many (but not all) edit boxes and pipeline command parameters allow the use of expressions instead of numerical values. visual3d uses a common expression parser, so the following syntax is common across all command parameters that allow expressions.

evaluate_expression command

the evaluate_expression command allows the users to include expressions for defining the processing of the signals.

evaluate_expression /expression= !/signal_type= !/signal_folder= !/signal_name= /result_type= /result_folder= /result_name= !/apply_as_suffix_to_signal_name=false **;**

expression_items

data tree signals expressions_in_model_builder_mode pipeline_parameters tags c3d_parameters model metrics string data

visual3d reserved characters

there are five characters that cause the equation parser considerable trouble. we have introduced reserved pipeline commands that can be used in the place of these characters.

reserved_characters	
&	ampersand
:	colon
;	semicolon
+	plus
/	forward slash

visual3d reserved names

reserved_names	
current_signal	short hand for signal names
nan	not a number
isnan	is not a number?

numbers

mathematical constants	
pi()	3.14159265358979323846
gravity_vector()	gravity vector in the laboratory coordinate system. for a z-up coordinate system returns (0,0,-9.81)
rand()	generate a random number

mathematical operators

note: visual3d parses the mathematical operators before it parses the signal names. if you have a signal name that contains a mathematical operator (e.g. r-foot1), visual3d will probably not be able to parse the equation expression properly. also note the potential conflict between some of the operators and the reserved characters. if the string is obviously an expression, there is no conflict.

mathematical operators		
+	plus or add	
-	minus or subtract	
*	multiply	
/	divide	
^	power → for example, x	**2 = x to the power 2**
|	logical or → the adjective not is allowed	
&	logical and → the adjective not is allowed	
== or =	equals	
<> or ><	not equals	
<	less than	
<= or =<	less than or equals to	
>	greater than	
>= or =>	greater than or equals to	
not()	not()	

the ugly truth of the logical and

& is used by visual3d for concatenating strings it works quite well, but there is one circumstance where this choice of delimiter is a nuisance. and that is when you want & to actually be a logical and and the parser throws it away in this case you can use the reserved string amp it was only really ugly before version 2024_04 because it didn't work before then example given a file that has tags labelled test1 and test2 to select files containing the tags test1 and test2 select_active_file /file_name=all_files /query=test1 & test2 ! /subject_tags=no_subject **;** ! but what if you want to generalize and use another pipeline parameter for test2 set_pipeline_parameter /parameter_name=scott2 /parameter_value=test1 &::amp &::scott **;** select_active_file /file_name=all_files /query=test1 &::amp &::scott ! /subject_tags=no_subject **;**

brackets

brackets	
( )	contain the parameters in a function
[ ]	specify frame numbers and components

functions

visual3d has pre-defined functions imbedded in the pipeline to help you out. these functions are

commonly used or have been added based on customer use.

metric functions

metric_minimum metric_maximum metric_range metric_mean metric_median metric_stddev metric_rms metric_sum metric_integrate metric_interquartile cross_correlation

signal functions

data exists frame_count add sort transpose interpolate first derivative second derivative resolve_discontinuity indefinite_integral point_relative_to_3points snip spline point_relative_to_3points append_as

string data

specifying string data modifying string data parsing string data string_left string_right string_mid string_find string_reverse_find string_to_lower string_to_upper to_string

least squares fitting of data

best_fit_plane best_fit_circle best_fit_sphere simple_linear_regression

intersection functions

line_line_intersect line_plane_intersect project_point_on_plane point_distance_to_plane point_distance_to_line is_point_inside_polygon

assigned force functions

force center of pressure free moment

trigonometric functions

cosine sine tangent cotangent arctangent cosine arcsine arccos arctan2

array and matrix functions

vector unit_vector list rotation_multiply rotation_inverse rotation_transpose pose_euler2rotation4x4

common math expressions

int(a) - truncate each component to the nearest integer **round(a)** - round each component to the nearest integer **fmod(y,x)** - returns the floating-point remainder of y/x ⇒ y and x must be one dimensional signals; returns a one dimensional signal **abs(a)** - absolute value **sign(a)**= sign of the signal ((a<0)=-1, (a>0)=1, (a==0) = 0 **sqrt(a)** - square root **exp(a)** - exponent **log(a)** - natural log **log10(a)** - log base 10 **length(x)** – creates the length (magnitude) of a vector **distance(a,b)** – distance between two signals. example evaluate_expression /expression= distance(landmark::original::right_hip , target::original::lhip) /result_name=prox_thigh_radius ! /result_type=derived ! /result_folder=processed **;**

dot (a, b) – creates the dot product of a and b **cross (a, b)** – creates the cross product of a and b **note: often you will want to do dot (a, b)/ length (b), or cross (a, b) / length (b)**

expression examples

examples of using evaluate_expressions can be found here: **expressions examples**