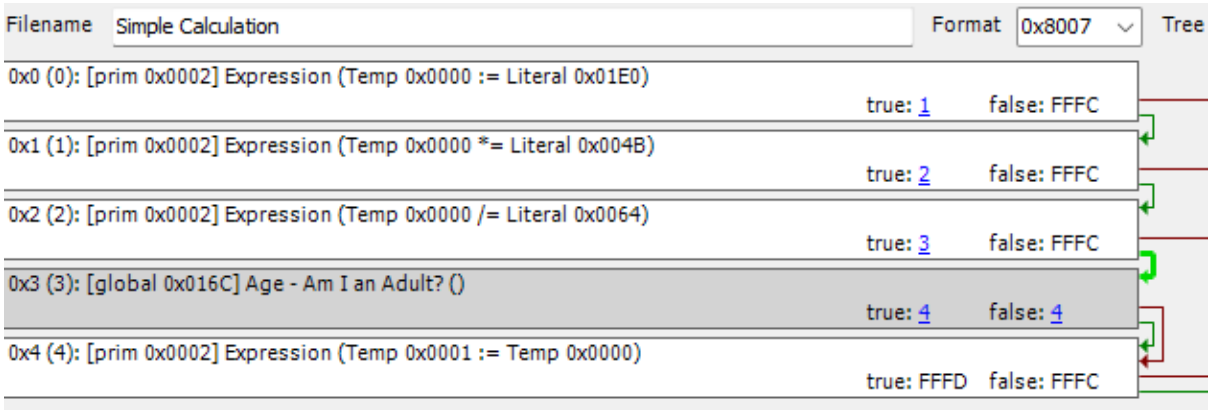
A reasonable way to calculate percentages is "divide by 10, multiply by percent, divide by 10".

480 / 10 * 75 / 10 = 360

10 / 10 * 75 / 10 = 7

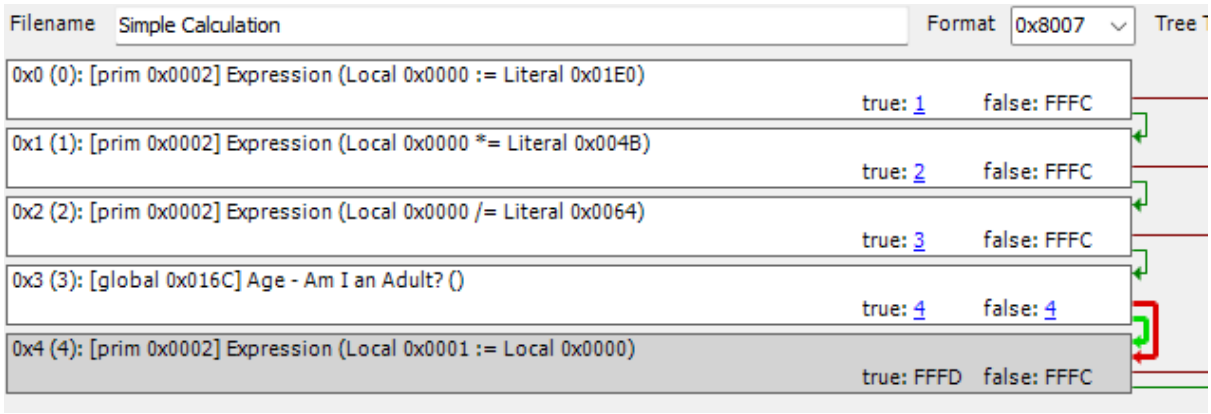
Temps Vs Locals

Temporary data holders are just that – temporary. You should not use them for anything other than "quick and dirty" variables. They are shared by everything in SimAntics, and anything can put a value into them. Some primitives use them to receive values, for example, the animation primitives assume the Sim's "handedness" is indicated by the value in T3. Some primitives return values in them, for example, the Dialog (0x0024) primitive uses a temp to return the notification ID. Most global and semi-global BHAVs stomp all over them!



T1 will NOT be whatever we calculated in T0! (It will be the life-stage code for the Sim, as "Age – Am I an Adult" uses T0 itself).

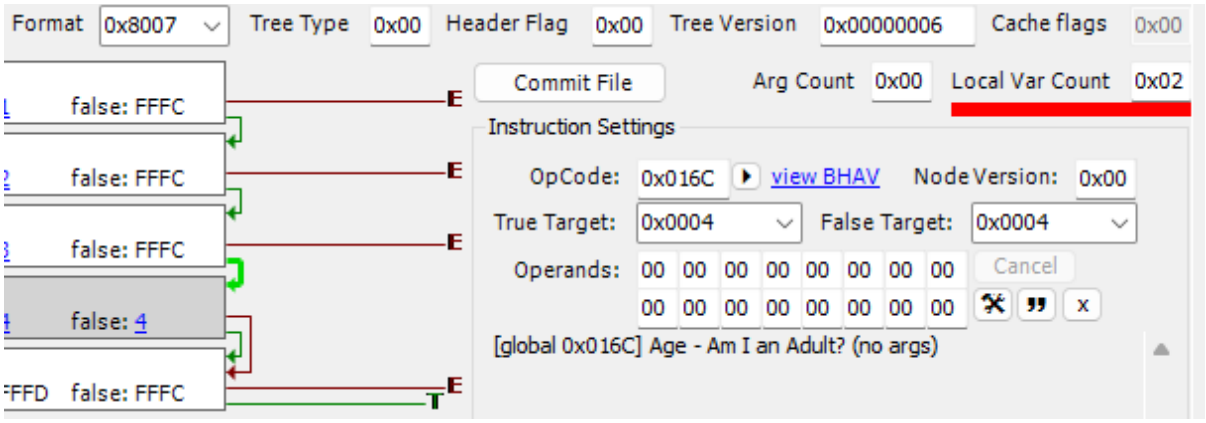
For anything other than very short-term storage, you should use a local data holder. You can have as many locals as you need - I have a BHAV with 27 locals. They are commonly referred to as "Local 0", "Local 1", "Local 2", etc (or more succinctly as "L0", "L1", "L2", etc)



L1 will be whatever we calculated and stored into L0 – even if "Age – Am I an Adult?" used L0 internally instead of T0 to do its own calculations.

Stack Number out of Range 'gotcha'.

The most important thing about locals is we need to tell SimAntics how many we need in the BHAV they are used in, via the "Local Var Count" box.



As locals start at "Local 0", a count of two means we can use L0 and L1, but NOT L2 (or higher).

Failing to set Local Var Count to "one more" than the highest local you use will result in the "Stack Number out of Range" error message.

Why Stack Numbers?

Stack numbers, stack objects, wtf is a "stack"?

OK, stick with me.

We're at our desk about to start some new calculations. We have a scrap piece of paper for our "working outs" and a new ream of paper for our "good" calculations. We place a blank piece of paper in front of us and start calculating … L0 = X, L1 = Y, L2 = L0 + 2 times X (2 times X we do on the scrap), but now we need to find Z – which is a whole different set of calculations. So, we take another blank piece of paper and place it on top of our sheet with our X and Y work on.

We can write L0 and L1 (and L2 and L3 if needed) on this new sheet without changing anything on the sheet underneath, but our "workings out" (temps) are slowly being obscured by our new jottings on the piece of scrap paper.

And now we need to calculate U and V, so we place another new piece of paper on top of our Z work.

And what have we got? A stack of paper!

Having finished our UV working out, we can get back to our Z calculations by throwing away the top-most sheet of paper from our stack – but it's more than throwing it away, we shred it! So, make sure to jot any required UV answers down on the scrap piece (temps) first! (And to keep them safe, copy the temps into locals on our Z sheet.)

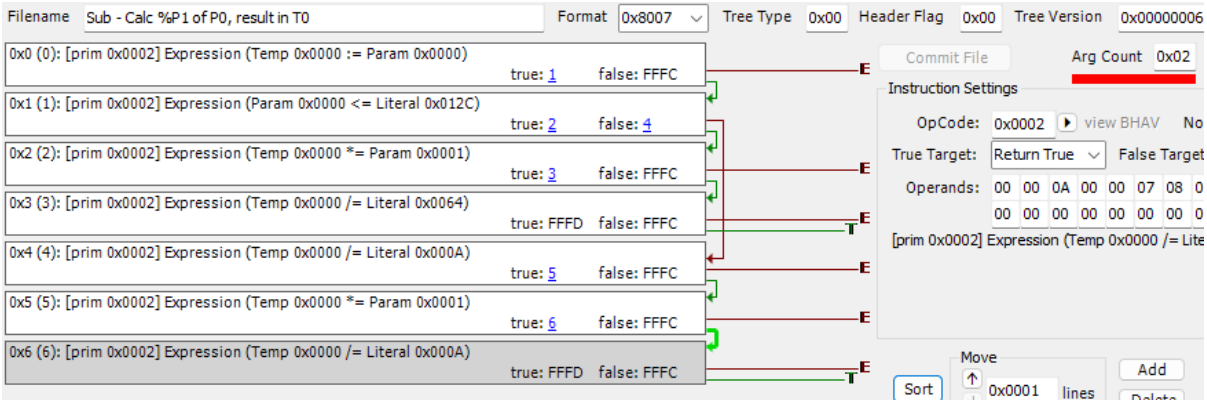
Parameters

Our simple code to calculate 75% of a literal is not very useful – we would need a LOT of BHAVs if we took this approach for any "%X of Y" calculation that we needed. Much better if we could write a BHAV that did just that, "Calculate X% of Y".

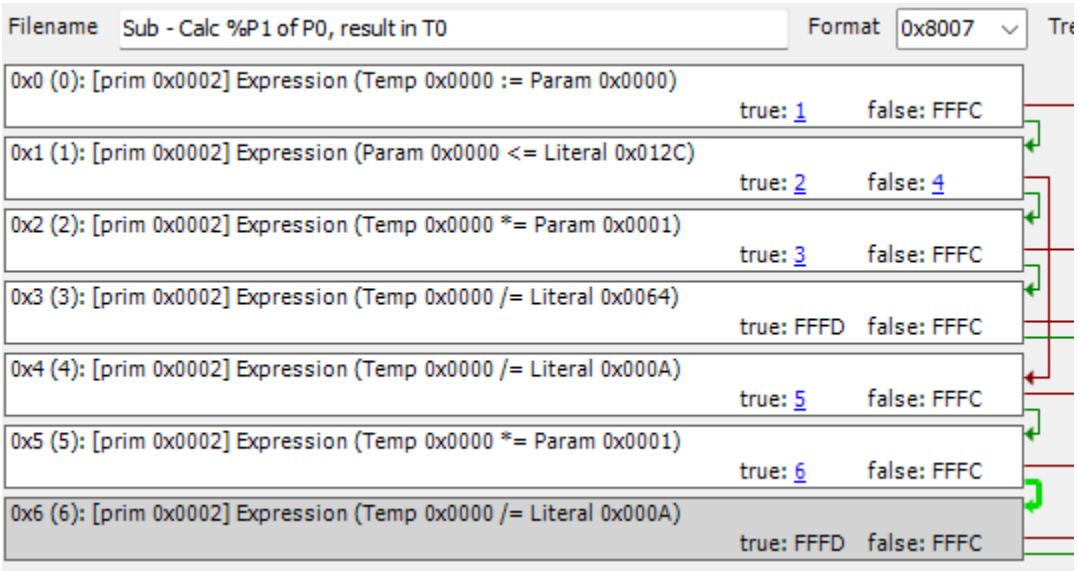
While that seems trivial, remember that such a calculation may vary depending of the “size” of X and Y – if Y is small (less than 300), we can do the “times by X divide by 100” approach, but if it’s larger we need to do the “divide by 10, times by X, divide by 10” approach – and we can encode that knowledge into the BHAV.

Parameters are data holders that pass values to a BHAV from the calling code, that the BHAV can then access. In theory you can have up to 8 parameters, but typically a maximum of 4 are used. Parameters are commonly referred to as "Param 0", “Param 1”, “Param 2”, etc (or more succinctly as "P0", “P1”, “P2”, etc). Should you need to pass more than 4 parameters you can always place the other values into T0 thru T7. If you need more than 12 parameters (P0 thru P3 and T0 thru T7) you should seriously consider restructuring your code!

Like locals, we need to tell SimAntics how many parameters we need in the BHAV they are used in, via the “Arg Count” box. (Parameters are also know as ‘arguments’ or ‘args’.)



Making the code clearer

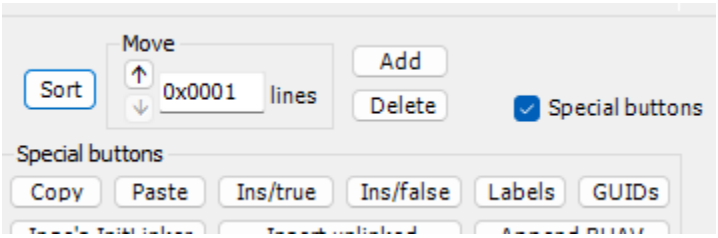


Note that we cannot replace Temp 0 with Local 0, as Local 0 is not on our piece of scrap paper so the calling BHAV could not access the result!

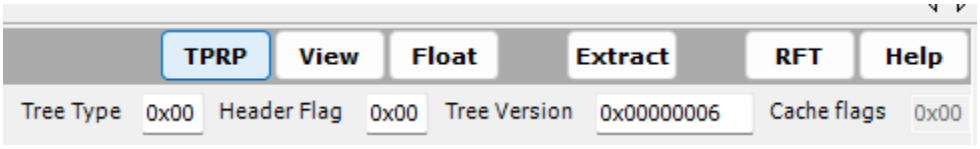
Labels

Parameters and locals can be given labels. And it is good practice to do so. You may be able to remember what is in P2 and what L4 and L5 are used for in the middle of the BHAV today, but in 3 months you will have forgotten!

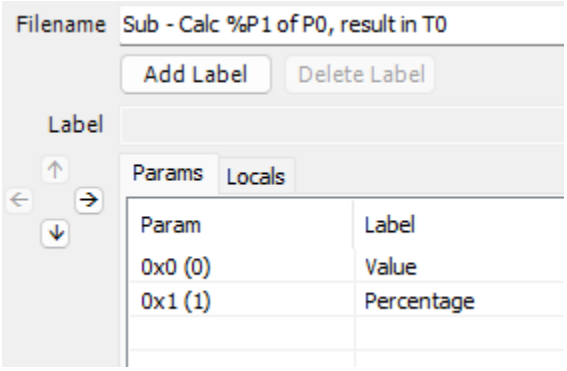
To add labels, make sure the "Special buttons" check box is ticked and click the "Labels" button.



Click "OK" on the "Done!" pop-up, and then click the "TPRP" button.

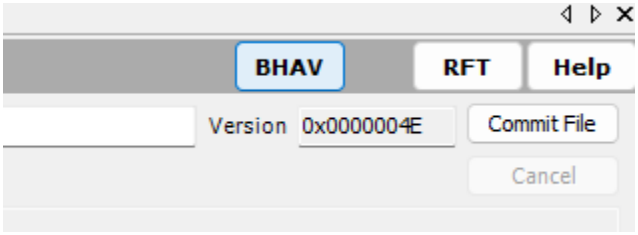


You can now enter meaningful names for the BHAV's parameters and locals.

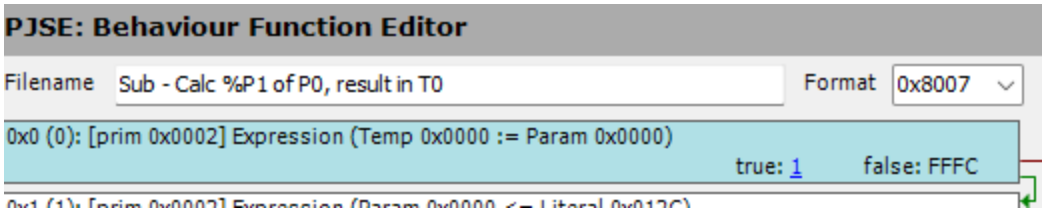


Don't forget to "Commit File".

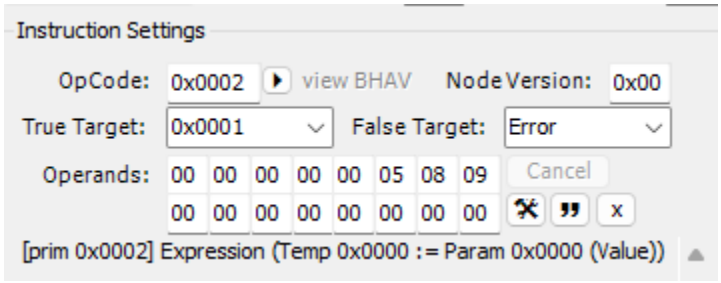
Click the BHAV button to switch back to the code.



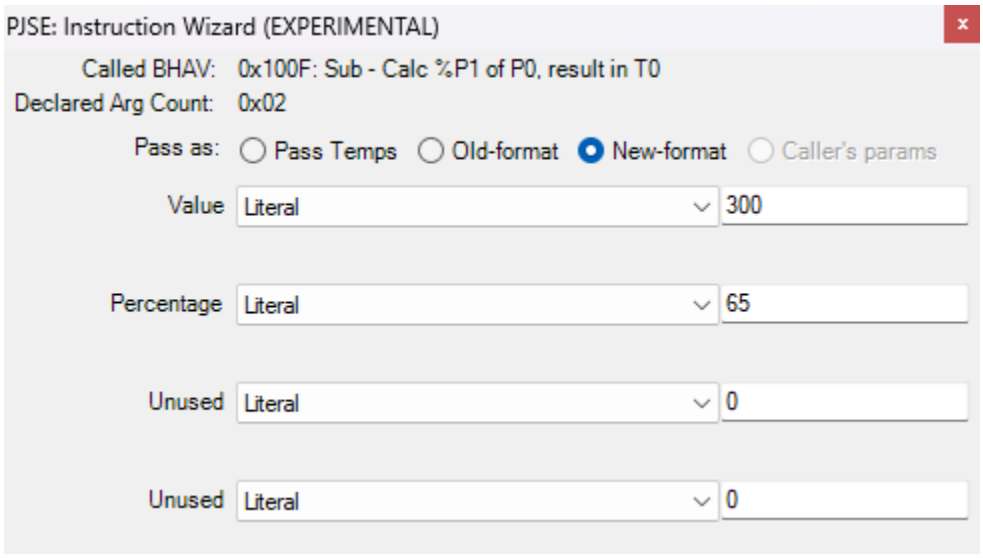
While labels don't show in the main code flow area,



they do when the line is selected in the instruction settings area.



And the parameter labels are also shown when using the wizard to set up the call to the BHAV.



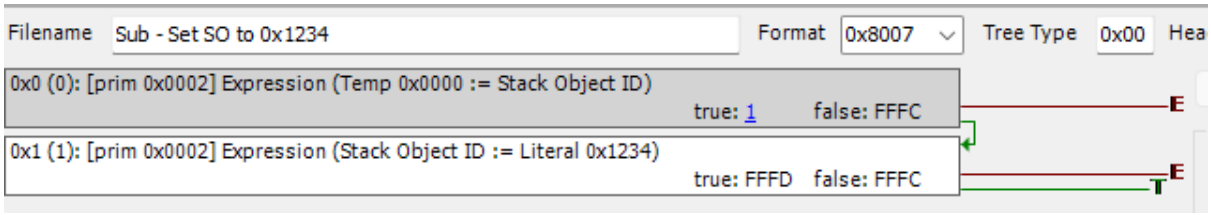
Parameters Revisited

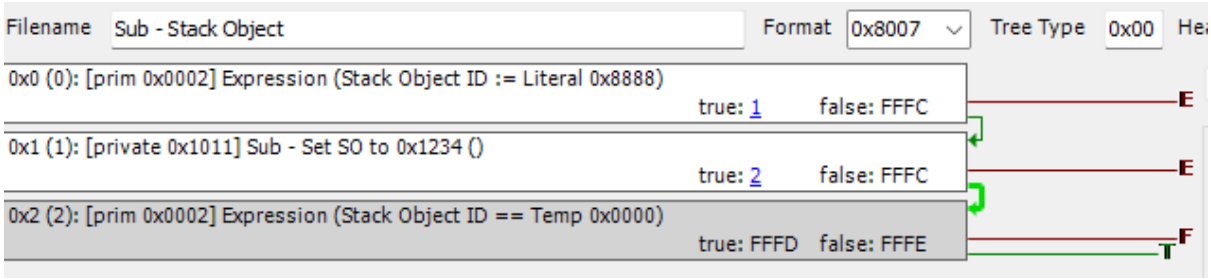
Except in a few very rare and special circumstances, parameters should be treated as read-only within the BHAV. That is, you should never assign a value to a parameter or manipulate one, for example, by adding one to it. You *will* forget you did this, and you *will* assume further on in the code that the parameter has its original value, and you *will* create a bug that is very, very hard to track down! Use a local or a temp – it's only one more line of code!

It is NOT possible to use parameters to return values back to the caller. Other than the true/false return from a BHAV, only temps can return values directly to the calling code.

Stack Object – A Special Parameter

The Stack Object data holder (commonly referred to as the SO for brevity) behaves as a parameter. Any value set into the SO before the call to a BHAV is still in the SO within that BHAV, and like parameters, any changes to the SO are NOT available to the BHAV from which it was called.





"Sub – Stack Object" will return true, as at line 0x2 both T0 and SO will have the value 0x8888

You will come across BHAVs that at line 0x0 loving store the SO into some local and then at the end copy the local back into the SO. This is completely unnecessary.

Object Attributes

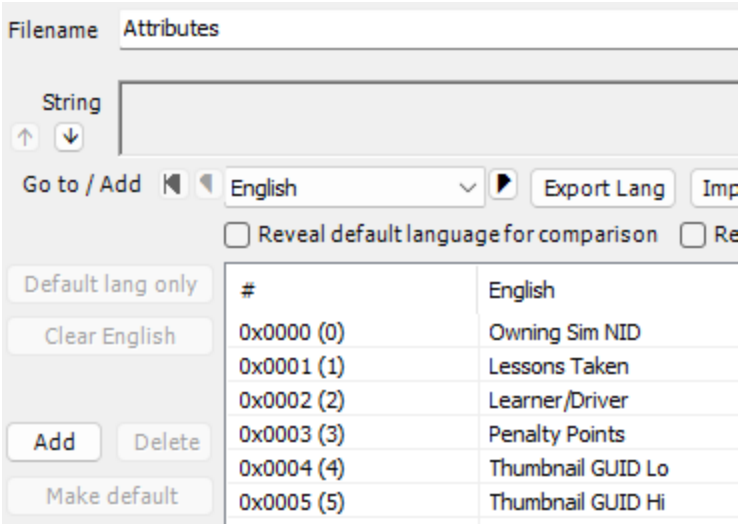
So, if temps can be overwritten at any time by anything, and params and locals only exist within the BHAV that declares them, and the SO is just weird, how do we store data about an object long term?

How do we "remember" who owns the driving licence, if they have passed their test or are still a learner, and how many penalty points have they accrued?

Enter object attributes (and also object semi-attributes – but those are beyond this "simple" introduction).

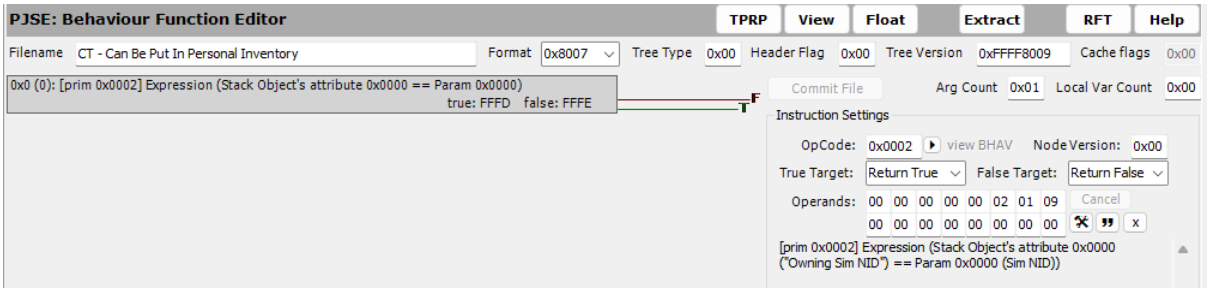
Every object has, by default, eight attributes, but if we need more, we can change the "num attributes" (0x003A) entry in the object's OBJD resource. However, we should be frugal with attributes, don't add "another 20 just in case" as every object instance we create from the OBJD will allocate space for those "just in case" attributes and that consumes memory unnecessarily.

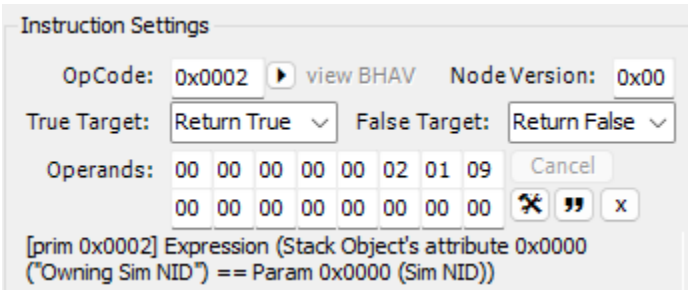
We can (and should) label the attributes by adding a STR# resource with instance 0x0100 and putting the labels for the attributes into the initial strings.



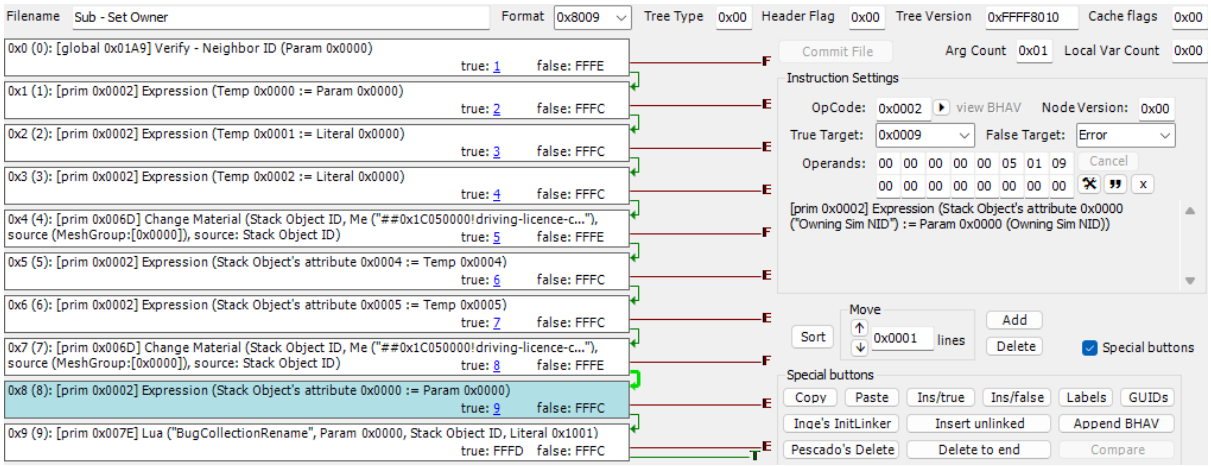
Any BHAV can read or change any of the current Stack Object's attributes. The current Stack Object is whatever object the value in the SO refers to if it assumed to be an object ID (OID).

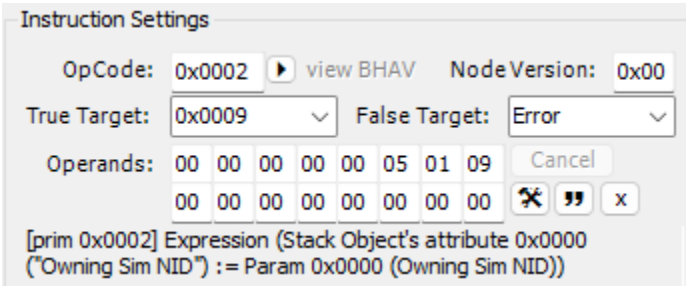
Reading the SO's "Owning Sim NID" attribute in the BHAV "CT – Can Be Put In Personal Inventory"





Updating the SO's "Owning Sim NID" attribute from the BHAV "Sub – Set Owner".


Filename Sub - Set Owner
Format 0x8009
Tree Type 0x00
Header Flag 0x00
Tree Version 0xFFFF8010
Cache flags 0x00
0x0 (0): [global 0x01A9] Verify - Neighbor ID (Param 0x0000)
true: 1 false: FFFE
0x1 (1): [prim 0x0002] Expression (Temp 0x0000 := Param 0x0000)
true: 2 false: FFFC
0x2 (2): [prim 0x0002] Expression (Temp 0x0001 := Literal 0x0000)
true: 3 false: FFFC
0x3 (3): [prim 0x0002] Expression (Temp 0x0002 := Literal 0x0000)
true: 4 false: FFFC
0x4 (4): [prim 0x006D] Change Material (Stack Object ID, Me ("##0x1C050000!driving-licence-c..."), source (MeshGroup:[0x0000]), source: Stack Object ID)
true: 5 false: FFFE
0x5 (5): [prim 0x0002] Expression (Stack Object's attribute 0x0004 := Temp 0x0004)
true: 6 false: FFFC
0x6 (6): [prim 0x0002] Expression (Stack Object's attribute 0x0005 := Temp 0x0005)
true: 7 false: FFFC
0x7 (7): [prim 0x006D] Change Material (Stack Object ID, Me ("##0x1C050000!driving-licence-c..."), source (MeshGroup:[0x0000]), source: Stack Object ID)
true: 8 false: FFFE
0x8 (8): [prim 0x0002] Expression (Stack Object's attribute 0x0000 := Param 0x0000)
true: 9 false: FFFC
0x9 (9): [prim 0x007E] Lua ("BugCollectionRename", Param 0x0000, Stack Object ID, Literal 0x1001)
true: FFFD false: FFFC
Commit File
Arg Count 0x01
Local Var Count 0x00
Instruction Settings
OpCode: 0x0002 view BHAV Node Version: 0x00
True Target: 0x0009 False Target: Error
Operands: 00 00 00 00 00 05 01 09
00 00 00 00 00 00 00 00
Cancel
x
[prim 0x0002] Expression (Stack Object's attribute 0x0000 ("Owning Sim NID") := Param 0x0000 (Owning Sim NID))
Move
Sort
0x0001 lines
Add
Delete
Special buttons
Special buttons
Copy
Paste
Ins/true
Ins/false
Labels
GUIDs
Inge's InitLinker
Insert unlinked
Append BHAV
Pescado's Delete
Delete to end
Compare


Instruction Settings
OpCode: 0x0002 view BHAV Node Version: 0x00
True Target: 0x0009 False Target: Error
Operands: 00 00 00 00 00 05 01 09
00 00 00 00 00 00 00 00
Cancel
x
[prim 0x0002] Expression (Stack Object's attribute 0x0000 ("Owning Sim NID") := Param 0x0000 (Owning Sim NID))

=== END ===