The MotionMonitor xGen Software Guide: Operator and Script Function Examples

This document reviews the supported operators and script functions in The MotionMonitor xGen software. Only the operators and script functions described in this document are supported in The MotionMonitor xGen. This document provides a description for each operator and script function as well as an example use for the use of each.

For more information on defining some of the more common biomechanical variables in The MotionMonitor xGen, please refer to the Biomechanical variables and Sample Calculations Knowledge Base article. For more information on using and creating scripts in The MotionMonitor xGen, please refer to the Creating and Using Scripts Knowledge Base article.

Select the type of operator or script function to jump to that section:

Boolean Operators

The following list is of Boolean Operators. Boolean Operators will either use Boolean variable types or return a Boolean output. Booleans can take the value of TRUE, FALSE or INVALID. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". For the following examples, B1 and B2 are Boolean variables, and the Emulator is a hardware device in The MotionMonitor xGen that can generate data in the absence of actual hardware devices.

INVALID

Sets the value for the Boolean variable to INVALID. In the following example, a Boolean variable expression is set to INVALID when the condition is not true. When "Emulator1.Channels.Channel0.Voltage < 0" is not true, the IF statement will return an output of INVALID.

if(Emulator1.Channels.Channel0.Voltage < 0, TRUE, ***INVALID****)*

FALSE

Sets the value for the Boolean variable to FALSE. In the following example, a Boolean expression is set to FALSE when the condition is not true. When "Emulator1.Channels.Channel0.Voltage < 0" is not true, the IF statement will return an output of FALSE.

if(Emulator1.Channels.Channel0.Voltage < 0, TRUE, **FALSE**)

TRUE

Sets the value for the Boolean variable to TRUE. In the following example, a Boolean expression is set to TRUE when the condition is true. When "Emulator1.Channels.Channel0.Voltage < 0" is true, the IF statement will return an output of TRUE.

if(Emulator1.Channels.Channel0.Voltage < 0, ***TRUE****, FALSE)*

B1 == B2

Examines whether the Boolean condition B1 and B2 are equal. In the following example, a Boolean expression is examining whether the Boolean variable B1 is equal to the Boolean variable B2. When the two Boolean variables are equal, the IF statement will return an output of TRUE, and when they are not equal, the IF statement will return an output of FALSE.

*if(****B1 == B2****, TRUE, FALSE)*

B1 != B2

Examines whether the Boolean condition B1 and B2 are not equal. In the following example, a Boolean expression is examining whether the Boolean variable B1 is not equal to the Boolean variable B2. When the two Boolean variables are not equal, the IF statement will return an output of TRUE, and when they are equal, it will return an output of FALSE.

*if(****B1 != B2****, TRUE, FALSE)*

!B

Examines whether the Boolean condition is not true. In the following example, a Boolean expression is examining whether the Boolean variable B1 is not true. If B1 is not true, the IF statement will return an output of 0 and when B1 is true, the IF statement will return an output of 1.

*if(****!B1****, 0, 1)*

B1 || B2

Examines whether the Boolean conditions B1 or B2 are True. In the following example, a Boolean expression is examining whether the Boolean conditions "Emulator1.Channels.Channel0.Voltage < 0" or "Emulator1.Channels.Channel1.Voltage < 0" are true. When either Boolean condition is true, the IF statement will return an output of TRUE. If neither is true, then the IF statement will return an output of FALSE.

*if(**Emulator1.Channels.Channel0.Voltage < 0 || Emulator1.Channels.Channel1.Voltage < 0**, TRUE, FALSE)*

B1 && B2

Examines whether the Boolean conditions B1 and B2 are True. In the following example, a Boolean expression is examining whether the Boolean conditions "Emulator1.Channels.Channel0.Voltage < 0" and "Emulator1.Channels.Channel1.Voltage < 0**"** are true. When both Boolean conditions are true, the IF statement will return an output of TRUE. If only one of the conditions is true or if neither condition is true, then the IF statement will return an output of FALSE.

*if(**Emulator1.Channels.Channel0.Voltage < 0 && Emulator1.Channels.Channel1.Voltage < 0**, TRUE, FALSE)*

if(B, ValueIfTrue, ValueIfFalse)

Constructs an IF statement where the first element, B, is the Boolean condition, the second element, ValueIfTrue, is the Boolean value returned when the condition is true and the third element, ValueIfFalse, is the Boolean value returned when the condition is false. This operator can be used with any variable type as the condition or for the ValueIfTrue and ValueIfFalse outputs. In the following example, the IF statement is examining the status for the condition "Emulator1.Channels.Channel0.Voltage < 0". When the condition is true, the IF statement will return an output of TRUE. When the condition is false, the IF statement will return an output of FALSE.

if(Emulator1.Channels.Channel0.Voltage < 0, TRUE, FALSE)

count(B, BaseTime, Interval)

Counts the number of times the Boolean condition B has transitioned from FALSE to TRUE since BaseTime. Interval is a sampling frequency, in seconds. This operator returns an Integer value. In the following example, the Boolean condition "*Emulator1.Channels.Channel0.Voltage < 0"* is evaluated for how many times it becomes true since the time "InitialTime", which is defined as the first frame of an Activity. The interval that the condition is evaluated at is 0.01 seconds.

count(Emulator1.Channels.Channel0.Voltage < 0, InitialTime, 0.01)

prevtruetime(B, Interval[, NumToSkip])

Returns the previous time at which the Boolean condition B transitioned to a TRUE value. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, the Boolean condition "*Emulator1.Channels.Channel0.Voltage < 0"* is evaluated at an interval of 0.01 seconds for when the condition becomes true. The optional NumToSkip value is set to 1, meaning that the time for the previous true time won't be displayed until the second instance of the Boolean condition becoming true, at which time the time of the first true event will be displayed.

prevtruetime(Emulator1.Channels.Channel0.Voltage < 0, 0.01, 1)

nexttruetime(B, Interval[, NumToSkip])

Returns the next time at which the Boolean condition B transitioned to a TRUE value. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, the Boolean condition "*Emulator1.Channels.Channel0.Voltage < 0"* is evaluated at an interval of 0.01 seconds for when the condition becomes true. There is no NumToSkip value specified.

nexttruetime(Emulator1.Channels.Channel0.Voltage < 0, 0.01)

Integer Operators

The following list is of Integer Operators. Integer Operators will either use Integer variable types or return a Integer output. Integers are whole numbers. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". For the following examples, N1 and N2 are Integer variables.

INVALID

Sets the value for the Integer variable to INVALID. In the following example, an Integer variable, N1, is set to INVALID at the start of a script. There is a semi-colon ";" used since this example is demonstrating how the operator would be used in a script.

*N1 = **INVALID**;*

OK

Sets the value for the Integer variable to OK. In the following example, a Script Integer variable, OKCancelInteger, is evaluated within an IF statement in a script. When OKCancelInteger is equal to OK, the items within the curly brackets "{}" will be evaluated. For instance, this may commonly be used with the OKCancelMessage script function. In the example below, when the result of the message dialog is OK, the items within the curly brackets will be evaluated. If OKCancelInteger is equal to OK, the script will go to the StartRecording Label location in the script.

*if (*OKCancelInteger == ***OK**)*
{goto StartRecording;}

CANCEL

Sets the value for the Integer variable to CANCEL. In the following example, a Script Integer variable, OKCancelInteger, is evaluated within an IF statement in a script. When OKCancelInteger is equal to CANCEL, the items within the curly brackets "{}" will be evaluated. For instance, this may commonly be used with the OKCancelMessage script function. In the example below, when the result of the message dialog is CANCEL, the items within the curly brackets will be evaluated. If OKCancelInteger is equal to CANCEL, the script operator "return;" will be evaluated and the script will be prevented from proceeding.

*if (*OKCancelInteger == ***CANCEL**)*
{return;}

YES

Sets the value for the Integer variable to YES. In the following example, a Script Integer variable, YesNoCancelInteger, is evaluated within an IF statement in a script. When YesNoCancelInteger is equal to YES, the items within the curly brackets "{}" will be evaluated. For instance, this may commonly be used with the YesNoCancelMessage script function. In the example below, when the result of the message dialog is YES, the items within the curly brackets will be evaluated. If YesNoCancelInteger is equal to YES, the script will go to the StartRecording Label location in the script.

*if (YesNoCancelInteger == **YES**)*
{goto StartRecording;}

NO

Sets the value for the integer variable to NO. In the following example, a Script Integer variable, YesNoCancelInteger, is evaluated within an IF statement in a script. When YesNoCancelInteger is equal to NO, the items within the curly brackets "{}" will be evaluated. For instance, this may commonly be used with the YesNoCancelMessage script function. In the example below, when the result of the message dialog is NO, the items within the curly brackets will be evaluated. If YesNoCancelInteger is equal to NO, the script will go to the RetakeMeasurement Label location in the script.

*if (YesNoCancelInteger == **NO**)*
{goto RetakeMeasurement;}

N1 == N2

Examines whether the Integer variables N1 and N2 are equal. In the following example, a Script Integer variable, OKCancelInteger, is evaluated within an IF statement in a script. When OKCancelInteger is equal to OK, the items within the curly brackets "{}" will be evaluated. For instance, this may commonly be used with the OKCancelMessage script function. In the example below, when the result of the message dialog is OK, the items within the curly brackets will be evaluated. If OKCancelInteger is equal to OK, the script will go to the StartRecording Label location in the script.

*if(**OKCancelInteger == OK**)*
{goto StartRecording;}

N1 != N2

Examines whether the Integer variables N1 and N2 are not equal. In the following example, a Script Integer variable, OKCancelInteger, is evaluated within an IF statement in a script. When OKCancelInteger is not equal to OK, the items within the curly brackets "{}" will be evaluated. For instance, this may commonly be used with the OKCancelMessage script function. In the example below, when OKCancelInteger *is not equal to OK*, the items within the curly brackets will be evaluated. If OKCancelInteger is not equal to OK, the script operator "return;" will be evaluated and the script will be prevented from proceeding.

*if(**OKCancelInteger != OK**)*
{return;}

N1 < N2

Examines whether the Integer variable N1 is less than N2. In the following example, the condition N1<2 is evaluated within an IF statement. When N1 is less than 2, the IF statement will return an output of TRUE and when N1 is not less than 2, it will return an output of FALSE.

*if(**N1 < 2**,TRUE, FALSE)*

N1 <= N2

Examines whether the Integer variable N1 is less than or equal to N2. In the following example, the condition N1<=2 is evaluated within an IF statement. When N1 is less than or equal to 2, the IF statement will return an output of TRUE and when N1 is not less than or equal to 2, it will return an output of FALSE.

*if(**N1 <= 2**,TRUE, FALSE)*

N1 > N2

Examines whether the Integer variable N1 is greater than N2. In the following example, the condition N1>2 is evaluated within an IF statement. When N1 is greater than 2, the IF statement will return an output of TRUE and when N1 is not greater than 2, it will return an output of FALSE.

*if(**N1 > 2**,TRUE, FALSE)*

N1 >= N2

Examines whether the Integer variable N1 is greater than or equal to N2. In the following example, the condition N1>=2 is evaluated within an IF statement. When N1 is greater than or equal to 2, the IF statement will return an output of TRUE and when N1 is not greater than or equal to 2, it will return an output of FALSE.

*if(**N1 >= 2**,TRUE, FALSE)*

N1 + N2

Adds the Integer variables N1 and N2. Within a script, it might be desirable to count the number of iterations through a process or the number of recordings that have been performed. In the following example, 1 is added to N1 for each cycle that is completed. There is a semi-colon ";" used since this example is demonstrating how the operator would be used in a script.

*N1=**N1+1**;*

- N

Negates the sign for an Integer, N. Within a script, it may be desirable to change the sign of a variable from positive to negative. In the following example, N1 being set to negative one. Subsequently, when N1 is multiplied by other variables, it would then change the sign for the product. There is a semi-colon ";" used since this example is demonstrating how the operator would be used in a script.

*N1 = **-1**;*

N1 - N2

Subtracts the Integer variables N2 from N1. Within a script, it might be desirable to count the number of iterations through a process or the number of recordings that have been performed. When an iteration needs to be repeated, 1 may need to be subtracted from the current count. In the following example, 1 is subtracted from N1 to repeat the previous cycle. There is a semi-colon ";" used since this example is demonstrating how the operator would be used in a script.

```
if(YesNoCancelInteger == NO)
        {
        CloseActivity();
        N1 = N1 - 1;
        goto NewRecording;
        }
```

N1 * N2

Multiplies Integer variables N1 and N2. Two Integers may be multiplied in order to scale up the value for an integer. In the following example, N1 is multiplied by 10. There is a semi-colon ";" used since this example is demonstrating how the operator would be used in a script.

*N1 = **N1*10;***

N1 / N2

Divides Integer variable N1 by N2. Two Integers may be divided in order to scale down the value for an integer. In the following example, N1 is divided by 10. There is a semi-colon ";" used since this example is demonstrating how the operator would be used in a script.

*N1 = **N1/10;***

abs(N)

Returns the absolute value for an Integer, N. In the following example, the absolute value of N1 is used in a variable expression.

abs(N1)

Switch(N, Case1, Value1, … , CaseN, ValueN, Default)

Controlling Integer variable, N, followed by any number of (case, result) pairs, followed by the default value. Takes an unlimited number of arguments (although the number must be even). This operator is commonly used in place of nested IF clauses because it can be executed more quickly. The following example demonstrates the operator being used in a variable expression as part of the process for assigning a 6DoF rigid body to a body segment through a dialog. The output of the operator in this instance is an Axes variable type. SacrumInteger is a Script Integer variable that receives its value from a script IntegerControl field in a dialog. The integer values are the cases of the Integer scenarios provided through the IntegerControl field and the paired Tx values are Axes variables that have been assigned kinematic tracker axes (i.e. IMU sensor axes definition or Electromagnetic sensor definition). If SacrumInteger is set to 0 or no integer is provided through the script for the IntegerControl field, the switch operator will output a value of INVALD. If the operator is set to 1 through the script, the operator will output a value of T1.

> **switch(SacrumInteger, 0, INVALID, 1, *T1, 2, T2, 3, T3, 4, T4, 5, T5, 6, T6, 7, T7, 8, T8, 9, T9, 10, T10, 11, T11, 12, T12, 13, T13, 14, T14, 15, T15, 16, T16, 17, T17, 18, T18, 19, T19, INVALID)***

Scalar Operators

The following list is of Scalar Operators. Scalar Operators will either use Scalar variable types or return a Scalar output. Scalars are real numbers. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". For the following examples, X1 and X2 are Scalar variables.

INVALID

Sets the value for the Scalar variable to INVALID. In the following example, the condition Forceplate0.Force.Mag < 20 is evaluated within an IF statement in a Scalar variable formula. When the Forceplate0.Force.Mag is less than 20 Newtons, the IF statement will return an output of INVALID, and when Forceplate0.Force.Mag is not less than 20 Newtons, the IF statement will return an output of Forceplate0.Force.Mag.

*if(Forceplate0.Force.Mag < 20, **INVALID**, Forceplate0.Force.Mag)*

X1 == X2

Examines whether the Scalar variables X1 and X2 are equal. In the following example, the condition Forceplate0.Force.Mag == 20 is evaluated within an IF statement in a Scalar variable formula. When the Forceplate0.Force.Mag is equal to 20 Newtons, the IF statement will return an output of 1, and when Forceplate0.Force.Mag is not equal to 20 Newtons, the IF statement will return an output of 0.

*if(**Forceplate0.Force.Mag == 20**, 1, 0)*

X1 != X2

Examines whether the Scalar variables X1 and X2 are not equal. In the following example, the condition Forceplate0.Force.Mag != 20 is evaluated within an IF statement in a Scalar variable formula. When the Forceplate0.Force.Mag is not equal to 20 Newtons, the IF statement will return an output of 1, and when Forceplate0.Force.Mag is equal to 20 Newtons, the IF statement will return an output of 0.

*if(**Forceplate0.Force.Mag != 20**, 1, 0)*

X1 < X2

Examines whether the Scalar variable X1 is less than X2. In the following example, the condition Forceplate0.Force.Mag < 20 is evaluated within an IF statement in a Scalar variable formula. When Forceplate0.Force.Mag is less than 20 Newtons, the IF statement will return an output of 0, and when Forceplate0.Force.Mag is not less than 20 Newtons the IF statement will return an output of Forceplate0.Force.Mag.

*if(**Forceplate0.Force.Mag < 20**, 0, Forceplate0.Force.Mag)*

X1 <= X2

Examines whether the Scalar variable X1 is less than or equal to X2. In the following example, the condition Forceplate0.Force.Mag <= 20 is evaluated within an IF statement in a Scalar variable formula. When Forceplate0.Force.Mag is less than or equal to 20 Newtons, the IF statement will return an output of 0, and when Forceplate0.Force.Mag is not less than or equal to 20 Newtons the IF statement will return an output of Forceplate0.Force.Mag.

*if(**Forceplate0.Force.Mag <= 20**, 0, Forceplate0.Force.Mag)*

X1 > X2

Examines whether the Scalar variable X1 is greater than X2. In the following example, the condition Forceplate0.Force.Mag > 20 is evaluated within an IF statement in a Scalar variable formula. When Forceplate0.Force.Mag is greater than 20 Newtons, the IF statement will return an output of *Forceplate0.Force.Mag,* and when Forceplate0.Force.Mag is not greater than 20 Newtons the IF statement will return an output of 0.

*if(**Forceplate0.Force.Mag > 20**, Forceplate0.Force.Mag, 0)*

X1 >= X2

Examines whether the Scalar variable X1 is greater than or equal to X2. In the following example, the condition Forceplate0.Force.Mag >= 20 is evaluated within an IF statement in a Scalar variable formula. When Forceplate0.Force.Mag is greater than or equal to 20 Newtons, the IF statement will return an output of *Forceplate0.Force.Mag,* and when Forceplate0.Force.Mag is not greater than or equal to 20 Newtons the IF statement will return an output of 0.

*if(**Forceplate0.Force.Mag >= 20**, Forceplate0.Force.Mag, 0)*

X1 + X2

Adds the Scalar variables X1 and X2. In the following example, Forceplate0.Force.Mag and Forceplate2.Force.Mag are added together in a Scalar variable formula.

Forceplate0.Force.Mag + Forceplate2.Force.Mag

- X

Negates the sign for a scalar, X. In the following example, Forceplate0.Force.Z is negated in a Scalar variable formula. This changes the force data from the force applied to the force plate in the Z direction to the Z component of the ground reaction force.

-Forceplate0.Force.Z

X1 – X2

Subtracts the Scalar variable X2 from the Scalar variable X1. In the following example, Forceplate1.Force.Mag is subtracted from Forceplate0.Force.Mag in a Scalar variable formula.

*abs(**Forceplate0.Force.Mag - Forceplate1.Force.Mag**)/ (Forceplate0.Force.Mag + Forceplate1.Force.Mag)*

X1 * X2

Multiplies Scalar variables X1 and X2. In the following example, the force data are multiplied by themselves (i.e. Forceplate0.Force.X* Forceplate0.Force.X) in a Scalar variable formula to manually calculate the magnitude for the force vector.

*(**Forceplate0.Force.X * Forceplate0.Force.X** + **Forceplate0.Force.Y * Forceplate0.Force.Y** + **Forceplate0.Force.Z * Forceplate0.Force.Z**) ^ 0.5*

X1 / X2

Divides the Scalar variable X1 by X2. In the following example the difference from Forceplate0.Force.Mag and Forceplate1.Force.Mag is divided by the sum of Forceplate0.Force.Mag and Forceplate1.Force.Mag in a Scalar variable formula.

*abs(**Forceplate0.Force.Mag - Forceplate1.Force.Mag) / (Forceplate0.Force.Mag + Forceplate1.Force.Mag**)*

X1 % X2

The modulus operator returns the remainder of X1 divided by X2 (Returns a value with the same sign as X1). In the following example, now().Sec modulus 10 is used in a Script variable formula. The resulting output will be a value that goes from 0 to 9 repeatedly, as time progresses.

now().Sec % 10

X1 ^ X2

Raises the Scalar variable X1 to the power of X2. In the following example, the force data are squared (i.e. Forceplate0.Force.X ^ 2) in Scalar variable formula to manually calculate the magnitude for the force vector.

*(**Forceplate0.Force.X ^ 2** + **Forceplate0.Force.Y ^ 2** + **Forceplate0.Force.Z ^ 2**) ^ 0.5*

scalar(N)

Returns the Scalar value for Integer, N. In the following example, the Scalar value for Integer variable N1 is taken in a Scalar variable formula.

scalar(N1)

trunc(X)

Truncates the value for a Scalar variable, X, by removing any decimals, returning an Integer value. In the following example, the truncation operator was applied to Forceplate1.Force.Mag in a Scalar variable formula.

trunc(Forceplate1.Force.Mag)

abs(X)

Returns the absolute value for a Scalar variable, X. In the following example, the absolute value of Forceplate0.Force.Mag minus Forceplate1.Force.Mag is taken in a Scalar variable formula.

abs(Forceplate0.Force.Mag - Forceplate1.Force.Mag) */ (Forceplate0.Force.Mag + Forceplate1.Force.Mag)*

sin(X)

Calculates the trigonometric sine function for a Scalar variable, X. In the following example, the sine of Subject1.Segments.RightShank.Angles.Flexion is taken in a Scalar variable formula.

sin(Subject1.Segments.RightShank.Angles.Flexion)

cos(X)

Calculates the trigonometric cosine function for a scalar variable, X. In the following example, the cosine of Subject1.Segments.RightShank.Angles.Flexion is taken in a Scalar variable formula.

cos(Subject1.Segments.RightShank.Angles.Flexion)

tan(X)

Calculates the trigonometric tangent function for a Scalar variable, X. In the following example, the tangent of Subject1.Segments.RightShank.Angles.Flexion is taken in a Scalar variable formula.

tan(Subject1.Segments.RightShank.Angles.Flexion)

asin(X)

Calculates the inverse trigonometric arcsine function for a Scalar variable, X. In the following example, the arcsine of Subject1.Segments.RightShank.Angles.Flexion is taken in a Scalar variable formula.

asin(Subject1.Segments.RightShank.Angles.Flexion)

acos(X)

Calculates the inverse trigonometric arccosine function for a Scalar variable, X. In the following example, the arccosine of Subject1.Segments.RightShank.Angles.Flexion is taken in a Scalar variable formula.

acos(Subject1.Segments.RightShank.Angles.Flexion)

atan(X)

Calculates the inverse trigonometric arctangent function for a Scalar variable, X. In the following example, the arctangent of Subject1.Segments.RightShank.Angles.Flexion is taken in a Scalar variable formula.

atan(Subject1.Segments.RightShank.Angles.Flexion)

atan2(Y, X)

Calculates the inverse trigonometric arctangent function for 2 Scalar variables, Y and X. The first argument is the opposite and the second the adjacent. In the following example the arctangent of Subject1.Segments.RightFoot.RightAnkle.Pos.Y and Subject1.Segments.RightFoot.RightAnkle.Pos.X is taken in a Scalar variable formula.

atan2(Subject1.Segments.RightFoot.RightAnkle.Pos.Y, Subject1.Segments.RightFoot.RightAnkle.Pos.X)

disp(X, BaseTime)

Calculates the displacement or change in value of a Scalar variable, X, from the time, BaseTime. Each point in time is evaluated relative to the value of X at BaseTime. In the following example, the displacement of the Subject1.Entire.COM.Pos.X is evaluated in a Scalar variable formula relative to the position at time InitialTime, which is the first frame of the Activity.

disp(Subject1.Entire.COM.Pos.X, InitialTime)

diff(X)

Calculates the 1st derivative of a Scalar variable, X. In the following example, the first derivative of the Subject1.Entire.COM.Pos.X is taken in a Scalar variable formula, which would return the velocity for the Subject1.Entire.COM.Pos in the X direction.

diff(Subject1.Entire.COM.Pos.X)

diff2(X)

Calculates the 2nd derivative of a Scalar variable, X. In the following example, the second derivative of the Subject1.Entire.COM.Pos.X is taken in a Scalar variable formula, which would return the acceleration for the Subject1.Entire.COM.Pos in the X direction.

diff2(Subject1.Entire.COM.Pos.X)

spline(X, Interval)

Applies a Spline fit to a Scalar variable, X, without any filtering applied. Interval is a sampling frequency, in seconds. In the following example, a spline fit is applied to the Scalar variable MarkerSet1.heel_l.Pos.X in a Scalar variable formula. The spline fit is being applied to the data at an interval of 0.01 seconds.

spline(MarkerSet1.heel_l.Pos.X, 0.01)

butterworth(X, Interval, Freq)

Applies a 4th order zero phase shift Butterworth filter of a specified frequency to a Scalar variable, X. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. In the following example, a Butterworth filter is applied to the scalar variable Forceplate0.Force.Z in a Scalar variable formula. The interval used to apply the filter to the data is 0.01 seconds and the cut-off frequency is 20Hz.

butterworth(Forceplate0.Force.Z, 0.01, 20)

chebyshev(X, Interval, Freq)

Applies a 4th order zero phase shift Chebyshev filter of a specified frequency to a Scalar variable, X. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. In the following example, a Chebyshev filter is applied to the scalar variable Forceplate0.Force.Z in a Scalar variable formula. The interval used to apply the filter to the data is 0.01 seconds and the cut-off frequency is 20Hz.

chebyshev(Forceplate0.Force.Z, 0.01, 20)

fftlowpass(X, Interval, Freq, Rolloff)

Applies a low-pass filter of a specified frequency to a Scalar variable, X. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. Rolloff is the width of the transition band, in Hz. In the following example, an FFT low-pass filter is applied to the Scalar variable MCCDAQ2.Channel2.Voltage in a Scalar variable formula. The interval used to apply the filter 0.001 seconds, and the cut-off frequency is 400Hz with a roll off of 2Hz.

fftlowpass(MCCDAQ2.Channel2.Voltage, 0.001, 400, 2)

ffthighpass(X, Interval, Freq, Rolloff)

Applies a high-pass filter of a specified frequency to a Scalar variable, X. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. Rolloff is the width of the transition band, in Hz. In the following example, an FFT high-pass filter is applied to the Scalar variable MCCDAQ2.Channel2.Voltage in a Scalar variable formula. The interval used to apply the filter 0.001 seconds, and the cut-off frequency is 20Hz with a roll off of 2Hz.

ffthighpass(MCCDAQ2.Channel2.Voltage, 0.001, 20, 2)

fftnotch(X, Interval, Freq, Width, Rolloff)

Applies a notch filter of a specified frequency and width to a Scalar variable, X. Rolloff is the width of the transition band, in Hz. In the following example, an FFT notch filter is applied to the Scalar variable MCCDAQ2.Channel2.Voltage in a Scalar variable formula. The interval used to apply the filter 0.001 seconds, the frequency to remove from the signal is 60Hz with a width of 5 Hz (+/- 2.5Hz) and a roll off of 2Hz.

fftnotch(MCCDAQ2.Channel2.Voltage, 0.001, 60, 5, 2)

int(X, BaseTime, Interval)

Calculates the integral of a Scalar variable, X, where BaseTime is the starting point for performing the integration and Interval is the sampling interval for performing the calculation. In the following example, the integral of Forceplate0.Force.Z is performed in a Scalar variable formula from the time InitialTime, which is the first frame of the Activity, at an interval of 0.01 seconds.

int(Forceplate0.Force.Z, InitialTime, 0.01)

avg(X, BaseTime, Interval[, EndTime])

Calculates the average value of Scalar variable, X, since BaseTime. Interval is a sampling frequency, in seconds. EndTime is an optional argument that will prevent further computations after the specified time. In the following example, the average of Subject1.Segments.RightShank.Angles.Flexion is performed in a Scalar variable formula, from the time InitialTime, which is the first frame of the Activity, at an interval of 0.01 seconds. The average is evaluated at the time, FinalTime, which is the final frame of the Activity. The output would be a constant value reporting the average at time, FinalTime.

avg(Subject1.Segments.RightShank.Angles.Flexion, InitialTime, 0.01, FinalTime)

movavg(X, Period, Interval[, Time])

Applies a moving-average filter for a Scalar variable X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. Time is an optional argument, which is the time at which to evaluate the function. The Interval is half before and half after.
In the following example, the moving average of Subject1.Segments.RightShank.Angles.Flexion is performed in a Scalar variable formula, with a period of 0.5 seconds and at an interval of 0.01 seconds. The optional Time argument is not used.

movavg(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

rms(X, Period, Interval)

Calculates the RMS for a Scalar variable, X, where Period is the total time period (centered on the current time) and Interval is the sampling interval for performing the calculation. In the following example, the RMS of MCCDAQ2.Channel2.Voltage is performed in a Scalar variable formula, with a period of 0.1 seconds and an at an interval of 0.01 seconds.

rms(MCCDAQ2.Channel2.Voltage, 0.1, 0.001)

min(X, StartTime, StopTime, Interval)

Identifies the minimum value for a Scalar variable, X, occurring during a time period defined by StartTime and StopTime and Interval that is the step size (in seconds) for evaluating the time period. In the following example, the minimum of Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula, between the time periods of InitialTime and FinalTime, which are the start and end of the Activity respectively, at an interval of 0.01 seconds.

min(Subject1.Segments.RightShank.Angles.Flexion, InitialTime, FinalTime, 0.01)

max(X, StartTime, StopTime, Interval)

Identifies the maximum value for a Scalar variable, X, occurring during a time period defined by StartTime and StopTime and Interval that is the step size (in seconds) for evaluating your time period. In the following example, the maximum of Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula, between the time periods of InitialTime and FinalTime, which are the start and end of the Activity, at an interval of 0.01 seconds.

max(Subject1.Segments.RightShank.Angles.Flexion, InitialTime, FinalTime, 0.01)

tmin(X, StartTime, StopTime, Interval)

Identifies the time when the minimum value for a Scalar variable, X, occurred over a time period defined by StartTime and StopTime and Interval that is the step size (in seconds) for evaluating your time period. In the following example, time at which the minimum occurred for Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Time variable formula, between the time periods of InitialTime and FinalTime, which are the start and end of the Activity, at an interval of 0.01 seconds.

tmin(Subject1.Segments.RightShank.Angles.Flexion, InitialTime, FinalTime, 0.01)

tmax(X, StartTime, StopTime, Interval)

Identifies the time when the maximum value for a Scalar variable, X, occurred over a time period defined by StartTime and StopTime and Interval that is the step size (in seconds) for evaluating your time period. In the following example, time at which the maximum occurred for Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Time variable formula, between the time periods of InitialTime and FinalTime, which are the start and end of the Activity, at an interval of 0.01 seconds.

tmax(Subject1.Segments.RightShank.Angles.Flexion, InitialTime, FinalTime, 0.01)

islocalmin(X, Period, Interval)

Determines if the current value of a Scalar variable X is a local minimum within the specified period, as determined by sampling readings at the specified interval. Period and Interval are both times, in seconds and a Boolean value is returned. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Boolean variable formula for a local minimum within a moving 0.5 second window and is evaluated at an interval of 0.01 seconds.

islocalmin(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

islocalmax(X, Period, Interval)

Determines if the current value of a Scalar Variable X is a local max within the specified period, as determined by sampling readings at the specified interval. Period and Interval are both times, in seconds. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Boolean variable formula for a local maximum within a moving 0.5 second window and is evaluated at an interval of 0.01 seconds.

islocalmax(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

prevmintime(X, Period, Interval[, NumToSkip])

Returns the previous time at which a Scalar variable X was at a local minimum. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Time variable formula and the previous minimum time is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. The optional NumToSkip value is set to 1, meaning that the time for the previous true time won't be displayed until the second instance of a minimum, at which time the time of the first true event will be displayed.

prevmintime(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01, 1)

nextmintime(X, Period, Interval[, NumToSkip])

Returns the next time at which a Scalar variable X was at a local minimum. Interval is a time, in seconds. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Time variable formula and the next minimum time is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. There is no NumToSkip value specified.

nextmintime(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

prevmaxtime(X, Period, Interval[, NumToSkip])

Returns the previous time at which a Scalar variable X was at a local maximum. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Time variable formula and the previous maximum time is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. There is no NumToSkip value specified.

prevmaxtime(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

nextmaxtime(X, Period, Interval[, NumToSkip])

Returns the next time at which a Scalar variable X was at a local maximum. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Time variable formula and the next maximum time is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. There is no NumToSkip value specified.

nextmaxtime(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

prevminvalue(X, Period, Interval[, NumToSkip])

Returns the previous local-minimum value of a Scalar variable X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula and the previous minimum is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. The optional NumToSkip value is set to 1, meaning that the value for the previous minimum won't be displayed until the second instance of a minimum occurs, at which time the value of the first minimum will be displayed.

prevminvalue(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01, 1)

nextminvalue(X, Period, Interval[, NumToSkip])

Returns the next local-minimum value of a Scalar variable X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula and the next minimum is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. There is no NumToSkip value specified.

nextminvalue(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

prevmaxvalue(X, Period, Interval[, NumToSkip])

Returns the previous local-maximum value of a Scalar varaible X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula and the previous maximum is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. There is no NumToSkip value specified.

prevmaxvalue(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

nextmaxvalue (X, Period, Interval[, NumToSkip])

Returns the next local-maximum value of a Scalar variable X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument. In the following example, Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula and the next maximum is reported for a moving 0.5 second window and is evaluated at an interval of 0.01 seconds. There is no NumToSkip value specified.

nextmaxvalue(Subject1.Segments.RightShank.Angles.Flexion, 0.5, 0.01)

maxfrom(X, BaseTime, Interval)

Returns the maximum value of a Scalar variable X starting from the time, BaseTime. Interval is a sampling frequency, in seconds. The output will update as new maximums are encountered. In the following example, the maximum of Subject1.Segments.RightShank.Angles.Flexion is evaluated in a Scalar variable formula from the time, InitialTime plus 1, which is one second after the start of the Activity, and is evaluated at an interval of 0.01 seconds.

maxfrom(Subject1.Segments.RightShank.Angles.Flexion, InitialTime+1, 0.01)

meanfreq(X, StartTime, StopTime, Interval[, Detrend])

Returns the mean frequency of a Scalar variable X over the time period defined from StartTime to StopTime. Interval is a sampling frequency, in seconds. Detrend is an optional argument that detrends the data for X and defaults to FALSE. In the following example, the mean frequency of MCCDAQ2.Channel2.Voltage is evaluated in a Scalar variable formula from InitialTime to FinalTime, which are the start and end of the Activity respectively, at an interval of 0.01 seconds. The optional argument to detrend the data was set to TRUE.

meanfreq(MCCDAQ2.Channel2.Voltage, InitialTime, FinalTime, 0.001, TRUE)

medianfreq(X, StartTime, StopTime, Interval[, Detrend])

Returns the median frequency of a Scalar variable X over the time period defined from StartTime to StopTime. Interval is a sampling frequency, in seconds. Detrend is an optional argument that detrends the data for X and defaults to FALSE. In the following example, the median frequency of MCCDAQ2.Channel2.Voltage is evaluated in a Scalar variable formula from InitialTime to FinalTime, which are the start and end of the Activity respectively, at an interval of 0.01 seconds. The optional argument to detrend was not included, so it is set to FALSE.

medianfreq(MCCDAQ2.Channel2.Voltage, InitialTime, FinalTime, 0.001)

sampen(X, StartTime, StopTime, Interval, N, M, R)

Performs a calculation of Sample Entropy for a Scalar variable X time series. N is the number of samples to average (Data length N is typically between 1 and 20), M is the pattern length (Embedding dimension M is typically between 2 and 3), and R is the pattern-matching tolerance (Tolerance R is typically between 0.1 and 0.2), given as a fraction of the standard deviation of the data points. The Interval argument should typically be equal to the measurement rate interval, otherwise you'll be analyzing a signal that is very different from the one that was recorded. In the following example, the sample entropy for Forceplate0.COP.X is evaluated in a Scalar variable formula from InitialTime to FinalTime, which are the start and end of the Activity respectively, at an interval of 0.001 seconds. The Data length is set to 6, Embedding dimension is set to 3, and Tolerance is set to 0.15.

sampen(Forceplate0.COP.X, InitialTime, FinalTime, 0.001, 6, 3, 0.15)

higlen(X, StartTime, StopTime, Interval, K)

Returns the Higuchi length association with decimation factor K for a Scalar variable X time series. This can be used to manually determine Kmax. In the following example, the Higuchi length for Forceplate0.COP.X is evaluated in a Scalar variable formula from InitialTime to FinalTime, which are the start and end of the Activity respectively, at an interval of 0.001 seconds and decimation factor of 4.

higlen(Forceplate0.COP.X, InitialTime, FinalTime, .001, 4)

higdim(X, StartTime, StopTime, Interval, Kmax[, Tol])

Calculates the fractal dimension of a data series for the Scalar variable X using Higuchi's method. All the lengths up through Kmax (upper bound) will be recalculated, and then returns a fractal dimension. Tol is an optional argument for a tolerance value, which allows the algorithm to automatically pick a value for Kmax, rather than determining it manually. Tolerance is expressed as a ratio of the standard deviation of the data points. In the following example, the fractal dimension is calculated using Higuchi's method for Forceplate0.COP.X in a Scalar variable formula from InitialTime to FinalTime, which are the start and end of the Activity respectively, at an interval of 0.001 second. The Kmax is set to 30 and optional Tolerance was set to 1.5.

higdim(Forceplate0.COP.X, InitialTime, FinalTime, 0.001, 30, 1.5)

katzdim(X, StartTime, StopTime, Interval)

Calculates the fractal dimension, based on Katz's method. In the following example, the fractal dimension is calculated using Katz's method for Forceplate0.COP.X in a Scalar variable formula from InitialTime to FinalTime, which are the start and end of the Activity respectively, at an interval of 0.001 second.

katzdim(Forceplate0.COP.X, InitialTime, FinalTime, .001)

Vector Operators

The following list is of Vector Operators. Vector Operators will either use Vector variable types or return a Vector output. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". For the following examples, V1 and V2 are Vector variables.

INVALID

Sets the value for the Vector variable to INVALID. In the following example, the condition Forceplate0.Force.Mag < 20 is evaluated within an IF statement in a Vector variable formula. When the Forceplate0.Force.Mag is less than 20 Newtons, the IF statement will return an output of INVALID, and when Forceplate0.Force.Force.Mag is not less than 20 Newtons, the IF statement will return an output of Forceplate0.Force.

*if(Forceplate0.Force.Mag<20, **INVALID**, Forceplate0.Force)*

vec(X, Y, Z)

Composes a Vector, where X, Y, and Z are Scalar values. In the following example, a Vector is constructed within a Vector variable formula, where the X component is Subject1.Entire.COM.Pos.X, the Y component is Subject1.Entire.COM.Pos.Y, and the Z component is 0.

vec(Subject1.Entire.COM.Pos.X, Subject1.Entire.COM.Pos.Y, 0)

V1 == V2

Examines whether the Vector variables V1 and V2 are equal. In the following example, the condition Forceplate0.Force == Forceplate1.Force is evaluated within an IF statement in a Scalar variable formula. When Forceplate0.Force is equal to Forceplate1.Force, the IF statement will return an output of 1 and when Forceplate0.Force is not equal to Forceplate1.Force, the IF statement will return an output of 0.

*if(**Forceplate0.Force == Forceplate1.Force**, 1, 0)*

V1 != V2

Examines whether the Vector variables V1 and V2 are not equal. In the following example, the condition Forceplate0.Force != Forceplate1.Force is evaluated within an IF statement in a Scalar variable formula. When Forceplate0.Force is not equal to Forceplate1.Force, the IF statement will return an output of 1, and when Forceplate0.Force is equal to Forceplate1.Force, the IF statement will return an output of 0.

*if(**Forceplate0.Force != Forceplate1.Force**, 1, 0)*

V1 + V2

Adds the Vector variables V1 and V2. In the following example, Forceplate0.Force and Forceplate1.Force are added together in a Vector variable formula.

Forceplate0.Force + Forceplate1.Force

-V

Negates the sign for a Vector, V. In the following example, Forceplate0.Force is negated in a Vector variable formula. This changes the force data from the force applied to the force plate to the ground reaction force.

-Forceplate0.Force

V1 – V2

Subtracts the Vector variable V2 from the Vector variable V1. In the following example, Forceplate1.Force is subtracted from Forceplate0.Force in a Vector variable formula.

Forceplate0.Force - Forceplate1.Force

V * Scale

Multiplies a Vector variable, V, by a scaling factor Scale. In the following example, the force plate vector is multiplied by a scale factor of 2 in a Vector variable formula.

Forceplate0.Force*2

V / Scale

Divides a Vector variable, V, by a scaling factor Scale. In the following example, the force plate vector is divided by Subject1.Mass in a Vector variable formula

Forceplate0.Force/Subject1.Mass

mag (V)

Returns the magnitude of a Vector variable, V, which is a Scalar value. In the following example, the magnitude of the Vector Forceplate0.Force is taken in a Scalar variable formula.

mag(Forceplate0.Force)

unit(V)

Returns the Unit Vector for a Vector variable, V. In the following example, the unit vector for Forceplate0.Force is returned in a Vector variable formula.

unit(Forceplate0.Force)

dot(V1, V2)

Performs the dot product or scalar product of Vectors V1 and V2 and returns a scalar value. In the following example, the dot product of the moment vector and angular velocity vectors is computed in a Scalar variable formula to calculate the joint power.

dot(reldir(Subject1.Segments.RightFoot.RightAnkle.Moment,
Subject1.Segments.RightShank.AxisSystems.Anatomical.Axes),
angvel(rel(Subject1.Segments.RightFoot.AxisSystems.Anatomical.Axes,
Subject1.Segments.RightShank.AxisSystems.Anatomical.Axes).Ori))*0.01745329*

cross(V1, V2)

Performs the cross product or vector product of Vectors V1 and V2. In the following example, the cross product of RightFootDistJointArm, a Vector variable for a moment arm, and RightFootDistForce, a Vector variable for force, is computed in a Vector variable formula to calculate a moment or torque.

cross(RightFootDistJointArm, RightFootDistForce)

disp(V, BaseTime)

Calculates the displacement or change in value of a Vector variable, V, from the time, BaseTime. Each point in time is evaluated relative to the value of V at BaseTime. In the following example, the displacement of the Subject1.Entire.COM.Pos is evaluated in a Vector variable formula relative to the position at time InitialTime, which is the first frame of the Activity.

disp(Subject1.Entire.COM.Pos, InitialTime)

diff(V)

Calculates the 1st derivative of a Vector variable, V. In the following example, the first derivative of Subject1.Entire.COM.Pos is taken in a Vector variable formula, which would return the velocity Vector for the Subject1.Entire.COM.Pos.

diff(Subject1.Entire.COM.Pos)

diff2(V)

Calculates the 2nd derivate of a Vector variable, V. In the following example, the second derivative of the Subject1.Entire.COM.Pos is taken in a Vector variable formula, which would return the acceleration Vector for the Subject1.Entire.COM.Pos.

diff2(Subject1.Entire.COM.Pos)

int(V, BaseTime, Interval)

Calculates the integral of a Vector variable, V, where BaseTime is the starting point for performing the integration and Interval is the sampling interval for performing the calculation. In the following example, the integral of Forceplate0.Force is performed in a Vector variable formula from the time InitialTime, which is the first frame of the Activity, at an interval of 0.01 seconds.

int(Forceplate0.Force, InitialTime, 0.01)

spline(V, Interval)

Applies a Spline fit to a Vector variable, V, without any filtering applied. Interval is a sampling frequency, in seconds. In the following example, a spline fit is applied to the Vector variable MarkerSet1.heel_l.Pos in a Vector variable formula. The spline fit is being applied to the data at an interval of 0.01 seconds.

spline(MarkerSet1.heel_l.Pos, 0.01)

butterworth(V, Interval, Freq)

Applies a 4th order zero phase shift Butterworth filter of a specified frequency to a Vector variable, V. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. In the following example, a Butterworth filter is applied to the Vector variable Forceplate0.Force in a Vector variable formula. The interval used to apply the filter to the data is 0.01 seconds and the cut-off frequency is 20Hz.

butterworth(Forceplate0.Force, 0.01, 20)

chebyshev(V, Interval, Freq)

Applies a 4th order zero phase shift Chebyshev filter of a specified frequency to a Vector variable, V. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. In the following example, a Chebyshev filter is applied to the Vector variable Forceplate0.Force in a Vector variable formula. The interval used to apply the filter to the data is 0.01 seconds and the cut-off frequency is 20Hz.

chebyshev(Forceplate0.Force, 0.01, 20)

fftlowpass(V, Interval, Freq, Rolloff)

Applies a low-pass filter of a specified frequency to a Vector variable, V. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. Rolloff is the width of the transition band, in Hz. In the following example, an FFT low-pass filter is applied to the Scalar variable Forceplate0.Force in a Vector variable formula. The interval used to apply the filter 0.001 seconds, and the cut-off frequency is 20Hz with a roll off of 2Hz.

fftlowpass(Forceplate0.Force, 0.001, 20, 2)

fftthighpass(V, Interval, Freq, Rolloff)

Applies a high-pass filter of a specified frequency to a Vector variable, V. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. Rolloff is the width of the transition band, in Hz. In the following example, an FFT high-pass filter is applied to the Vector variable V1 in a Vector variable formula. The interval used to apply the filter 0.01 seconds, and the cut-off frequency is 20Hz with a roll off of 2Hz

fftthighpass(V1, 0.01, 20, 2)

fftnotch(V, Interval, Freq, Width, Rolloff)

Applies a notch filter of a specified frequency and width to a Vector variable, V. Rolloff is the width of the transition band, in Hz. In the following example, an FFT notch filter is applied to the Vector variable Forceplate0.Force in a Vector variable formula. The interval used to apply the filter 0.001 seconds, the frequency to remove from the signal is 60Hz with a width of 5 Hz (+/- 2.5Hz) and a roll off of 2Hz.

fftnotch(Forceplate0.Force, 0.001, 60, 5, 2)

Rotation Operators

The following list is of Rotation Operators. Rotation Operators will either use Rotation variable types or return a Rotation output. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". Rotations measures of orientation and can take the form of Euler angles, Quaternions or Cosine Matrices. For the following examples, R1 and R2 are Rotation variables.

INVALID

Sets the value for a Rotation variable to INVALID. In the following example, the condition angvel(SensorAxes.Ori) < 100 is evaluated within an IF statement in a Rotation variable formula. When the angvel(SensorAxes.Ori) is less than 100, the IF statement will return an output of INVALID, and when angvel(SensorAxes.Ori) is not less than 100, the IF statement will return an output of SensorAxes.Ori.

*if(angvel(SensorAxes.Ori) < 100, **INVALID**, SensorAxes.Ori)*

rot(Yaw, Pitch, Roll)

Composes a Rotation, where Yaw (Z), Pitch (Y) and Roll (X) are angles. In the following example, a Rotation is constructed for the Initial Object Transformation for an Object file, where the Yaw component is 180, the Pitch component is -45, and the Roll component is 0.

*axes(vec(0, 0, 0), **rot(180, -45, 0)**)*

rot(Q0, Q1, Q2, Q3)

Composes a Rotation using quaternions where Q0, Q1, Q2, Q3 correspond to w, x, y, z, respectively. In the example below, a Rotation is constructed in a Rotation variable formula for the Right Foot Segment orientation, where Q0 is Subject1.Segments.RightFoot.Ori.Quat.Q0, Q1 is Subject1.Segments.RightFoot.Ori.Quat.Q1, Q2 is Subject1.Segments.RightFoot.Ori.Quat.Q2, and Q3 is Subject1.Segments.RightFoot.Ori.Quat.Q3.

rot(Subject1.Segments.RightFoot.Ori.Quat.Q0, Subject1.Segments.RightFoot.Ori.Quat.Q1, Subject1.Segments.RightFoot.Ori.Quat.Q2, Subject1.Segments.RightFoot.Ori.Quat.Q3)

rot(XAxis, ZAxis)

Composes a Rotation using axes, where XAxis and ZAxis are Vector variables of the cosine matrix m(0,0), m(1,0), m(2,0), and m(0,2), m(1,2), m(2,2). In the example below, a Rotation is constructed in a Rotation variable formula using unit vectors for the upward and forward directions.

rot(UpwardUnitVector, ForwardUnitVector)

rot(M11, M12, …, M33)

Constructs a rotation matrix, specifying the nine elements of the rotation matrix. In the example below, a rotation matrix is constructed in a Rotation variable formula for the SensorAxes rigid body, where each element is a scalar variable.

rot(SensorAxes.Ori.Mat.M11,SensorAxes.Ori.Mat.M12,SensorAxes.Ori.Mat.M13,SensorAxes.Ori.Mat.M21,SensorAxes.Ori.Mat.M22,SensorAxes.Ori.Mat.M23,SensorAxes.Ori.Mat.M31,SensorAxes.Ori.Mat.M32,SensorAxes.Ori.Mat.M33)

R1 == R2

Examines whether the Rotation variables R1 and R2 are equal. In the following example, the condition Sensor1Axes.Ori == Sensor2Axes.Ori is evaluated within an IF statement in a Scalar variable formula. When the Sensor1Axes.Ori is equal to Sensor2Axes.Ori, the IF statement will return an output of 1, and when Sensor1Axes.Ori is not equal to Sensor2Axes.Ori, the IF statement will return an output of 0.

*if(**Sensor1Axes.Ori == Sensor2Axes.Ori**, 1, 0)*

R1 != R2

Examines whether the Rotation variables R1 and R2 are not equal. In the following example, the condition Sensor1Axes.Ori != Sensor2Axes.Ori is evaluated within an IF statement in a Scalar variable formula. When Sensor1Axes.Ori is not equal to Sensor2Axes.Ori, the IF statement will return an output of 1, and when Sensor1Axes.Ori is equal to Sensor2Axes.Ori, the IF statement will return an output of 0.

*if(**Sensor1Axes.Ori != Sensor2Axes.Ori**, 1, 0)*

R * V

Takes a direction/offset Vector variable, V, from the reference frame of Rotation variable, R, and places it with respect to the world reference frame's orientation. This operator is typically used with non-position vectors since it does not take the position of Rotation R into consideration and results in a Vector value. For instance, this could be used in calculating the current direction of a segment's anterior axis, whose direction is known relative to the sensor. In the following example, the Sensor Axes Rotation is multiplied by the direction Vector representing the anterior axis, whose orientation is reported relative to the sensor axes orientation, in a Vector variable formula. This operator performs the inverse operation to rel(V, R).

SensorAxes.Ori * AnteriorAxisRelSensor.Ori

R1 * R2

Takes Rotation variable R2 from the reference frame of Rotation variable R1 and places it with respect to the world reference frame's orientation. This operator does not consider the positions of R1 and R2. This operator can be used for calculating the current orientation for an Axes, whose orientation relative to a segment sensor is known. In the following example, SensorAxes.Ori is multiplied by SensorAxesRelSensor.Ori, whose orientation is reported relative to the sensor axes orientation, in a Rotation variable formula. This operator performs the inverse operation to rel(R1, R2).

SensorAxes.Ori * SensorAxesRelSensor.Ori

angvel(R)

Returns the angular velocity, in degrees/sec, of Rotation variable, R. This is analogous to performing the diff() operator for a vector and returns a Vector value. In the following example, the angular velocity for the Sensor Axes was performed in a Vector variable formula.

angvel(SensorAxes.Ori)

angacc(R)

Returns the angular acceleration, in degrees/sec^2, of Rotation variable, R. This is analogous to performing the diff2() operator for a vector and returns a Vector value. In the following example, the angular acceleration for the Sensor Axes was performed in a Vector variable formula.

angacc(SensorAxes.Ori)

rel(V, R)
Takes a direction/offset Vector variable, V, from the world reference frame and places it in the reference frame of Rotation variable, R. This operator is typically used with non-position vectors since it does not take the position of Rotation R into consideration and results in a Vector value. In the following example, the world up Vector variable is placed into the reference frame of a segment axes orientation in a Vector variable formula. This operator performs the inverse operation to R * V.

rel(WorldUp, SegmentAxes.Ori)

rel(R1, R2)
Takes Rotation variable R1 from the world reference frame and places it in the reference frame of Rotation variable R2. In the following example, the forearm orientation is put into the reference frame of the upper arm's orientation in a Rotation variable formula. This operator performs the inverse operation to R1 * R2.

rel(ForearmAxes.Ori, UpperarmAxes.Ori)

Axes Operators

The following list is of Axes Operators. Axes Operators will either use Axes variable types or return an Axes output. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". Axes are essentially rigid bodies and have a representation of both position and orientation. For the following examples, A1 and A2 are Axes variables.

INVALID

Sets the value for an Axes variable to INVALID. In the following example, the condition angvel(SensorAxes.Ori) < 100 is evaluated within an IF statement in an Axes variable formula. When the angvel(SensorAxes.Ori) is less than 100, the IF statement will return an output of INVALID, and when angvel(SensorAxes.Ori) is not less than 100, the IF statement will return an output of SensorAxes.

*if(angvel(SensorAxes.Ori) < 100, **INVALID**, SensorAxes)*

axes(Pos, Ori)

Composes an Axes variable, where Pos is a Vector and Ori is a Rotation. In the following example, an Axes is constructed in an Axes variable formula using the Vector Subject1.RightUpperArm.RightElbow.Pos and Rotation Xsens1.T00B410B7.Ori.

axes(Subject1.RightUpperArm.RightElbow.Pos, Xsens1.T00B410B7.Ori)

A1 == A2

Examines whether the Axes variables A1 and A2 are equal. In the following example, the condition Sensor1Axes == Sensor2Axes is evaluated within an IF statement in a Scalar variable formula. When the Sensor1Axes is equal to Sensor2Axes, the IF statement will return an output of 1, and when Sensor1Axes is not equal to Sensor2Axes, the IF statement will return an output of 0.

*if(**Sensor1Axes == Sensor2Axes**, 1, 0)*

A1 != A2

Examines whether the Axes variables A1 and A2 are not equal. In the following example, the condition Sensor1Axes != Sensor2Axes is evaluated within an IF statement in a Scalar variable formula. When Sensor1Axes is not equal to Sensor2Axes, the IF statement will return an output of 1, and when Sensor1Axes is equal to Sensor2Axes, the IF statement will return an output of 0.

*if(**Sensor1Axes != Sensor2Axes**, 1, 0)*

A * Pos

Takes a position Vector variable, Pos, from the reference frame of Axes variable, A, and places it in the world reference frame. This operator results in a Vector value and is typically used with position vectors. In the following example, the Axes variable, SensorAxes, is multiplied by the Vector variable, LandmarkPosRelSensor whose position is reported relative to SensorAxes, in a Vector variable formula. The result is the position of the LandmarkPosRelSensor variable in the world reference frame. This operator performs the inverse operation to relpos(Pos, A).

SensorAxes * LandmarkPosRelSensor

A * Ori

Takes an orientation variable, Ori, from the reference frame of an Axes variable, A, and places it in the world reference frame. This operator results in a Rotation value. In the following example, the Axes variable, SensorAxes, is multiplied by the Orientation variable, SensorAxes.Ori, whose orientation is reported relative to SensorAxes, in a Rotation variable formula. The result is the orientation of the SensorAxes.Ori variable in the world reference frame. This operator performs the inverse operation to rel(Ori, A).

SensorAxes* SensorAxes.Ori

A1 * A2

Takes an Axes variable A2 from the reference frame of Axes variable A1 and places it in the world reference frame. In the following example, an Axes variable SensorAxes is multiplied by the Axes variable SegmentAxesRelSensor, whose Axes are reported relative to SensorAxes, in an Axes variable formula. The result will report the Axes for SegmentAxesRelSensor in the world reference frame. This operator performs the inverse operation to rel(A1, A2).

SensorAxes * SegmentAxesRelSensor

inv(A)

Inverts Axes variable, A. This operator "undoes" the effect of the original value, so A * inv(A) = <identity>. With two objects, O1 and O2, where A is the axis system of O2 in the reference frame of O1, then inv(A) will be the axis system of O1 in the reference frame of O2. In the following example, the SegmentAxesRelSensor Axes are inverted in an Axes variable formula.

inv(SegmentAxesRelSensor)

relpos(Pos, A)

Takes a position Vector variable, Pos, from the world reference frame and places it in the reference frame of Axes variable, A. This operator returns a Vector value and is typically used with position vectors. In the following example, the position of LandmarkPos was put into the reference frame of the SensorAxes Axes variable in a Vector variable formula. This operator would take the digitized location of a bony landmark from the world reference frame and report it relative to a sensor or segment Axes. This operator performs the inverse operation to A * Pos.

relpos(LandmarkPos, SensorAxes)

reldir(V, A)

Takes a direction/offset Vector variable, V, from the world reference frame and places it in the orientation for the reference frame of Axes variable, A. This operator returns a Vector value and is typically used with non-position vectors. In the following example, the WorldUp vector is oriented relative to the SegmentAxes Axes in a Vector variable formula.

reldir(WorldUp, SegmentAxes)

rel(Ori, A)

Takes an Orientation variable, Ori, from the world reference frame and places it in the reference frame of Axes variable, A. This operator returns a Rotation value. In the following example, the ForearmAxes.Ori Orientation, reported in the reference frame of the world, is placed in the reference frame of the UpperArmAxes Axes in a Rotation variable formula. This operator performs the inverse operation to A * Ori

rel(ForearmAxes.Ori, UpperArmAxes)

rel(A1, A2)

Takes Axes variable A1 from the world reference frame and places it in the reference frame of Axes variable A2. In the following example, the ForearmAxes Axes from the world reference frame is placed in the reference frame of the UpperArmAxes Axes in an Axes variable formula. This operator performs the inverse operation to A1 * A2.

rel(ForearmAxes, UpperArmAxes)

hels(A1, A2)

Returns the parameters of the helical axis that transforms axis system A1 into axis system A2. S is the point that is closest to the origin of A1. The calculated helical axis parameter is a Vector value that is reported in the world coordinate system. In the following example, S is returned for the helical axis that transforms the ShankSensor Axes into the ThighSensor Axes, in a Vector variable formula.

hels(ShankSensor,ThighSensor)

heln(A1, A2)

Returns the parameters of the helical axis that transforms axis system A1 into axis system A2. N is the direction vector of the helical axis. The calculated helical axis parameter is a Vector value that is reported in the world coordinate system. In the following example, N is returned for the helical axis that transforms the ShankSensor Axes into the ThighSensor Axes, in a Vector variable formula.

heln(ShankSensor,ThighSensor)

helt(A1, A2)

Returns the parameters of the helical axis that transforms axis system A1 into axis system A2. T is the amount of translation along the axis that takes us from A1 to A2, in meters/second. The calculated helical axis parameter is a Scalar value. In the following example, T is returned for the helical axis that transforms the ShankSensor Axes into the ThighSensor Axes, in a Scalar variable formula.

helt(ShankSensor,ThighSensor)

helphi(A1, A2)

Returns the parameters of the helical axis that transforms axis system A1 into axis system A2. Phi is the amount of rotation about the axis that takes us from A1 to A2, in degrees/second. The calculated helical axis parameter is Scalar value. In the following example, Phi is returned for the helical axis that transforms the ShankSensor Axes into the ThighSensor Axes, in a Scalar variable formula.

helphi(ShankSensor,ThighSensor)

ihels(A)

Returns the parameters of the instantaneous helical axis of axis system A. S is the point that is closest to the origin of A. The calculated helical axis parameter is a Vector value that is reported in the world coordinate system. In the following example, S is returned for the instantaneous helical axis for the ShankSensor Axes, in a Vector variable formula.

ihels(ShankSensor)

iheln(A)

Returns the parameters of the instantaneous helical axis of axis system A. N is the direction vector of the helical axis. The calculated helical axis parameter is a Vector value that is reported in the world coordinate system. In the following example, N is returned for the instantaneous helical axis for the ShankSensor Axes, in a Vector variable formula.

iheln(ShankSensor)

ihelt(A)

Returns the parameters of the instantaneous helical axis of axis system A. T is the amount of translation along the axis, in meters/second. The calculated helical axis parameter is a Scalar value. In the following example, T is returned for the instantaneous helical axis for the ShankSensor Axes, in a Scalar variable formula.

ihelt(ShankSensor)

ihelphi(A)

Returns the parameters of the instantaneous helical axis of axis system A. Phi is the amount of rotation about the axis, in degrees/second. The calculated helical axis parameter is a Scalar value. In the following example, Phi is returned for the instantaneous helical axis for the ShankSensor Axes, in a Scalar variable formula.

ihelphi(ShankSensor)

spline(A, Interval)

Applies a Spline fit to an Axes variable, A, without any filtering applied. Interval is a sampling frequency, in seconds. In the following example, a spline fit is applied to the Axes variable SensorAxes in an Axes variable formula. The spline fit is being applied to the data at an interval of 0.01 seconds.

spline(SensorAxes, 0.01)

butterworth(A, Interval, Freq)

Applies a 4th order zero phase shift Butterworth filter of a specified frequency to an Axes variable, A. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. In the following example, a Butterworth filter is applied to the Axes variable SensorAxes in an Axes variable formula. The interval used to apply the filter to the data is 0.01 seconds and the cut-off frequency is 20Hz.

butterworth(SensorAxes, 0.01, 20)

chebyshev(A, Interval, Freq)

Applies a 4th order zero phase shift Chebyshev filter of a specified frequency to an Axes variable, A. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. In the following example, a Chebyshev filter is applied to the Axes variable SensorAxes in an Axes variable formula. The interval used to apply the filter to the data is 0.01 seconds and the cut-off frequency is 20Hz.

chebyshev(SensorAxes, 0.01, 20)

fftlowpass(A, Interval, Freq, Rolloff)

Applies a low-pass filter of a specified frequency to an Axes variable, A. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. Rolloff is the width of the transition band, in Hz. In the following example, an FFT low-pass filter is applied to the Axes variable SensorAxes in an Axes variable formula. The interval used to apply the filter 0.01 seconds, and the cut-off frequency is 20Hz with a roll off of 2Hz.

fftlowpass(SensorAxes, 0.01, 20, 2)

ffthighpass(A, Interval, Freq, Rolloff)

Applies a high-pass filter of a specified frequency to an Axes variable, A. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz. Rolloff is the width of the transition band, in Hz. In the following example, an FFT high-pass filter is applied to the Axes variable SensorAxes in a Vector variable formula. The interval used to apply the filter 0.01 seconds, and the cut-off frequency is 20Hz with a roll off of 2Hz

ffthighpass(SensorAxes, 0.01, 20, 2)

fftnotch(A, Interval, Freq, Width, Rolloff)

Applies a notch filter of a specified frequency and width to an Axes variable, A. Rolloff is the width of the transition band, in Hz. In the following example, an FFT notch filter is applied to the Axes variable SensorAxes in an Axes variable formula. The interval used to apply the filter 0.01 seconds, the frequency to remove from the signal is 60Hz with a width of 5 Hz (+/- 2.5Hz) and a roll off of 2Hz.

fftnotch(SensorAxes, 0.01, 60, 5, 2)

Time Operators

The following list is of Time Operators. Time Operators will either use Time variable types or return a Time output. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". Times are time stamps of the form 7-10-2011 14:16:53:848. When used in calculations time types return time in seconds. For the following examples, T1 and T2 are Time variables.

INVALID

Sets the value for a Time variable to INVALID. In the following example, the condition BeginTime = INVALID is shown how it could be used in a script, such as to initialize the Time script variable BeginTime at the start of a script by setting it to INVALID. Note how the expression is followed by a semi-colon ";" because it is in a script.

BeginTime = **INVALID**;

time(PartSec, Sec, Min, Hour, Day, Mon, Year)

Composes a Time variable. In the following example, a Time variable is defined as January 25, 2025 7:30am 0 seconds and 0 part seconds, in a Time variable formula.

time(0,0,30,7,25,1,2025)

now()

Sets the value for a Time variable to the current time. It can also be used in formulas to use the current time, such as now() – 1, which would represent 1 second prior to the current time in real time or 1 second prior to the current cursor position in an Activity. In the following example, the condition BeginTime = now() is shown how it could be used in a script, such as to initialize the Time script variable BeginTime at the start of a while condition by setting it to the current time. Note how the expression is followed by a semi-colon ";" because it is in a script.

BeginTime = **now()**;

T1 == T2

Examines whether the Time variables T1 and T2 are equal. In the following example, the condition BeginTime == INVALID is evaluated within an IF statement in a Scalar variable formula. When BeginTime is equal to INVALID, the IF statement will return an output of 1, and when BeginTime is not equal to INVALID, the IF statement will return an output of 0.

if(**BeginTime == INVALID**, 1,0)

T1 != T2

Examines whether the Time variables T1 and T2 are not equal. In the following example, the condition BeginTime != INVALID is evaluated within an IF statement in a Scalar variable formula. When the BeginTime is not equal to INVALID, the IF statement will return an output of 1 and when BeginTime is equal to INVALID, the IF statement will return an output of 0.

if(**BeginTime != INVALID**, 1,0)

T1 < T2

Examines whether the Time variable T1 is less than T2. In the following example, the condition now() < MaxRightKneeFlexionTime is evaluated within an IF statement in a Scalar variable formula. When the current time is less than MaxRightKneeFlexionTime, the IF statement will return an output of 1 and when the current time is not less than MaxRightKneeFlexionTime, the IF statement will return an output of 0.

if(**now() < MaxRightKneeFlexionTime**, 1, 0)

T1 <= T2

Examines whether the Time variable T1 is less than or equal to T2. In the following example, the condition now() <= MaxRightKneeFlexionTime is evaluated within an IF statement in a Scalar variable formula. When the current time is less than or equal to MaxRightKneeFlexionTime, the IF statement will return an output of 1, and when the current time is not less than or equal to MaxRightKneeFlexionTime, the IF statement will return an output of 0.

if(**now() <= MaxRightKneeFlexionTime**, 1, 0)

T1 > T2

Examines whether the Time variable T1 is greater than T2. In the following example, the condition now() > BeginTime is evaluated within an IF statement in a Scalar variable formula. When the current time is greater than BeginTime, the IF statement will return an output of 1 and when the current time is not greater than BeginTime, the IF statement will return an output of 0.

if(**now() > BeginTime**, 1, 0)

T1 >= T2

Examines whether the Time variable T1 is greater than or equal to T2. In the following example, the condition now() >= BeginTime is evaluated within an IF statement in a Scalar variable formula. When the current time is greater than or equal to BeginTime, the IF statement will return an output of 1 and when the current time is not greater than or equal to BeginTime, the IF statement will return an output of 0.

if(**now() >= BeginTime**, 1, 0)

T + Secs

Adds some amount of time in seconds to Time variable, T. In the following example, the condition Forceplate0.Force.Mag > 20 && attime(Forceplate0.Force.Mag, now() + 0.01) < 20 is evaluated within an IF statement in a Scalar variable formula. An attime() operator is used to evaluate the value of a force plate at a time of now() + 0.01 seconds to determine if it was less than 20 Newtons. When both conditions are True, the IF statement will return an output of 1, and when both conditions are not True, the IF statement will return an output of 0.

*if(Forceplate0.Force.Mag > 20 && attime(Forceplate0.Force.Mag, **now() + 0.01**)< 20, 1, 0)*

T – Secs

Subtracts some amount of time in seconds from Time variable, T. In the following example, the condition Forceplate0.Force.Mag > 20 && attime(Forceplate0.Force.Mag, now() - 0.01) < 20 is evaluated within an IF statement in a Scalar variable formula. An attime() operator is used to evaluate the value of a force plate at a time of now() - 0.01 seconds to determine if it was less than 20 Newtons. When both conditions are True, the IF statement will return an output of 1, and when both conditions are not True, the IF statement will return an output of 0.

*if(Forceplate0.Force.Mag > 20 && attime(Forceplate0.Force.Mag, **now() - 0.01**) < 20, 1, 0)*

T1 – T2

Subtracts Time variable T2 from Time variable T1. In the following example, InitialTime is subtracted from FinalTime. Both Time variables are automatically created within each Activity to represent the first frame and last frame of the Activity. When subtracted, the results will be the total length of the Activity, in seconds.

FinalTime - InitialTime

attime(Value, T)

Returns the value of some variable, Value, at Time, T. This operator takes on whatever variable type Value is defined as. In the following example, the condition Forceplate0.Force.Mag > 20 && attime(Forceplate0.Force.Mag, now() - 0.01) < 20 is evaluated within an IF statement in a Scalar variable formula. The attime() operator returns the value of Forceplate0.Force.Mag at the time now() – 0.01. When both conditions are True, the IF statement will return an output of 1, and when both conditions are not True, the IF statement will return an output of 0.

*if(Forceplate0.Force.Mag > 20 && **attime(Forceplate0.Force.Mag, now() - 0.01)** < 20, 1, 0)*

current(Value)

Returns the value of some variable, Value, at the current time. This operator takes on whatever variable type Value is defined as. In the following example, TargetCoMValue = current(Subject1.Entire.COM.Pos) is displayed how it could be used in a script, such as to capture the Center of Mass Vector position at a desired point during the execution of a script. In this case after an OKMessage dialog is closed the TargetCoMValue Vector script variable is set to the current value for Subject1.Entire.COM.Pos. Note how the expression is followed by a semi-colon ";" because it is in a script.

OKMessage("Click OK when ready to capture the CoM position.");
*TargetCoMValue = **current(Subject1.Entire.COM.Pos)**;*

String Operators

The following list is of String Operators. String Operators will either use String variable types or return a String output. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". Strings are groupings of alphanumeric characters. The characters are enclosed within quotation marks. For the following examples, S1 and S2 are String variables.

INVALID

Sets the value for a String variable to INVALID. In the following example, the condition Forceplate0.Force.Mag > 20 is evaluated within an IF statement in a String variable formula. When Forceplate0.Force.Mag is greater than 20 Newtons, the IF statement will return the string value for the force plate magnitude, str(Forceplate0.Force.Mag,2. When Forceplate0.Force.Mag is not greater than 20 Newtons, the IF statement will return INVALID for the string. If the string were displayed in an Animation window, the string value for the force plate magnitude will be displayed when the condition is True and the string value would be set to INVALID, and nothing is displayed when the condition is False.

if(Forceplate0.Force.Mag>20, str(Forceplate0.Force.Mag,2), ***INVALID****)*

“”

Composes a String value by enclosing text or numbers in quotation marks. In the following example, a string was created for the text Right Knee Flexion, by surrounding it in quotations within the Text edit field for Text added to an Animation window.

"Right Knee Flexion: " *+ str(RightKneeFlexion,2)*

str(N)

Converts the Integer variable, N, to a String value. In the following example, a string was created including the conversion of the Integer variable now().Min to a String value within the Text edit field for Text added to an Animation window.

"Time: " + str((now().Hour%12),0) + ":" + ***str(now().Min)***

str(X, Precision)

Converts the Scalar variable, X, to a String value with the specified Precision for the number of decimal places. In the following example, a string was created including the conversion of the Scalar variable RightKneeFlexion to a String value with precision of 2 decimals within the Text edit field for Text added to an Animation window.

"RightKneeFlexion: " + ***str(RightKneeFlexion,2)***

S1 == S2

Examines whether the String variables S1 and S2 are equal. In the following example, the condition RightThighString == LeftThighString is evaluated within an IF statement in a Scalar variable formula. When the String variable RightThighString is equal to the String variable LeftThighString, the IF statement will return an output of 1, and when RightThighString is not equal to LeftThighString, the IF statement will return an output of 0.

*if(****RightThighString == LeftThighString****, 1,0)*

S1 != S2

Examines whether the String variables S1 and S2 are not equal. In the following example, the condition RightThighString != LeftThighString is evaluated within an IF statement in a Scalar variable formula. When the String variable RightThighString is not equal to the String variable LeftThighString, the IF statement will return an output of 1, and when the String variable RightThighString is equal to the String variable LeftThighString, the IF statement will return an output of 0.

*if(**RightThighSting != LeftThighString**, 1,0)*

S1 + S2

Combines String variables S1 and S2. In the following example, a string was created including the combination of a string that was composed by surrounding text in quotations, RightKneeFlexion:, and the conversion of the Scalar variable RightKneeFlexion to a String value within the Text edit field for Text added to an Animation window.

"RightKneeFlexion: " + str(RightKneeFlexion,2)

Color Operators

The following list is of Color Operators. Color Operators will either use Color variable types or return a Color output. When an argument is optional, it will appear in brackets "[ ]". When used in a script, these operators should be followed by a semi-colon ";". Colors control the color assigned to elements, such as an object or skeleton segment in the Animation window. For the following examples, C1 and C2 are Color variables.

INVALID

Sets the value for a Color variable to INVALID. In the following example, the condition RightKneeFlexion > 40 is evaluated within an IF statement in a Color variable formula. When RightKneeFlexion is greater than 40, the IF statement will return the color value for white, color(1.0, 1.0, 1.0). When RightKneeFlexion is not greater than 40, the IF statement will return INVALID for the color. If the color variable were used to control the color of a segment of the Subject in an Animation window, the color for the segment would be white when the condition is True, and it would be INVALID, or have no color, when the condition is False

if(RightKneeFlexion > 40,color(1.0, 1.0, 1.0), ***INVALID****)*

color(Red, Green, Blue)

Composes a Color value comprised of Red, Green, and Blue components, where each value is less than or equal to 1. In the following example, the color for the True condition of an IF statement, is set to white, color(1.0, 1.0, 1.0).

if(RightKneeFlexion > 40, ***color(1.0, 1.0, 1.0)****, INVALID)*

C1 == C2

Examines whether the Color variables C1 and C2 are equal. In the following example, the condition RightThighColor == LeftThighColor is evaluated within an IF statement in a Scalar variable formula. When the Color variable RightThighColor is equal to the Color variable LeftThighColor, the IF statement will return an output of 1 and when the Color variable RightThighColor is not equal to the Color variable LeftThighColor, the IF statement will return an output of 0.

*if(****RightThighColor == LeftThighColor****, 1,0)*

C1 != C2

Examines whether the Color variables C1 and C2 are not equal. In the following example, the condition RightThighColor != LeftThighColor is evaluated within an IF statement in a Scalar variable formula. When the Color variable RightThighColor is not equal to the Color variable LeftThighColor, the IF statement will return an output of 1, and when the Color variable RightThighColor is equal to the Color variable LeftThighColor, the IF statement will return an output of 0.

*if(****RightThighColor != LeftThighColor****, 1,0)*

Script Program Flow Functions

The following list is of script Program Flow functions. Script Program Flow functions are used to control the flow and logic within a script. Some Script Program Flow functions are immediately followed by a semi-colon ";", while others only require that items within the curly brackets "{}" be followed by a semi-colon ";". Each script Program Flow function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

if (Condition) {}

IF statement, where Condition, is the condition being evaluated and the value entered in the curly brackets is what will be executed when the script evaluates the IF statement when the condition is True. In the following example the Boolean script variable IsCancelButtonPressed is the condition evaluated for the IF statement. If the script gets to the IF statement while the condition is True, the operators within the curly brackets will be evaluated. Otherwise, the contents within the curly brackets will be bypassed. Only operators within the curly brackets are followed by semi-colons ";".

```
if (IsCancelButtonPressed)
{
        CloseDialog("Dialog");
        DialogOpen = FALSE;
}
```

else{}

ELSE condition, where the value entered in the curly brackets is what the script will run when the IF statement is False. In the following example, when the condition for the IF statement is false, the operators within the curly brackets for the else condition will be evaluated. When the IF statement is True, the contents within the curly brackets for the else condition will be bypassed. Only operators within the curly brackets are followed by semi-colons ";".

```
if (Condition1 == 0)
        {S1="Left";}
else {S1="Right";}
```

while(Condition) {}

WHILE condition, where Condition is the condition being evaluated and the value entered in the curly brackets is what the script will run while this condition is True. In the following example, when Boolean script variable IsCancelButtonPressed is not True, the script will enter the while condition and continue to perform whatever operators are in the curly brackets until IsCancelButtonPressed becomes True. In this example, nothing is performed within the while condition curly brackets, so it's used to keep the script from proceeding until the condition is no longer True. This may be the case while waiting for a dialog to close. Only operators within the curly brackets are followed by semi-colons ";".

```
while (!IsOKButtonPressed)
{
}
```

goto Label;

Allows for the script to jump to a specified label elsewhere is the script. The jump can go either forwards or backwards. The label, however, cannot be contained within an argument if the goto function is outside of this argument. The label should be followed by a colon ":" and the goto label function should be followed by a semi-colon ";". In the following example, the script will jump to the label NewRecording: when goto NewRecording; is encountered in the script.

```
NewRecording:

    if(YesNoCancelInteger == NO)
    {
    CloseActivity();
    n=n-1;
    goto NewRecording;
    }
```

return;

Prevents the current sequence of commands from proceeding and returns to the invoking function. In the following example, when the IF statement condition is True while it is encountered in a script, the script will stop executing. This script function should be followed by a semi-colon ";".

```
if (OKCancelInteger == CANCEL)
    {return;}
```

RunScript(Name);

Calls a script from within a script. The script being called to run must be within the same tab of the Components window as the script invoking the call. For instance, if the initial script is a Permanent Script, any scripts being called must also be Permanent Scripts. In the following example, the script HeightWeightDialog is run from another script and the name of the script is enclosed within quotes. This script function should be followed by a semi-colon ";".

```
RunScript("HeightWeightDialog");
```

Delay(Secs);

Pauses the script for a specified time. Takes a scalar argument. Specifying a zero will perform Windows background processing and return immediately afterwards. Any other value will result in the script pausing for the specified time. This can be useful for when introducing a break in the processing of the script is desired or after operations such as Saving, Opening and Closing files. This function can introduce a delay after non-blocking script functions. In the example below, a delay is used to control the recording process. A 10 second delay is processed after the start of a recording and before the recording is stopped. This script function should be followed by a semi-colon ";".

```
StartRecording();
Delay(10);
StopRecording();
```

Script Variable Functions

The following list is of script Variable functions. Script Variable functions are used to assign a value to Permanent Script Variables or Analysis Script Variables through a script. Each script Variable function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script Variable function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

Var = Exp;

Sets a variable, Var, equal to an Expression, Exp. Var is a script variable of any type. In the following example, the String Script Variable S1 is set to the string "Left" when the condition in the IF statement is True or "Right" when the condition is not True. This script function should be followed by a semi-colon ";".

```
if (Condition1 == 0)
        {S1="Left";}
else {S1="Right";}
```

SetPastValue(Var, Exp, StartTime, StopTime);

Sets a variable to a certain value for a specified amount of time. Var is a script variable of any type, Exp holds a value, and StartTime and StopTime define the period over which Var should hold that value. In the following example, StartPointEvents is a Scalar script variable that initially gets set to zero from InitialTime - 1 to FinalTime +1. Then, C3D event data within an Activity are evaluated for events and the StartPointEvents Scalar script variable is updated to 1 for the Time period of StartPointEventTime to StartPointEventTime + SamplingInterval, where StartPointEventTime is a Time variable and SamplingInteval is a scalar variable. Iterations of this process are performed within a while loop until the while condition is no longer True. The result is a single variable, StartPointEvents, that displays a 1 for every time there is a C3D event with the base name "EventSet1.StartPoint". This script function should be followed by a semi-colon ";".

```
SetPastValue(StartPointEvents, 0.0, InitialTime - 1.0, FinalTime + 1.0);
StartPointEventIndex = 1;
SetExpression(StartPointEventTime, "InitialTime + EventSet1.StartPoint.TimeOffset");
while (StartPointEventTime != INVALID)
{
        SetPastValue(StartPointEvents, 1.0, StartPointEventTime, StartPointEventTime
        + SamplingInterval);
        StartPointEventIndex = StartPointEventIndex + 1;
        SetExpression(StartPointEventTime, "InitialTime + EventSet1.StartPoint" +
        str(StartPointEventIndex) + ".TimeOffset");
}
```

SetExpression(Var, Exp);

Changes the formula for a user-defined variable. The first argument is the name of the variable. The second argument is a string expression which will become the text of the variable's formula. The variable may be of any type, but it must be a regular variable, not a script variable. In the following example, StartPointEventTime, is a Time analysis variable whose formula field is updated to InitialTime + EventSet1.StartPoint.TimeOffset when the script executes. This script function should be followed by a semi-colon ";".

```
SetPastValue(StartPointEvents, 0.0, InitialTime - 1.0, FinalTime + 1.0);
StartPointEventIndex = 1;
SetExpression(StartPointEventTime, "InitialTime + EventSet1.StartPoint.TimeOffset");
while (StartPointEventTime != INVALID)
{
    SetPastValue(StartPointEvents, 1.0, StartPointEventTime, StartPointEventTime +
    SamplingInterval);
    StartPointEventIndex = StartPointEventIndex + 1;
    SetExpression(StartPointEventTime, "InitialTime + EventSet1.StartPoint" +
    str(StartPointEventIndex) + ".TimeOffset");
}
```

SetRandom(Scalar, MinValue, MaxValue);

Generates a random Scalar Number, where Scalar is the name for a Scalar script variable and MinValue and MaxValue represent the min and max values that can be generated, respectively. In the following example, RandomScriptScalar is a Scalar script variable whose random number will be assigned a value between 0 and 100 when the script is executed. This script function should be followed by a semi-colon ";" .

```
SetRandom(RandomScriptScalar,0,100);
```

SetNormal(Scalar, Mean, StdDev);

Generates a Scalar Number in a normal (Gaussian) distribution, where Scalar is the name for a Scalar script variable. The Mean and StdDev arguments determine the mean and standard deviation of the distribution, respectively. In the following example, RandomScriptScalar is a Scalar script variable whose random number will be assigned a value based on a Gaussian distribution with a mean of 10 and standard deviation of 5 when the script is executed. This script function should be followed by a semi-colon ";".

```
SetNormal(RandomScriptScalar,10,5);
```

GetActivityValue(Value, Exp, Time[, ActivityName]);

Runs from the Live workspace and extracts a value from an open activity, where Value is the script variable to receive the extracted value in the Live Window, Exp is the expression to be evaluated within the Activity (can be an existing variable within the Activity or a formula), and Time is the time at which to evaluate the expression within the Activity. The optional argument, ActivityName, allows for the name of the Activity to be specified. Otherwise, the most recently opened or recorded Activity is assumed. In the following example, MaxRightKneeFlexFromActivity is a Scalar script variable in the Live workspace that will be assigned the maximum right knee flexion value over the entire Activity. The expression will be evaluated in the open Activity at FinalTime, which is the end of the Activity. No optional ActivityName argument was included, so the most recently opened or recorded Activity would be used. This script function should be followed by a semi-colon ";".

```
GetActivityValue(MaxRightKneeFlexFromActivity,
max(Subject1.Segments.RightShank.Angles.Flexion, InitialTime, FinalTime,
0.01),FinalTime);
```

Script Messaging Functions

The following list is of script Messaging functions. Script Messaging functions are used to generate message dialogs and sounds. Each script Messaging function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script Messaging function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

OKMessage(Prompt);

Generates a Message with an OK button being displayed to the user. The Prompt should be surrounded by quotation marks. In the following example, an OK message window would be presented with an OK button and the message "Click OK to proceed" when the script is executed. The script would not proceed until the message dialog is closed. This script function should be followed by a semi-colon ";".

OKMessage("Click OK to proceed.");

OKCancelMessage(Prompt, Result);

Generates a Message with OK and Cancel buttons being displayed to the user. The Prompt should be surrounded by quotation marks and the Result is an Integer script variable that gets set to 1 when OK is selected and 2 when Cancel or the Close buttons are selected. In the following example, a message window with OK and Cancel buttons would be presented with the message "Do you want to save this recording?", when the script is executed. The script would not proceed until the message is closed. The Integer script variable, OKCancelInteger, would be updated based on which button is selected. The OKCancelMessage function is followed in the script by IF statements that will determine what happens next based on the value of the Integer script variable, OKCancelInteger. The software recognizes that OK is equal to 1 and CANCEL is equal to 2, so the words OK and CANCEL were used instead of the integers 1 and 2 in the IF statement conditions. This script function should be followed by a semi-colon ";".

```
OKCancelMessage("Do you want to save this recording?", OKCancelInteger);
//If OK is selected, save and close the Activity and repeat the recording process –
//go to the NewRecording label
if(OKCancelInteger == OK)
{
        SaveActivityAs("Trial_" + str(n));
        CloseActivity();
        goto NewRecording;
}
//If Cancel is selected, close the current Activity without saving, and quit out of the script.
if(OKCancelInteger == CANCEL)
{
        CloseActivity();
        return;
}
```

OKCancelMessage(Prompt, StylusName, Result);

Generates a Message with OK and Cancel buttons being displayed to the user and also references to a stylus, StylusName, whose Is-button-pressed expression can be used in place of the OK button. The Prompt and StylusName should be surrounded by quotation marks and the Result is an Integer script variable that gets set to 1 when OK is selected and 2 when Cancel or the Close buttons are selected. In the following example, a message window with OK and Cancel buttons would be presented with the message “Please place the stylus tip at the crown of the subject's head and hold still for the upcoming 2 seconds.”, when the script is executed. The script would not proceed until the message is closed, either by clicking one of the buttons or via the Is-button-pressed expression for Stylus1. The Integer script variable, SetupSubject.StylusInteger, would be updated based on which button is selected. The OKCancelMessage function is followed in the script by IF statements that will determine what happens next based on the value of the Integer script variable, SetupSubject.StylusInteger. The software recognizes that OK is equal to 1 and CANCEL is equal to 2, so the words OK and CANCEL were used instead of the integers 1 and 2 in the IF statement conditions. This script function should be followed by a semi-colon “;”.

```
OKCancelMessage("Please place the stylus tip at the crown of the subject's head and hold still for the upcoming 2 seconds.", "Stylus1", SetupSubject.StylusInteger);
//If CANCEL is selected, go back to the previous dialog – go to the OpenTheDialog label.
if (SetupSubject.StylusInteger==CANCEL)
{
        goto OpenTheDialog;
}
//If OK is selected, proceed with the process of measuring the subject’s head using the stylus.
if (SetupSubject.StylusInteger==OK)
{
        SetupSubject.HeightMeasureTime=now();
        while(now()-SetupSubject.HeightMeasureTime <3)
        {}
        SetupSubject.Height=avg(Stylus1.Tip.Pos.Z,SetupSubject.HeightMeasureTime,0.01;
        SetupSubject.HeightManual=SetupSubject.Height;
        //Provide a message to confirm and accept the height of the subject.
        YesNoCancelMessage("The height of the subject is: " + str(SetupSubject.Height,3)+
        " m.
        If you want to accept this height, click Yes. If you want to remeasure the subject
        click No.",SetupSubject.HeightAcceptance);
        //If YES is selected and the height is accepted, go back to the previous dialog –
        //go to the OpenTheDialog label
        if(SetupSubject.HeightAcceptance==4) //Yes
        {
                goto OpenTheDialog;
        }
        //If NO is selected and the height is not accepted, go back to re-measure the height –
        //go to the MeasureHeight label
        if(SetupSubject.HeightAcceptance==8) //No
        {
                goto MeasureHeight;
        }
        //If CANCEL is selected, reset the variables and go back to the previous dialog –
        //go to the OpenTheDialog label
        if(SetupSubject.HeightAcceptance==2) //Cancel
        {
                SetupSubject.IsHeightButtonClicked=FALSE;
                goto OpenTheDialog;
        }
}
```

YesNoCancelMessage(Prompt, Result);

Generates a Message with Yes, No, and Cancel buttons being displayed to the user. The Prompt should be surrounded by quotation marks and the Result is an Integer script variable that gets set to 4 when Yes is selected, 8 when No is selected and 2 when the Cancel or the Close buttons are selected. In the following example, a message window with Yes, No and Cancel buttons would be presented with the message "Do you want to save this recording?", when the script is executed. The script would not proceed until the message is closed. The Integer script variable, YesNoCancelInteger, would be updated based on which button is selected. The YesNoCancelMessage function is followed in the script by IF statements that will determine what happens next based on the value of the Integer script variable, YesNoCancelInteger. The software recognizes that YES is equal to 4, NO is equal to 8, and CANCEL is equal to 2, so the words YES, NO, and CANCEL were used instead of the integers 4, 8, and 2 in the IF statement conditions. This script function should be followed by a semi-colon ";".

```
YesNoCancelMessage("Do you want to save this recording?", YesNoCancelInteger);
//If Yes is selected, save and close the Activity and repeat the recording process –
//go to the New Recording label
if(YesNoCancelInteger == YES)
{
        SaveActivityAs("Trial_" + str(n));
        CloseActivity();
        goto NewRecording;
}
//If No is selected, close the current Activity without saving, reset the number for recordings
//and repeat the recording process – go to the NewRecording label
if(YesNoCancelInteger == NO)
{
        CloseActivity();
        n=n-1;
        goto NewRecording;
}
//If Cancel is selected, close the current Activity without saving, and quit out of the script.
if(YesNoCancelInteger == CANCEL)
{
        CloseActivity();
        return;
}
```

Beep();

Produces an audible "beep" sound. This function is non-blocking, so the script will continue without waiting for the beep to finish. In the following example, a beep is generated after the Delay of 2 seconds. This script function should be followed by a semi-colon ";".

```
OKMessage("Click OK and tell the participant to wait until they hear the beep.");
Delay(2);
Beep();
```

PlayTone(Freq, Duration);

Plays a tone at the specified frequency, Freq, and duration, Duration. This function is non-blocking, so the script will continue without waiting for the tone to finish. In the following example, a 500Hz tone is generated for 2 seconds after the Delay of 2 seconds. This script function should be followed by a semi-colon ";".

```
OKMessage("Click OK and tell the participant to wait until they hear the beep.");
Delay(2);
PlayTone(500, 2);
```

PlaySound(Filename, Volume);

Plays a sound file, Filename, at a volume, Volume, between 0 and 1. The Filename should be specified with an extension (.wav, .mp3, or m4a). When specified without a file path, the software will look for the sound file in the user and shared Sounds folders. This function is non-blocking, so the script will continue without waiting for the sound file to finish playing. In the following example, an audio file “go-go.mp3” is played at a volume of 0.5 after the Delay of 2 seconds. This script function should be followed by a semi-colon “;”.

```
OKMessage("Click OK and tell the participant to wait until they hear the beep.");
Delay(2);
PlaySound("go-go.mp3", 0.5);
```

Script Dialog Functions

The following list is of script Dialog functions. Script Dialog functions are used to generate custom dialogs in the software, including those for conveying messages to the software user or to provide a way for variable values to be specified through an edit field within a dialog. Each script Dialog function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script Dialog function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own. An entire example script containing all script Dialog functions will be provided at the end of the script Dialog functions section.

CreateTextControl(Name, Text[, MinWidth]);

Creates a text control that can be added to a dialog window with the name, Name, surrounded by quotation marks and text, Text, surrounded by quotation marks. MinWidth, is an optional argument for the minimum width of the text control. This provides a way for text to be added to a dialog for instructions, prompts or other messages. In the following example, a TextControl named TextControlSubjectID was created with the text to be displayed as "Subject ID: ". The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateTextControl("TextControlSubjectID", "Subject ID: ");

CreateButtonControl(Name, HasBeenPressed, Text);

Creates a button control that can be added to a dialog window with the name, Name, surrounded by quotation marks, a script Boolean variable for whether the button has been pressed, HasBeenPressed, and text for the button, Text, surrounded by quotation marks. In the following example, a ButtonControl named ButtonControl was created and a Boolean script variable, IsOKButtonPressed, was assigned as the HasBeenPressed Boolean variable, and the text to be displayed for the button was specified as "OK". The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateButtonControl("ButtonControl", IsOKButtonPressed, "OK");

CreateCheckboxControl(Name, Checked, Text);

Creates a checkbox that can be added to a dialog window with the name, Name, surrounded by quotation marks, a script Boolean variable for whether the box has been checked, Checked, and text for the checkbox, Text, surrounded by quotation marks. In the following example, a CheckboxControl named CheckboxControlLeft was created and a Boolean script variable, IsCheckboxLeftHandEnabled, was assigned as the Checked Boolean variable, and the text to be displayed for the checkbox was specified as "Left". The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateCheckboxControl("CheckboxControlLeft", IsCheckboxLeftHandEnabled, "Left");

CreateIntegerControl(Name, Integer, MinValue, MaxValue);

Creates an Integer control field that can be added to a dialog window with the name, Name, surrounded by quotation marks, a script Integer variable, Integer, and MinValue and MaxValue representing the min and max values that can be entered, respectively. In the following example, an IntegerControl named IntegerControlAge was created and an Integer script variable, Age, was assigned as the Integer variable, and the minimum and maximum values were specified as 5 and 100, respectively. The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateIntegerControl("IntegerControlAge", Age, 5, 100);

CreateScalarControl(Name, Scalar, MinValue, MaxValue, Precision);

Creates a Scalar control field that can be added to a dialog window with the name, Name, surrounded by quotation marks, a script Scalar variable, Scalar, and MinValue and MaxValue representing the min and max values that can be entered, respectively. Precision determines the number of digits used to express the Scalar script variable. In the following example, a ScalarControl named ScalarControlHeight was created and a Scalar script variable, Height, was assigned as the Scalar variable, and the minimum and maximum values were specified as 0 and 3, respectively. The precision for the Scalar script variable is set to 4. The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateScalarControl("ScalarControlHeight", Height, 0, 3, 4);

CreateStringControl(Name, String);

Creates a String control field that can be added to a dialog window with the name, Name, surrounded by quotation marks and permanent String variable, String. In the following example, a StringControl named StringControlSubjectID was created and a String script variable, SubjectID, was assigned as the String variable. The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateStringControl("StringControlSubjectID", SubjectID);

CreateDateControl(Name, Date);

Creates a date control with the name, Name, surrounded by quotation marks and a Time script variable, Date. The control displays the date portion of a time script variable, in the form of mm/dd/yyyy. If you click on the date display, a calendar interface opens, allowing you to select a new date. Doing so will update both the date display and the script variable, Date, itself. In the following example, a DateControl named DateControlDOB was created and a Time script variable, DOB, was assigned as the Time variable. The name of this control will be used when adding it to a ControlGrid. This script function should be followed by a semi-colon ";".

CreateDateControl("DateControlDOB", DOB);

CreateHorzControlGrid(Name);

Creates the horizontal grid components of a dialog window with name, Name, surrounded by quotation marks. This creates a horizontal grid that controls to be added to within a dialog. In the example below, a horizontal control grid, HorzControlGridSubjectID, was created. The name of this control grid will be used when adding it to other control grids, adding controls to it, or when creating a dialog. This script function should be followed by a semi-colon ";".

CreateHorzControlGrid("HorzControlGridSubjectID");

CreateVertControlGrid(Name);

Creates the vertical grid components of a dialog window with name, Name, surrounded by quotation marks. This creates a vertical grid that controls to be added to within a dialog. In the example below, a vertical control grid, VertControlGrid1, was created. The name of this control grid will be used when adding it to other control grids, adding controls to it, or when creating a dialog. This script function should be followed by a semi-colon ";".

CreateVertControlGrid("VertControlGrid1");

AddToControlGrid(ControlName, ControlGridName);

Adds any of the control types using their name, ControlName, surrounded by quotation marks, to the components of a control grid, ControlGridName. In the following example, the string control, StringControlSubjectID, is added to the horizontal control grid, HorzControlGridSubjectID. This script function should be followed by a semi-colon ";"**.**

AddToControlGrid("StringControlSubjectID", "HorzControlGridSubjectID");

CreateDialog(Name, Caption, ControlGridName, DefaultButtonControlName[, Width]);

Creates a dialog window with the name, Name, surrounded by quotation marks, text displayed in the dialog title bar, Caption, surrounded by quotation marks, using the components added to the control grid, ControlGridName, surrounded by quotation marks, and the button control "DefaultButtonControlName", surrounded by quotation marks. Width, is an option argument for the width of the dialog. In the following example, a dialog named SubjectInfoDialog was created with the caption "Subject Info Dialog" in the dialog title bar. The dialog will use the control grid, "VertControlGrid1", and use the button control, ButtonControl. The optional width argument is set to 300 to control the width of the dialog. The name of this dialog will be referenced when using the open, close, and is dialog open Dialog script functions. This script function should be followed by a semi-colon ";".

CreateDialog("SubjectInfoDialog","Subject Info Dialog", "VertControlGrid1", "ButtonControl", 300);

OpenDialog(Name);

Opens the dialog window with the name, Name, surrounded by quotation marks. In the following example, the dialog SubjectInfoDialog is opened. This script function should be followed by a semi-colon ";".

OpenDialog("SubjectInfoDialog");

CloseDialog(Name);

Closes the dialog window with the name, Name, surrounded by quotation marks. In the following example, the dialog SubjectInfoDialog is closed. This script function should be followed by a semi-colon ";".

CloseDialog("SubjectInfoDialog");

IsDialogOpen(Name, IsOpen);

Checks to see if the dialog window with the name, Name, surrounded by quotation marks, is open. IsOpen is a Boolean variable that will return True or False based on whether the dialog, Name, is open. In the following example, the status for whether the dialog SubjectInfoDialog is open is checked and the Boolean variable, IsDialogOpen, is updated for the current status of the dialog. This script function should be followed by a semi-colon ";".

```
IsDialogOpen("SubjectInfoDialog", IsDialogOpen);
if(!IsDialogOpen)
{
        goto CloseTheDialog;
}
```

The following example demonstrates all the script Dialog functions in a script to create a dialog.

```
//Variables used in this script
//IsOKButtonPressed                    (Boolean Script Variable)
//IsDialogOpen                         (Boolean Script Variable)
//IsCheckboxRightHandEnabled           (Boolean Script Variable)
//IsCheckboxLeftHandEnabled            (Boolean Script Variable)
//Age                                  (Integer Script Variable)
//Height                               (Scalar Script Variable)
//Weight                               (Scalar Script Variable)
//SubjectID                            (String Script Variable)
//DOB                                  (Time Script Variable)

//label for goto "ReturnToTheDialog" function
ReturnToTheDialog:

//Initialize Variables
IsOKButtonPressed = FALSE;
IsDialogOpen=FALSE;

//Create Text Controls
CreateTextControl("TextControl","Enter Subject information:
");
CreateTextControl("TextControlHandedness", "Handedness: ");
CreateTextControl("TextControlAge", "Age: ");
CreateTextControl("TextControlHeight", "Height (m) : ");
CreateTextControl("TextControlWeight", "Weight (kg): ");
CreateTextControl("TextControlSubjectID", "Subject ID: ");
CreateTextControl("TextControlDOB", "Date of Birth: ");

//Create Checkbox Controls
CreateCheckboxControl("CheckboxControlRight", IsCheckboxRightHandEnabled, "Right");
CreateCheckboxControl("CheckboxControlLeft", IsCheckboxLeftHandEnabled, "Left");

//Create Integer Controls
CreateIntegerControl("IntegerControlAge" ,Age, 5, 100);

//Create Scalar Controls
CreateScalarControl("ScalarControlHeight", Height, 0, 3, 4);
CreateScalarControl("ScalarControlWeight", Weight, 0, 500, 4);

//Create String Controls
CreateStringControl("StringControlSubjectID", SubjectID);

//Create Date Controls
CreateDateControl("DateControlDOB", DOB);

//Create Button Controls
CreateButtonControl("ButtonControl", IsOKButtonPressed, "OK");

//Create Vertical and Horizontal Control Grids that controls will be added to
CreateHorzControlGrid("HorzControlGridGeneralText");
CreateHorzControlGrid("HorzControlGridHandedness");
CreateHorzControlGrid("HorzControlGridAge");
CreateHorzControlGrid("HorzControlGridHeightWeight");
CreateHorzControlGrid("HorzControlGridSubjectID");
CreateHorzControlGrid("HorzControlGridDOB");
CreateHorzControlGrid("HorzControlGridOKButton");
```

```
CreateVertControlGrid("VertControlGrid1");

//Add Controls to the Control Grids (the order controls are added to a control grid will
//determine their order within the horizonal or vertical control grid)
AddToControlGrid("TextControl", "HorzControlGridGeneralText");

AddToControlGrid("TextControlHandedness", "HorzControlGridHandedness");
AddToControlGrid("CheckboxControlRight", "HorzControlGridHandedness");
AddToControlGrid("CheckboxControlLeft", "HorzControlGridHandedness");

AddToControlGrid("TextControlAge", "HorzControlGridAge");
AddToControlGrid("IntegerControlAge", "HorzControlGridAge");

AddToControlGrid("TextControlHeight", "HorzControlGridHeightWeight");
AddToControlGrid("ScalarControlHeight", "HorzControlGridHeightWeight");
AddToControlGrid("TextControlWeight", "HorzControlGridHeightWeight");
AddToControlGrid("ScalarControlWeight", "HorzControlGridHeightWeight");

AddToControlGrid("TextControlSubjectID", "HorzControlGridSubjectID");
AddToControlGrid("StringControlSubjectID", "HorzControlGridSubjectID");

AddToControlGrid("TextControlDOB", "HorzControlGridDOB");
AddToControlGrid("DateControlDOB", "HorzControlGridDOB");

AddToControlGrid("ButtonControl", "HorzControlGridOKButton");

AddToControlGrid("HorzControlGridGeneralText", "VertControlGrid1");
AddToControlGrid("HorzControlGridSubjectID", "VertControlGrid1");
AddToControlGrid("HorzControlGridAge", "VertControlGrid1");
AddToControlGrid("HorzControlGridDOB", "VertControlGrid1");
AddToControlGrid("HorzControlGridHeightWeight", "VertControlGrid1");
AddToControlGrid("HorzControlGridHandedness", "VertControlGrid1");
AddToControlGrid("HorzControlGridOKButton", "VertControlGrid1");

CreateDialog("SubjectInfoDialog", "Subject Info Dialog", "VertControlGrid1", "ButtonControl",
300);

//Open the dialog and specify what happens while the dialog is open
OpenDialog("SubjectInfoDialog");

while (!IsOKButtonPressed)
{
        //Set the checkbox controls so that only one can be selected
        if (IsCheckboxRightHandEnabled) {IsCheckboxLeftHandEnabled = FALSE;}
        if (IsCheckboxLeftHandEnabled) {IsCheckboxRightHandEnabled = FALSE;}

        //Check whether the dialog is still open
        IsDialogOpen("SubjectInfoDialog", IsDialogOpen);
        if(!IsDialogOpen)
        {
                //Close the dialog if it is not open.  Go to the CloseTheDialog label.
                goto CloseTheDialog;
        }
}
//label for goto “CloseTheDialog” function
CloseTheDialog:
```

```
//If neither checkbox is selected for handedness, close the dialog, present a message to the
//user and open the dialog again.
if (!IsCheckboxLeftHandEnabled  && !IsCheckboxRightHandEnabled)
        {
        CloseDialog("SubjectInfoDialog");
        OKMessage("Please make sure Right or Left is selected for handedness");
        goto ReturnToTheDialog;
        }

//Close the dialog and exit the script
CloseDialog("SubjectInfoDialog");
return;
```

Script File Functions

The following list is of script File functions. Script File functions are used to replicate the functionality of menu items found in the File menu or for handling and sorting through files and folders. Each script File function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script File function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

Delete(FilePath);

Deletes the specified file, surrounded by quotation marks. 'FilePath' requires the entire file path, including file extension. In the following example, an Activity whose file path and name are specified would be deleted. This script function should be followed by a semi-colon ";".

Delete("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities\Activity1.iac");

Rename(OldFilePath, NewFilePath);

Renames the file 'OldFilePath' to the file 'NewFilePath'. Both arguments require the entire file path, including file extension, to be surrounded by quotation marks. This function can be used to rename or move a file. In the following example, the Activity file whose file path and name are specified will be renamed from Gait Walk1.iac to Gait Walk1_Edited.iac in the same folder. This script function should be followed by a semi-colon ";".

Rename("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities\Gait Walk1.iac",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities\Gait Walk1_Edited.iac");

Copy(SourceFilePath, DestFilePath);

Copies the file 'SourceFilePath' to the file 'DestFilePath'. Both arguments require the entire file path, including file extension, to be surrounded by quotation marks. In the following example, the Activity file whose file path and name are specified will be copied from the source file path to the destination file path. This script function should be followed by a semi-colon ";".

Copy("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities\Gait Walk1.iac",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\Shared\Activities\Gait Walk1.iac");

BrowseFile(FolderPath, FilenameFilter, FileDescription, SelectedFilename, SelectedFolderPath);

Opens a file-browser dialog but doesn't actually open any files, instead it will return the filename and path in the last two arguments for the Activity that is selected. 'FolderPath' is the initial folder to look in and requires the entire file path, including file extension, to be surrounded by quotation marks. 'FilenameFilter' is the filter to apply (e.g. "*.iac") in the file-browser dialog, where (*) represents wildcard characters. The entire argument should be surrounded by quotation marks. 'FileDescription' is the string shown to the user in front of the filter (e.g. "Activity Files") for the Types of files in the file-browser dialog, and the last two arguments are string script variables that receive the file information (folder path and file name, including the extension). The file-browser dialog will open and display the FileDescription + (FilenameFilter) (e.g.Activity Files (*.iac)) in the "Files of type:" drop list within the file-browser dialog. In the following example, the specified directory will open in the file-browser dialog and the "Files of type" drop list will specify "Activity Files (*.iac)" and only files ending with *.iac will be displayed in the window. The folder path and file name for a selected Activity file will be sent to the FolderPath and FileName script String variables, respectively. This script function should be followed by a semi-colon ";".

BrowseFile("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities", "*.iac", "Activity Files", FileName, FolderPath);

NextFilename(FolderPath, FilenameFilter, CurrentFilename, NextFilename);

Scans through a folder and returns the file names one by one, sorted by name. 'FolderPath' is a string specifying the absolute path of the folder, surrounded by quotation marks. 'FilenameFilter' restricts the search to certain types of files (e.g. "*.iac") where (*) represents wildcard characters. The entire argument should be surrounded by quotation marks. If "*" is specified by itself, no filter will be applied. 'CurrentFilename' is a string variable that would usually be the previous string value returned by this function, or "" to start a new search. 'NextFilename' must be a string script variable, which receives the return value. This is either the next filename in the folder, or "" if there are no more files remaining. In the following example, the specified directory will be scanned using the FilenameFilter "*.iac", where the CurrentFilename is a String script variable, FileName, and the NextFilename is the String script variable, NextFileName. This script function should be followed by a semi-colon ";".

FileName= NextFileName;
NextFilename("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities", "*.iac", FileName, NextFileName);

NextFolderName(ParentPath, FolderNameFilter, CurrentFolderName, NextFolderName);

Scans through a directory and returns the folder names one by one, sorted by name. 'ParentPath' is a string specifying the path of the parent directory, surrounded by quotation marks. 'FolderNameFilter' restricts the search to certain folders, where (*) represents wildcard characters. If "*" is specified by itself, no filter will be applied. The entire argument should be surrounded by quotation marks. 'CurrentFolderName' is a string variable that would usually be the previous string value returned by this function, or "" to start a new search. 'NextFolderName' must be a string Script variable, which receives the return value. This is either the next filename in the folder, or "" if there are no more files remaining. In the following example, the specified directory will be scanned using the FolderNameFilter "*", which applies no filter, where the CurrentFolderName is a String script variable, FolderName, and the NextFolderName is the String script variable, NextFolderName. This script function should be followed by a semi-colon ";".

FolderName= NextFolderName;
NextFolderName("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files", "*", FolderName, NextFolderName);

ParseFilePath (FilePath, FolderPath, BaseName, Extension);

Parses a file path. The first argument, 'FilePath' can be a filename or path String, and the remaining arguments must be String script variables, which will receive the components of the path. If the first argument is just a filename, then the FolderPath variable will be set to an empty string. Note that the Extension variable will not include the leading period character. In the following example, the specified file path with file name string is parsed. The String script variables, FolderPath, FileName, and Extension will receive the FolderPath, BaseName, and Extension components of the path, respectively. The FolderPath would be "C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities", the FileName would be "GaitWalk1", and the Extension would be "iac". This script function should be followed by a semi-colon ";".

ParseFilePath("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities\GaitWalk1.iac", FolderPath, FileName, Extension);

ParseKinatraxFilename(Filename, Timestamp, TeamName, UniformNumber, PlayerName, Location);

Extracts the parameters from a Kinatrax C3D filename and stores them in script variables. Timestamp is a time variable, UniformNumber and Location are Integer variables, and TeamName and PlayerName are string variables. In the following example, the specified Kinatrax filename String is parsed, where Date, is the Timestamp Time script variable that will receive the time component from the filename, TeamName, is the is the TeamName String script variable that will receive the team name component from the filename, UniformNumber, is the UniformNumber Integer script variable that will receive the uniform number component from the filename, PlayerName, is the PlayerName String script variable that will receive the player name component from the filename, and Location, is the Location Integer script variable that will receive the location component from the filename. A location of Home will return a 0 and a location of Away will return a 1. This script function should be followed by a semi-colon ";".

ParseKinatraxFilename("2021_07_23_19_11_35_Team_A_30_John_Doe_Home.c3d", Date, TeamName, UniformNumber, PlayerName, Location);

SetScreenCaptureParams(Region[, WindowName[, FrameRate]]);

Modifies the region (0=desktop, 1=main window, 2=child window) for screen capture parameters. Optional arguments for the child window name to be captured and frame rate associated with screen capture can be specified. If optional arguments are not specified, the existing parameters in the File|Edit Screen Capture Parameters dialog will be used. In the following example, the Animation child window was selected, and the screen capture measurement rate was set to 15Hz. This script function should be followed by a semi-colon ";".

SetScreenCaptureParams(2, "Animation", 15);

EditScreenCaptureParams();

Opens the File|Edit Screen Capture Parameters menu dialog. In the following example, the File|Edit Screen Capture Parameters menu dialog would open, and the settings can be modified manually. This script function should be followed by a semi-colon ";".

EditScreenCaptureParams();

BeginSessionCapture([Name[, FolderPath]]););

Begins a session capture using the parameters from the File|Edit Screen Capture Parameters menu dialog or set through the SetScreenCaptureParams(); script function. Optional Name and FolderPath arguments can be specified for saving the screen recording. The default save location is the Screen Captures User subfolder and the default filename is in the format of Session_Year.Month.Day.Min.Sec.PartSec.mp4. In the following example, the optional name and folder path locations were specified for the video file generated from the session capture that will be started by this script operator. This script function should be followed by a semi-colon ";".

BeginSessionCapture("Animation",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Screen Captures");
Delay(10);
EndSessionCapture();

EndSessionCapture();

Ends the screen capture recording. The video is saved based on the BeginSessionCapture(); script function optional arguments or with the default settings where the folder path will be the Screen Captures User subfolder and the filename in the format of Session_Year.Month.Day.Min.Sec.PartSec.mp4. In the following example, the session capture ends following a 10 second delay after the session was started. This script function should be followed by a semi-colon ";".

BeginSessionCapture("Animation",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Screen Captures");
Delay(10);
EndSessionCapture();

TakeScreenshot([Name[, FolderPath]]);

Takes a screenshot using the parameters from the File|Edit Screen Capture Parameters menu dialog or set through the SetScreenCaptureParams(); script function. Optional Name and FolderPath options can be specified for saving the image. The default save location is the Screen Captures User subfolder and the default filename is in the format of Screenshot_Year.Month.Day.Min.Sec.PartSec.png. In the following example, the optional name and folder path locations were specified for the image file generated. This script function should be followed by a semi-colon ";".

TakeScreenshot("Animation",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Screen Captures");

ReprocessInBackground([ShowOptions]);

Runs the File|Reprocess In Background menu item, which reprocesses videos associated with Activities that have not yet been processed and proceeds to the markerless processing engine, when applicable. The optional 'ShowOptions' argument can be set to TRUE to open the File|Reprocess In Background menu dialog. The default value is FALSE. In the following example, the optional argument is set to TRUE and the File|Reprocess In Background menu dialog will open, where setting can be modified manually. This script function should be followed by a semi-colon ";".

ReprocessInBackground(TRUE);

StopReprocessingInBackground();

Ends the Reprocessing in background operation. When the current Activity being processed is completed, the process will stop. In the following example, background reprocessing will be stopped. This script function should be followed by a semi-colon ";".

StopReprocessingInBackground();

Script Setup Functions

The following list is of script Setup functions. Script Setup functions are associated with processes involving the setup of an experiment or workspace. Each script Setup function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script Setup function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

OpenWorkspace([Name]);

Opens a workspace. Name is an optional argument, which if absent will cause a file-browser dialog to open. When specifying the filename without a file path, the current User Workspace folder is used. The Name should be contained within quotation marks and the file extension is not required. In the following example, the file-browser dialog would open to allow for the selection of a workspace (*.iws) file since no optional name argument was provided. This script function should be followed by a semi-colon ";".

OpenWorkspace();

SaveWorkspace();

Saves open workspace as the Current.iws workspace file. This is the file that gets loaded when initially logging in to The MotionMonitor xGen software. The Current.iws workspace file also gets automatically saved during certain processes in the software, so this file is more intended to save progress within a session than to save the software settings in a desired state. In the following example, the Current.iws workspace would be saved. This script function should be followed by a semi-colon ";".

SaveWorkspace();

SaveWorkspaceAs();

Saves a workspace with a given name after opening a file-browser dialog. The workspace should be saved with a specific name once the software settings are in the desired state that you want to get back to, such as for the start of an experiment session. In the following example, the file-browser dialog would open to allow for the current software settings to be saved to an existing or new workspace file (*.iws). This script function should be followed by a semi-colon ";".

SaveWorkspaceAs();

ImportComponentSet([Name]);

Imports a component set. Name is an optional argument, which if absent will cause a file-browser dialog to open. When specifying the filename without a file path, the current User Component Sets folder is used. The Name should be contained within quotation marks and the file extension should not be included. In the following example, a component set, ForcePlateSetup.ics, including the force plate hardware parameters settings, including information for the alignment in the world axes, are loaded into the live workspace. This script function should be followed by a semi-colon ";".

ImportComponentSet(ForcePlateSetup);

ExportComponentSet(Name, Merge, Comp1, …, CompN);

Exports a component set. The first argument is a string or string variable containing the name of the component set to be exported (new or existing), the second argument is a Boolean Merge argument which determines whether an existing component set will be overwritten or appended. The third argument is the list of components to be exported to the file. This list can be of any length, and the listed names should not be contained within quotations. In the following example, the component set, PlayerBio.ics, is appended to because the Merge argument is set to True. The variables FirstName, LastName, DOB, PlayerID, and Height are written to the component set. This script function should be followed by a semi-colon ";".

ExportComponentSet(PlayerBio, TRUE, FirstName, LastName, DOB , PlayerID, Height);

ExportMicrophone(Name);

Exports the .wav audio file for the specified microphone device, Name, surrounded by quotation marks. The file is saved to the Export User subfolder by default. This script function is performed from an Activity. In the following example, an audio file for Microphone1 is exported from the Activity. This script function should be followed by a semi-colon ";".

ExportMicrophone("Microphone1");

Activate(Name);

Activates the hardware component with the given name, surrounded by quotation marks. In the following example, Camera1 would be activated. This script function should be followed by a semi-colon ";".

Activate("Camera1");

ActivateAll();

Activates all hardware listed in the Components Setup tab. In the following example, all hardware listed in the Components Setup tab would be activated and an error message or warning would be provided for any devices that couldn't be activated or whose physical configurations have changed, such as when digital force plates are added or removed following a previous activation. This script function should be followed by a semi-colon ";".

ActivateAll();

Deactivate(Name);

Deactivates the hardware component with the given name, surrounded by quotation marks. In the following example, Camera1 would be deactivated. This script function should be followed by a semi-colon ";".

Deactivate("Camera1");

DeactivateAll();

Deactivates all hardware listed in the Components Setup tab. In the following example, all hardware listed in the Components Setup tab would be deactivated. This script function should be followed by a semi-colon ";".

DeactivateAll();

AllActivated(AllActivated);

Sets its argument to True if all hardware devices listed in the Components Setup tab are fully activated, where “AllActivated” is a Boolean variable. This is used to confirm that all hardware devices were successfully activated. In the following example, this function was used during the hardware activation procedure to confirm that all devices were activated. If all hardware were successfully activated, the Boolean script variable, IsAllActivated, would be set to True and the subsequent IF statement would result in an activation successful message. This script function should be followed by a semi-colon “;”.

```
AllActivated(IsAllActivated);
if(IsAllActivated)
{
        // Announce Successful Activation
        OKMessage("Activation Successful !");
}
```

AnyDeactivated(AnyDeactivated);

Sets its argument to True if any hardware devices listed in the Components Setup tab have failed to activate, where “AnyDeactivated” is a Boolean variable. This is used to confirm if any hardware devices are deactivated. In the following example, this function was used during the hardware activation procedure to confirm if any devices did not activate. If any hardware were not activated, the Boolean script variable, IsAnyDeactivated, would be set to True and the subsequent IF statement would result in an activation failed message. This script function should be followed by a semi-colon “;”.

```
AnyDeactivated(IsAnyDeactivated);
if(IsAnyDeactivated)
{
        // Announce Unsuccessful Activation
        OKMessage("Activation Failed!  Please confirm all hardware are connected and
        powered on.");
}
```

EditWorldAxesSettings();

Opens the World Axes dialog. In the following example, a World Axes dialog would open for the settings to be confirmed or updated manually. This script function should be followed by a semi-colon “;”.

```
EditWorldAxesSettings();
```

Calibrate(Name [, Hidden]);

Runs the calibration procedure for the subject or hardware. The first argument is the name of the hardware to calibrate, surrounded by quotation marks. The second argument is an optional Boolean which displays or hides the accompanying calibration dialogs. The default setting is FALSE. In the following example, the AMTIGen5DeviceSet1 force plates will be calibrated or zeroed. There was no optional Hidden argument specified, so all the calibration messages will be displayed. This script function should be followed by a semi-colon “;”.

```
Calibrate("AMTIGen5DeviceSet1");
```

CalibrateAllCameras([Reuse Dynamic Images]);

Performs the Calibrate All cameras process, using the calibration parameters from the first camera listed under the Hardware node in the Components Setup tab. The optional Boolean argument to Reuse Dynamic Images toggles whether new images or existing intrinsic images will be used for the camera calibration. If not specified, the existing setting for reusing dynamic images in the camera parameters panel will be used. In the following example, all the cameras listed under the Hardware node in the Components Setup tab will be calibrated to determine their position for use in Markerless or object overlay applications. Since no optional argument was included, the existing setting for reusing dynamic images in the camera parameters panel will be used. This script function should be followed by a semi-colon ";".

CalibrateAllCameras();

Align(Name[, DeviceIndex]);

Performs the hardware alignment procedure for the device, Name, if supported for that hardware type, surrounded by quotation marks. The second argument is an optional Device Index for hardware devices that have multiple components that support the alignment option. In the following example, the AMTIGen5Device force plate 1 will be aligned based on the alignment settings configured in the parameters panel for that device. Since multiple USB force plates can be listed under the AMTIGen5DeviceSet1 hardware node, a device index of 1 was selected to specify a particular force plate. This script function should be followed by a semi-colon ";".

Align("AMTIGen5DeviceSet1", 1);

EnableSubjectSegment(SubjectName, SegmentName, Enabled);

Enables or disables any segment from a script. The first argument is the existing subject name, the second argument is the segment name, and the third argument is a Boolean for the enabled status. Both the SubjectName and SegmentName should be surrounded by quotation marks. In the following example, the Boolean script variable, SetupSubject.IsRShankSegmentEnabled, is used to determine whether the RightShank is enabled for Subject1. When the Boolean variable is True, the segment will be enabled for setup and data collection. When the Boolean variable is False, the segment will not be enabled and will not be calibrated when Subject1 is calibrated. This script function should be followed by a semi-colon ";".

EnableSubjectSegment("Subject1", "Right Shank",
SetupSubject.IsRShankSegmentEnabled);

SetSubjectNeutralStance(SubjectName, NeutralStance);

Allows you to select a subject's neutral stance using a script. The first argument is the subject name, surrounded by quotation marks. The second argument is an integer that corresponds to the neutral stance configuration, see below. 0= Arms down, Thumbs forward, 1= Arms down, Thumbs lateral, 2= Arms lateral, Thumbs forward, 3= Arms lateral, Thumbs upward, and 4= Arms forward, Thumbs upward. In the following example, the neutral stance configuration with Arms down, thumbs directed laterally (1) was selected for Subject1. This script function should be followed by a semi-colon ";".

SetSubjectNeutralStance("Subject1", 1);

SetupSubject(Name);

Performs the Subject setup procedure for the subject, Name, surrounded by quotation marks. In the following example, the calibration process for Subject1 was initiated, based on the settings in the Components setup tab for Subject1. This script function should be followed by a semi-colon ";".

SetupSubject("Subject1");

EnableCameraViewfinder(CameraName,Enabled);

Enables or disables the viewfinder (live view) for a camera. It takes two arguments: 'CameraName' is the name of the camera, surrounded by quotation marks, and 'Enabled' is a Boolean to enable or disable the viewfinder. In the following example, the viewfinder for Camera1 is disabled. This script function should be followed by a semi-colon ";".

EnableCameraViewfinder("Camera1", FALSE);

LoadEmbededActivity(Name);

Loads data from a previously recorded activity under the name, Name, surrounded by quotation marks, within the current live session or recorded activity. In the following example, the Activity from the specified path, was loaded into the current live workspace, making data from the embedded Activity available in the Live workspace. This script function should be followed by a semi-colon ";".

LoadEmbeddedActivity("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities\Gait Walk1.iac");

DeleteEmbeddedActivity(Name);

Deletes embedded activity. Name of the embedded Activity refers to the name as it is listed under the Embedded Activity node in the Components setup tab and should be surrounded by quotation marks. In the following example, the name of the embedded activity to be deleted was, EmbeddedActivity1. This script function should be followed by a semi-colon ";".

DeleteEmbeddedActivity("EmbeddedActivity1");

Script Recording Functions

The following list is of script Recording functions. Script Recording functions relate to processes involving Recording or Activities. Each script Recording function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script Recording function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

EditRecordingParams();

Opens the Recording Parameters dialog. In the following example, the Record|Edit Recording Parameters dialog is opened, and parameters can be manually updated. This script function should be followed by a semi-colon ";".

EditRecordingParams();

StartRecording();

Initiates the recording process in the live workspace. In the following example, a recording was started, using the settings form the Edit Recording Parameters dialog. This script function should be followed by a semi-colon ";".

StartRecording();

StopRecording();

Terminates the recording process in the live workspace. In the following example, the current recording process is stopped. This script function should be followed by a semi-colon ";".

StopRecording();

CancelRecording();

Cancels the recording process in the live workspace. In the following example, the current recording process is cancelled, with all temporary data for the recording being discarded. This script function should be followed by a semi-colon ";".

CancelRecording();

RunBiofeedback(Name);

Runs Biofeedback of a given name, surrounded by quotation marks, in the live workspace. In the following example, the biofeedback, Biofeedback1, is run. Thus, initiating the biofeedback recording. This script function should be followed by a semi-colon ";".

RunBiofeedback("Biofeedback1");

OpenActivity([Name[, FolderPath[, FilenameFilter]]]);

Opens a file-browser dialog. An optional Name can be specified as the first argument to open an activity with a particular Name in the default Activities folder for the current User. An optional FolderPath can be specified for the second argument to designate the folder in which the specified activity Name resides. If "" is specified for the first argument, a file-browser dialog will open with the specified directory from the second argument. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. FilenameFilter is an optional filter to apply (e.g. "*.iac") in the file-browser dialog, where (*) represents wildcard characters. To use 'FilenameFilter' pass an empty string for the Name argument, and either an empty string or a valid path for FolderPath. In the following example, the Activity and folder path for the Activity to open were specified. This script function should be followed by a semi-colon ";".

OpenActivity("Gait Walk1",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities");

OpenActivityTiled(Region[, Name[Folderpath[, FilenameFilter]]]);

Similar to the OpenActivity() function, except it will open the activity tiled in a particular region of the main window. 'Region' corresponds to a number 0 – 3, corresponding to the (0) left, (1) right, (2) top, and (3) bottom regions of the main window, respectively. Within each region, the tiled activity will always appear in the "last" position, meaning at the bottom of the left and right regions, or at the right-hand end of the top or bottom regions if there are already windows in these regions. Name is an optional filename argument; if omitted, the user will be shown a file-browser dialog. If Name is supplied, FolderPath is an optional argument to specify a folder other than the User Activities directory. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. FilenameFilter is an optional filter to apply (e.g. "*.iac") in the file-browser dialog, where (*) represents wildcard characters. To use FilenameFilter pass an empty string for the Name argument, and either an empty string or a valid path for FolderPath. In the following example, the Activity and folder path for the Activity to open were specified and the Activity will open in the bottom region of the main window. This script function should be followed by a semi-colon ";".

OpenActivityTiled(3, "Gait Walk1.iac", "C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities");

OpenActivityHidden(Name[, FolderPath]);

Opens an activity, Name, invisibly and with any warning messages suspended. If applicable, it will automatically reprocess any video files in the background. An optional FolderPath argument can be used to specify a folder other than the User Activities directory. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. In the following example, the Activity and folder path for the Activity to open invisibly were specified. This script function should be followed by a semi-colon ";".

OpenActivityHidden("Gait Walk1.iac", "C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Activities");

SaveActivity();

Saves an open activity using the default name. This function is performed from within the Activity. In the following example, the Activity this function was performed from was saved, using the default Activity name given to the Activity upon completion of the recording. This script function should be followed by a semi-colon ";".

SaveActivity();

SaveActivityAs([Name[, FolderPath]]);

Saves the most recently recorded activity as, Name, if specified. If no name is specified, a file-browsing dialog will open. If the optional FolderPath argument is specified in addition to a Name argument, the file will be saved in the specified directory. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. In the following example, a file-browser dialog will open allowing for the control over where the Activity is saved and its filename. This script function should be followed by a semi-colon ";".

SaveActivityAs();

CloseActivity([Name]);

Closes an open activity, starting with the most recently opened or recorded first. An optional Name argument can be specified to close an activity with a particular Name, where the Name argument is accepted with or without an extension and is surrounded by quotation marks. In the following example, the Activity, Gait Walk1.iac, is closed. This script function should be followed by a semi-colon ";".

CloseActivity("Gait Walk1.iac");

CloseAllActivities();

Closes all currently open Activities regardless of whether they are visible and does not alert the user – even if they are modified. In the following example, all open Activities were closed. This script function should be followed by a semi-colon ";".

CloseAllActivities();

LoadC3DImportAnalysis([Name[, FolderPath]]);

Allows for the selection of a new analysis file that will be applied to imported C3D files. Opens an 'Open File' window. An optional Name can be specified as the first argument to open an analysis with a particular Name in the default Analyses folder for the current User. An optional 'FolderPath' can be specified for the second argument to designate the folder in which the specified analysis resides. All arguments should be surrounded by quotation marks. In the following example, the Analysis and folder path for the Analysis to be applied to C3D imports were specified. This script function should be followed by a semi-colon ";".

LoadC3DImportAnalysis("Gait Walk Data Reduction.ian", "C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses");

ImportC3DFile([Hidden]);

Imports C3D files. There is an optional Boolean Hidden argument, which if TRUE will bypass the Static C3D import setup and after selecting the static C3D file will immediately proceed to selecting dynamic C3D files for importing. In the following example, the C3D import process was performed with the Boolean Hidden argument set to True. So, the static C3D import setup step will be skipped. This script function should be followed by a semi-colon ";".

ImportC3DFile(TRUE);

ImportC3DFileWithoutStaticFile ([Name[, FolderPath[, KeepOpen]]]);

Imports C3D files without using or referencing a Static C3D file. The command can take an optional Name and FolderPath argument. If omitted, the user will be queried for the C3D file(s) to import. The Name argument is accepted with or without an extension. A biomechanical model definition for the C3D import should have already been created within the existing workspace. The Subject setup parameter, 'Assume rigid bodies to be aligned with segment axes', would also need to be used with this function. When the optional argument, KeepOpen, is set to TRUE, it causes the imported Activity to be retained in memory and is not immediately written to disk. The default setting for this argument, if not specified, is FALSE. All arguments should be surrounded by quotation marks. In the following example, no optional arguments were specified, so a file-browser dialog will open to select the C3D files to be imported, and no Activities will be kept open. This script function should be followed by a semi-colon ";".

ImportC3DFileWithoutStaticFile();

CloseC3DActivity();

Closes C3D file. This script function can be used to close the Static C3D file when importing C3D files. In the following example, the open static C3D file is closed. This script function should be followed by a semi-colon ";".

CloseC3DActivity();

ToggleLooping();

Toggles looping Activity playback setting on/off. This function is performed from within an Activity. In the following example, the looping playback setting was toggled from the state it is currently in, to the alternate state. If the Activity playback looping was previously disabled, so that it would only play once, it would now playback in a continuous loop when played. This script function should be followed by a semi-colon ";".

ToggleLooping();

Play();

Runs playback of an Activity. This function is performed from within an Activity. In the following example, the Activity playback is started. This script function should be followed by a semi-colon ";".

Play();

Pause();

Pauses playback of an Activity. This function is performed from within an Activity. In the following example, the Activity playback is paused. This script function should be followed by a semi-colon ";".

Pause();

Stop();

Stops playback of an Activity. This function is performed from within an Activity. In the following example, the Activity playback is stopped. This script function should be followed by a semi-colon ";".

Stop();

SetCurrentTime(CurrentTime);

Run from an Activity, this will set the Activity cursor and currently displayed activity time from the passed value. The CurrentTime argument must be an absolute time (e.g., InitialTime + 2.0), not just the offset from the start of the Activity. In the following example, the cursor in the Activity moved to the 1 second mark in the Activity. This script function should be followed by a semi-colon ";".

SetCurrentTime(InitialTime+1);

AboutMotionMonitor();

Opens 'About MotionMonitor' dialog. In the following example, the Help| About MotionMonitor dialog is opened. This script function should be followed by a semi-colon ";".

AboutMotionMonitor();

Script Analysis Functions

The following list is of script Analysis functions. Script Analysis functions relate to processes involving Activities or the post processing of data. Each script Analysis function is immediately followed by a semi-colon ";". When an argument is optional, it will appear in brackets "[ ]". Each script Analysis function will be followed by an example demonstrating its use within a script. These examples are intended to demonstrate how to implement each script function. They may have been taken from a larger script and may not be fully functional on their own.

SetRepair(Var, Enabled, MaxInterval);

Enables a data repair function for the selected hardware. The first argument is the data to be interpolated. The second argument is a Boolean for the enabled status and the third argument sets the maximum frame interval for which the data will be interpolated. In the following example, the interpolation for missing Vicon Marker data was set to True with a maximum frame interval of 0.01 seconds.

SetRepair(Vicon1.Markers, TRUE, 0.01);

SetButterworthFilter(Var, Enabled, Freq);

Enables a Butterworth 4th order zero phase shift filter for the selected hardware, Var. The second argument is a Boolean for the enabled status. The final argument sets the cut-off frequency for the filter in Hz. In the following example, the Butterworth filter for Vicon Marker data was set to True, and the cutoff frequency was set to 20 Hz. This script function should be followed by a semi-colon ";".

SetButterworthFilter(Vicon1.Markers, TRUE, 20);

SetChebyshevFilter(Var, Enabled, Freq);

Enables a Chebyshev 4th order zero phase shift filter for the selected hardware. The second argument is a Boolean for the enabled status. The final argument sets the cut-off frequency for the filter in Hz. In the following example, the Chebyshev filter for Vicon Marker data was set to True, and the cutoff frequency was set to 20 Hz. This script function should be followed by a semi-colon ";".

SetChebyshevFilter(Vicon1.Markers, TRUE, 20);

SetFFTLowpassFilter(Var, Enabled, Freq, Rolloff);

Enables an FFT low pass filter for the selected hardware. The first argument is the hardware to be filtered. The second argument is a Boolean for the enabled status. The third argument sets the cut-off frequency for the filter in Hz. The final argument sets the width of the transition band (Rolloff) in Hz. In the following example, the FFT Lowpass filter for Measurement Computing A/D data was set to True, and the cutoff frequency was set to 400 Hz with a 2Hz rolloff. This script function should be followed by a semi-colon ";".

SetFFTLowpassFilter(MCCDAQ2, TRUE, 400, 2);

SetFFTHighpassFilter(Var, Enabled, Freq, Rolloff);

Enables an FFT high pass filter for the selected hardware. The first argument is the hardware to be filtered. The second argument is a Boolean for the enabled status. The third argument sets the cut-off frequency for the filter in Hz. The final argument sets the width of the transition band (Rolloff) in Hz. In the following example, the FFT Highpass filter for Measurement Computing A/D data was set to True, and the cutoff frequency was set to 20 Hz with a 2Hz rolloff. This script function should be followed by a semi-colon ";".

SetFFTHighpassFilter(MCCDAQ2, TRUE, 20, 2);

SetFFTNotchFilter(Var, Index, Enabled, Freq, Width, Rolloff);

Sets a notch filter in recorded activities. The first argument is the hardware to be filtered. The second argument refers to the notch filter number. The third argument is a Boolean for the enabled status. The fourth argument is the frequency in Hz to be filtered. The fifth argument sets the width of the notch filter in Hz. The final argument sets the width of the transition band (Rolloff) in Hz. The Notch filter index number must have already been added through the components window. In the following example, the Measurement Computing A/D FFT notch filter #0 was set to True, and the cutoff frequency was set to 60 Hz with a 5Hz width and 2Hz rolloff. This script function should be followed by a semi-colon ";".

SetFFTNotchFilter(MCCDAQ2, 0, TRUE, 60, 5, 2);

EditFilterSettings(Var);

Opens the Edit Filter Settings dialog for a specified hardware data. In the following example, the Filter settings dialog for the Measurement Computing MCCDAQ2 device, channel 0 will open, providing the opportunity to manually update all the filter settings for this hardware data. This script function should be followed by a semi-colon ";".

EditFilterSettings(MCCDAQ2.Channel0);

ShowGraph(Name);

Displays the graph with graph name, Name, surrounded by quotation marks. In the following example, the Graph named 'RThigh' would open. This script function should be followed by a semi-colon ";".

ShowGraph("RThigh");

HideGraph(Name);

Hides the graph with graph name, Name, surrounded by quotation marks. In the following example, the Graph named 'RThigh' would close but is not deleted. This script function should be followed by a semi-colon ";".

HideGraph("RThigh");

RenameGraph(OldName, NewName);

Renames a graph. The first argument is the previous name given to the graph. The second argument is what it'll be renamed as. All arguments should be surrounded by quotation marks. In the following example, the Graph named 'Channel 0' would be renamed as 'GastrocEMG'. This script function should be followed by a semi-colon ";".

RenameGraph("Channel 0", "GastrocEMG");

RenameGraphPlot(GraphName, OldPlotName, NewPlotName);

Renames a plot in a specified graph. The first argument is the graph that contains the plot to be renamed. The second argument is the previous plot name. The final argument is what the plot will be renamed as. All arguments should be surrounded by quotation marks. In the following example, the plot named 'Channel 0' in the graph 'GastrocEMG' was renamed 'Raw'. This script function should be followed by a semi-colon ";".

RenameGraphPlot("GastrocEMG", "Channel 0 ", "Raw");

ShowAnimation(Name);

Displays the Animation with animation name, Name, surrounded by quotation marks. In the following example, the Animation window named 'Animation' would open. This script function should be followed by a semi-colon ";".

ShowAnimation("Animation");

HideAnimation(Name);

Hides the Animation with animation name, Name, surrounded by quotation marks. In the following example, the Animation window named 'Animation' would close but is not deleted. This script function should be followed by a semi-colon ";".

HideAnimation("Animation");

ShowMusculoskeletalAnimation(Name);

Displays the Musculoskeletal Animation with animation name, Name, surrounded by quotation marks. In the following example, the Musculoskeletal Animation window named 'MusculoskeletalAnimation1' would open. This script function should be followed by a semi-colon ";".

ShowMusculoskeletalAnimation("MusculoskeletalAnimation1");

HideMuscusloskeletalAnimation(Name);

Hides the Musculoskeletal Animation with animation name, Name, surrounded by quotation marks. In the following example, the Musculoskeletal Animation window named 'MusculoskeletalAnimation1' would close but is not deleted. This script function should be followed by a semi-colon ";".

HideMusculoskeletalAnimation("MusculoskeletalAnimation1");

ShowVideo(Name);

Displays the video with camera name, Name, surrounded by quotation marks. In the following example, the Video window for 'Camera1' would open. This script function should be followed by a semi-colon ";".

ShowVideo("Camera1");

HideVideo(Name);

Hides the video with camera name, Name, surrounded by quotation marks. In the following example, the Video window for 'Camera1' would close, but is not deleted. This script function should be followed by a semi-colon ";".

HideVideo("Camera1");

GenerateReport(Name[, AnalysisName]);

Generates a Report. The first argument is the report name to be generated. The optional second argument is the analysis file used to generate the report. This is the analysis file that contains the report component, not an analysis file that gets applied to the activities. All arguments should be surrounded by quotation marks. In the following example, Report1 was used to generate a report and the parameters for the report were included in the specified analysis file. This script function should be followed by a semi-colon ";".

GenerateReport("Report1",
"C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\Shared\Analyses\ReportAnalysis.ian");

GenerateReportAs(Name, FileName[, AnalysisName]);

Generates a Report with a specified name. The first argument is the report name to be generated. The second argument saves the report as the specified name in this field. The optional final argument is the analysis file used to generate the report. This is the analysis file that contains the report component, not an analysis file that gets applied to the activities. All arguments should be surrounded by quotation marks. In the following example, Report1 was used to generate a report that was named saved as Generated_Report and the parameters for the report were included in the specified analysis file. This script function should be followed by a semi-colon ";".

GenerateReportAs("Report1", "Generated_Report", "C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\Shared\Analyses\ReportAnalysis.ian");

GenerateReportIn(Name, FilePath[, AnalysisName]);

Generates a Report with a specified name. The first argument is the report name to be generated. The second argument is the desired destination file path, and the optional final argument is the analysis file used to generate the report. This is the analysis file that contains the report component, not an analysis file that gets applied to the activities. All arguments should be surrounded by quotation marks. In the following example, Report1 was used to generate a report that was saved to the specified file path. No optional analysis file was selected, so the parameters used for the report were contained within the Activity or live workspace where the script function was performed from. This script function should be followed by a semi-colon ";".

GenerateReportIn("Report1","C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\Shared\Export");

SendReportToSmartabase(Name[, AnalysisName]);

Generates Report and sends it to Smartabase using the Parameters specified within the "Smartabase Settings" dialog for the Report. The first argument is the report name to be generated. The optional second argument is the analysis file used to generate the report. This is the analysis file that contains the report component, not an analysis file that gets applied to the activities. All arguments should be surrounded by quotation marks. In the following example, Report1 was used to generate a report for sending data to Smartabase. No optional analysis file was selected, so the parameters used for the report were contained within the Activity or live workspace where the script function was performed from. This script function should be followed by a semi-colon ";".

SendReportToSmartabase("Report1");

Spawn(ExePath[, Arg]);

Spawns an executable in a specified file path with an optional argument option. For example, this can be used to spawn Excel. All arguments should be surrounded by quotation marks. In the following example, Excel was launched from the specified file path. The optional Arg argument was not used, so no additional actions were performed. This script function should be followed by a semi-colon ";".

Spawn("C:\Program Files\Microsoft Office\root\Office16\EXCEL.EXE");

SpawnPythonScript(ScriptPath[, WorkingDir]);

Spawns a Python script. The first argument must be a complete path to the Python script you want to run. The optional second argument must be a complete path to the working directory that you want the script to use. If omitted, the same folder where the script resides will be used for the working directory. When called, the function will open a command prompt window in a minimized state and run the Python script inside of it. This window will remain active until the user closes it. All arguments should be surrounded by quotation marks. In the following example, the python script to be executed was specified and the same file path will be used for the working directory since no optional WorkingDir argument was provided. This script function should be followed by a semi-colon ";".

SpawnPythonScript("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\Python\python_script.py");

Excel(FilePath, FunctionName);

Opens the specified Excel workbook file, FilePath, in Excel and executes the Macro, FunctionName. All arguments should be surrounded by quotation marks. In the following example, the specified Excel workbook file was opened, followed by the execution of the specified excel macro. This script function should be followed by a semi-colon ";".

Excel("c:\\ReportGenerator\StandardReptGen.xlsm", "GenerateGolfReportMacro");

OpenAnalysis([Name]);

Opens a file-browsing dialog for selecting an Analysis File. If the optional Name argument is specified, the Analysis by the specified name will open. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. If no file path is included with the Analysis file, the Analysis file will be loaded from the current Users Analyses folder. In the following example, the specified Analysis file will be loaded from the specified file path. This script function should be followed by a semi-colon ";".

OpenAnalysis("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses\Gait Walk Data Reduction.ian");

OpenFilterExclusiveAnalysis([Name]);

Opens a file-browsing dialog for selecting an Analysis File that will be loaded without applying the filter settings from the Analysis file. If the optional Name argument is specified, the Analysis by the specified name will open. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. If no file path is included with the Analysis file, the Analysis file will be loaded from the current Users Analyses folder. In the following example, the specified Analysis file will be loaded without its filter settings from the specified file path. This script function should be followed by a semi-colon ";".

OpenFilterExclusiveAnalysis("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses\Gait Walk Data Reduction.ian");

OpenAnalysisInActivity(ActivityName[, AnalysisName]);

Opens an Analysis file in an open Activity. This function is performed from the live workspace. The argument ActivityName must be the name of an Activity that is already open. AnalysisName, is an optional argument for the name of an analysis file that can be applied to the Activity. If omitted, a file-browser dialog will open for the selection of an Analysis file. The Activity and Analysis arguments are accepted with or without an extension and all arguments must be surrounded by quotation marks. If no file path is included with the Analysis file, the Analysis file will be loaded from the current User Analyses folder. In the following example, the specified Analysis file will be loaded from the specified file path in the 'Gait Walk1' open Activity. This script function should be followed by a semi-colon ";".

OpenAnalysisInActivity("Gait Walk1", "C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses\Gait Walk Data Reduction.ian");

MergeAnalysis([Name]);

Merges analysis files. A name can be specified for the analysis file to merge, otherwise a file-browser dialog will open. If an analysis component is present in the existing workspace (or activity), but not in the analysis file, the component will be kept. If a component is present in the analysis file, but not the existing workspace, it will be loaded. If a component is present in both places, the version in the analysis file will replace the existing version in the workspace. The Analysis Name argument is accepted with or without an extension and must be surrounded by quotation marks. If no file path is included with the Analysis file, the Analysis file will be loaded from the current User Analyses folder. In the following example, the specified Analysis file will be loaded from the specified file path in the current Activity or workspace where the function was executed from. This script function should be followed by a semi-colon ";".

MergeAnalysis("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses\Gait Walk Data Reduction.ian");

MergeFilterExclusiveAnalysis([Name]);

Merges an Analysis File that will be loaded without applying the filter settings from the Analysis file. A name can be specified for the analysis file to merge, otherwise a file-browser dialog will open. If an analysis component is present in the existing workspace (or activity), but not in the analysis file, the component will be kept. If a component is present in the analysis file, but not the existing workspace, it will be loaded. If a component is present in both places, the version in the analysis file will replace the existing version in the workspace. The Analysis Name argument is accepted with or without an extension and must be surrounded by quotation marks. If no file path is included with the Analysis file, the Analysis file will be loaded from the current User Analyses folder. In the following example, the specified Analysis file will be loaded from the specified file path in the current Activity or workspace where the function was executed from. This script function should be followed by a semi-colon ";".

MergeFilterExclusiveAnalysis("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses\Gait Walk Data Reduction.ian");

SaveAnalysisAs();

Opens a file-browser dialog where the Analysis can be saved with a custom name in a desired file location. In the following example, a file-browser dialog is opened for saving an Analysis file. This script function should be followed by a semi-colon ";".

SaveAnalysisAs();

OpenWindowLayout([Name]);

Analogous to the OpenAnalysis() function, except that the only parts of the Analysis that are applied are those pertaining to the window formatting, also including the default display time and Animation viewpoint and light settings. Opens a file-browsing dialog for selecting an Analysis File. If the optional Name argument is specified, the Analysis by the specified name will open. The Name argument is accepted with or without an extension and all arguments must be surrounded by quotation marks. If no file path is included with the Analysis file, the Analysis file will be loaded from the current Users Analyses folder. In the following example, the specified Analysis file will be loaded from the specified file path. This script function should be followed by a semi-colon ";".

OpenWindowLayout("C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\Sample Scholastic Files\Analyses\Gait Walk Data Reduction.ian");

SaveWindowLayoutAs();

Analogous to the SaveAnalysis() function, except that it only saves parts of the analysis pertaining to the window formatting, also including the default display time and Animation viewpoint and light settings. Opens a file-browser dialog where the Analysis can be saved with a custom name in a desired file location. In the following example, a file-browser dialog for saving the window layout. This script function should be followed by a semi-colon ";".

SaveWindowLayoutAs();

DataReduction([AnalysisName[, FolderPath[, FilenameFilter, LastNFiles, OutputFilename]]]);

Opens 'Data Reduction Parameters' dialog. The optional argument, AnalysisName, points to an analysis file that has the Data Reduction Parameters dialog settings, not an analysis file that gets applied to the activities. The AnalysisName argument is accepted with or without an extension. The optional argument, FolderPath, is the folder where the source activities are located. FilenameFilter is the filter to apply when scanning for source files, LastNFiles is the maximum number of files to include, and OutputFilename is the name to give to the data reduction Activity file. Any of these parameters will be skipped or use their default values if you pass a zero for LastNFiles or an empty string for the other arguments. The FilenameFilter argument can take multiple wildcard (*) characters, including in the leading position. For example, "*S01*.iac" will find any activity with "S01" anywhere in the name. The LastNFiles will look for the most recent files that match the FolderPath and FilenameFilter arguments. All arguments besides LastNFiles must be surrounded by quotation marks. In the following example, data reduction is performed and the analysis file that contains the parameters for the data reduction was specified. None of the other optional arguments were included. This script function should be followed by a semi-colon ";".

DataReduction("C:/ProgramData/Innsport/TMM_xGen/MotionMonitor/User/Sample Scholastic Files/Analyses/Gait Walk Data Reduction");

DataReductionFromOpenActivities([AnalysisName[, OutputFilename]]);

Performs Data Reduction for the Activities that are currently open. Both arguments are optional. AnalysisName points to an analysis file that has the Data Reduction Parameters dialog settings, not an analysis file that gets applied to the activities. OutputFilename is the name to give to the data reduction Activity file. The Analysis Name argument is accepted with or without an extension and each argument must be surrounded by quotation marks. Any of these parameters will be skipped or use their default values if you pass an empty string. In the following example, data reduction is performed on the open Activities and neither of the optional arguments were included. This script function should be followed by a semi-colon ";".

DataReductionFromOpenActivities();

ExtractTrials([AnalysisName[, FolderPath[, FilenameFilter, LastNFiles, OutputFilnameSuffix]]]);

Opens 'Trial Extraction Parameters' dialog. The optional argument, AnalysisName points to an analysis file that has the Extract Trials Parameters dialog settings, not an analysis file that gets applied to the activities. The AnalysisName argument is accepted with or without an extension. The optional argument, FolderPath, is the folder where the source activities are located. FilenameFilter is the filter to apply when scanning for source files, LastNFiles is the maximum number of files to include, and OutputFilenameSuffix is the name to give to the extracted Activity files. If used, this string will be appended to the end of the source activity's filename. The resulting filename will in turn have the individual trial numbers appended to it, to arrive at the trial activities' filenames. Any of these parameters will be skipped or use their default values if you pass a zero for LastNFiles or an empty string for the other arguments. The FilenameFilter argument can take multiple wildcard (*) characters, including in the leading position. For example, "*S01*.iac" will find any activity with "S01" anywhere in the name. The LastNFiles will look for the most recent files that match the FolderPath and FilenameFilter arguments. All arguments besides LastNFiles must be surrounded by quotation marks. In the following example, extract trials is performed and none of the optional arguments were included. This script function should be followed by a semi-colon ";".

ExtractTrials();