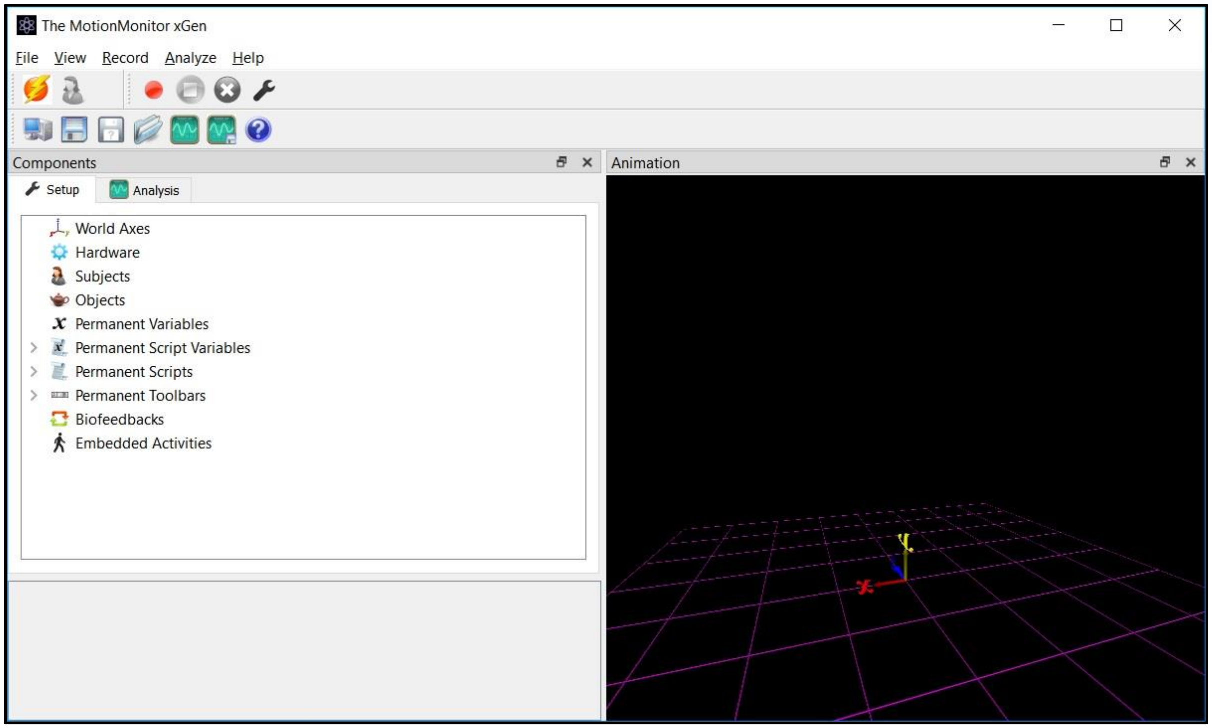
The MotionMonitor xGen Elements

The MotionMonitor xGen is an elegant and powerful set of tools for the acquisition, analysis, and visualization of biomechanical data. It consists of a few basic elements that once understood can be used to collect research data using a variety of protocols or to create diverse applications that range from the simple to the very complex. These elements consist of Setup and Analysis Components that can be viewed from the selection of the appropriate tab as shown in the image below.

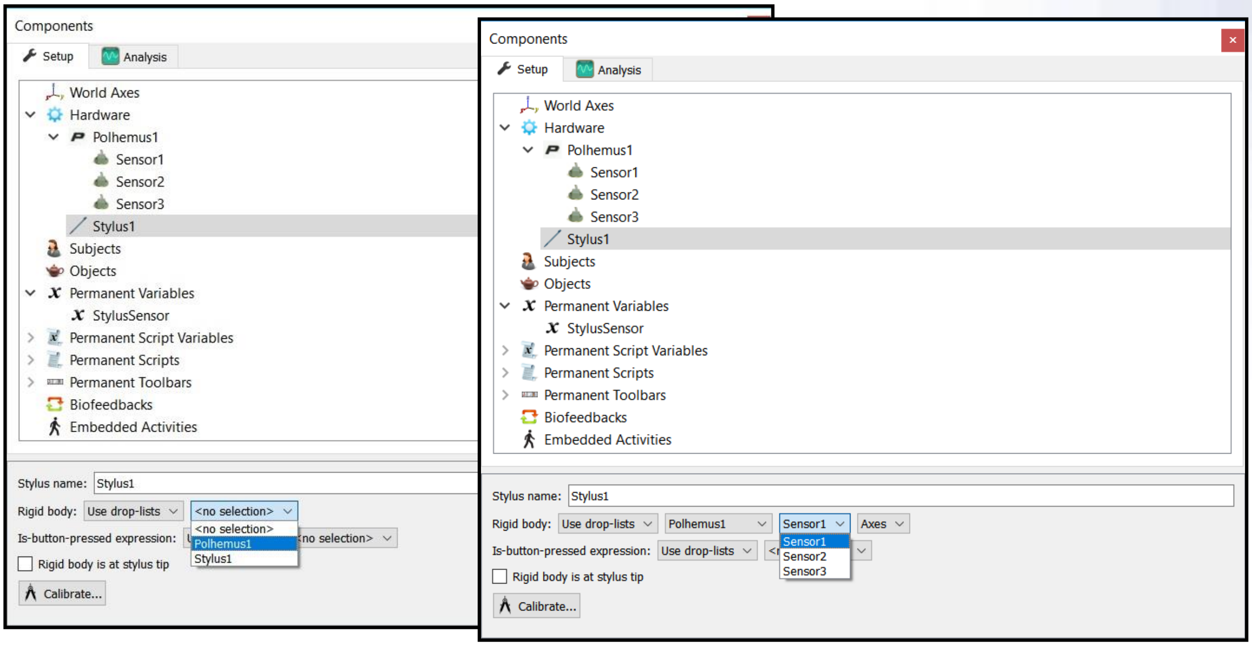


Before discussing these components in detail, it will be helpful to discuss data and how it is represented within the software. The MotionMonitor xGen derives much of its power and flexibility from the use of data variables, expressions and formulas to configure the software. It is arguably the most important aspect of the software. It will make configuration much easier if the structure of data is understood. A few minutes spent on the next 5 pages will accelerate your understanding of the next generation of motion capture software!

Data Representation

Data is used in many places within the software. For example, data can be used to determine how objects will move in the animation window during biofeedback exercises or how the collection of activities will be started or ended or how graphs will be populated. Typically, when data is being requested, the software will display a drop menu that indicates "Use Drop List" or "Use Formula" or "Use Expression". In the parameter panel at the bottom of the images below, "Rigid body" data is being requested. Ignore what the software is trying to accomplish, we'll discuss that later. For now, focus only on the fact that "Rigid body" data is being requested. The drop-list method is the easiest way to determine what Rigid body data (6 degree of freedom or position and orientation data) exists within the system. In this example, the only data that exists is the Polhemus1 sensor data and the Stylus1 that is being defined by the selection. If data is hierarchical ie. the selection contains more than one of the desired data types, additional fields will automatically display. In this case, the

Polhemus1 system had 3 sensors that each provide 6 degrees of freedom data. And when selected, the drop list was extended to permit selection of the desired sensor.



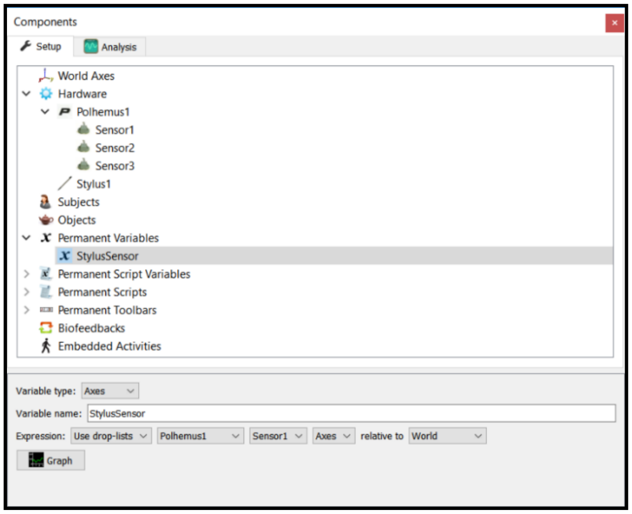
In the example above, the "Drop-lists" method was used to identify **existing** data that the system could use to satisfy its request. The "Drop-lists" method can also be used to **create new** variables. Let us consider several examples.

First, understand that data can take many forms within The MotionMonitor xGen including:

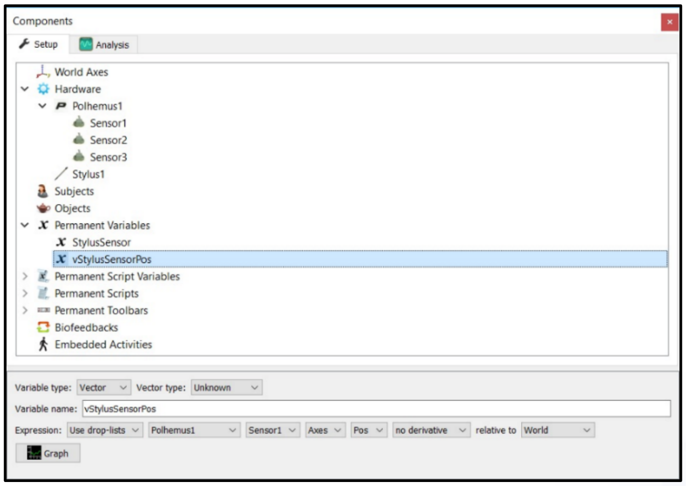
- Boolean Types**,** which can take a value of TRUE, FALSE or INVALID.
- Integer Types, which are whole numbers.
- Scalar Types*,* which are real numbers.
- Vector Types, which represent a 3 dimensional position in space or any quantity that possess a magnitude and direction, are characterized by an x, y and z value as in vec(x,y,z) Each of the x,y,z components is a scalar expression.
- Rotation Types, which are measures of orientation and can take the form of Euler angles, Quaternions or Cosine Matrices. Each of the components making up a rotation is a scalar expression.
  .
- Axes Types, which are essentially rigid bodies and have a representation of both position and orientation (6DOF) as in: axes(vec(x,y,z), rot(Z,Y,X)) where x,y,z and Z,Y,X are scalar expressions.
- Time Types, which are time stamps of the form 7-10-2011 14:16:53:848. When used in a calculation, time types return time in seconds. A time variable, time(ps,sec,min,hour,day,mon,year), is composed of Timestamp.PartSec, Timestamp.Sec, Timestamp.Min, Timestamp.Hour, Timestamp.Day, Timestamp.Mon, and Timestamp.Year. Where PartSec is Scalar and other fields are Integer values.

- String Types, which are groupings of alphanumeric characters. Within a formula, the characters are enclosed within quotation marks.
- Color Types, of the form: color(red,green,blue) where the arguments are scalar expressions.

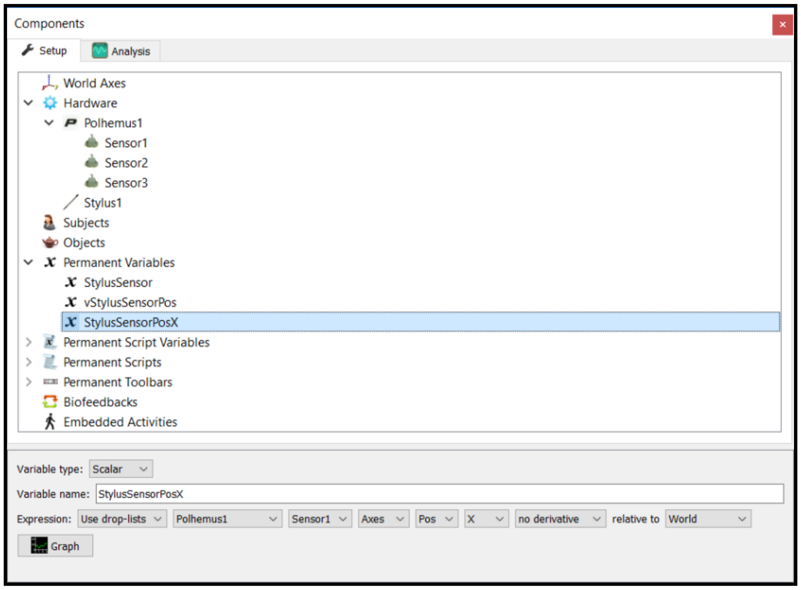
In the example below, focus only on the parameter panel for StylusSensor. We'll discuss Permanent Variables later. For now, we want to focus only on the creation of the variable. In this case "StylusSensor" is being defined as an axes or rigid body variable that contains 6DOF data. It was created by using the "Use Drop-lists" method to select Polhemus1 Sensor1 axes data. This is identical to our previous example but now we have a new variable "StylusSensor" that reports the same data as Polhemus1 Sensor1 axes data. The "relative to" field provides an easy method for transforming data to other coordinate systems (Axes).

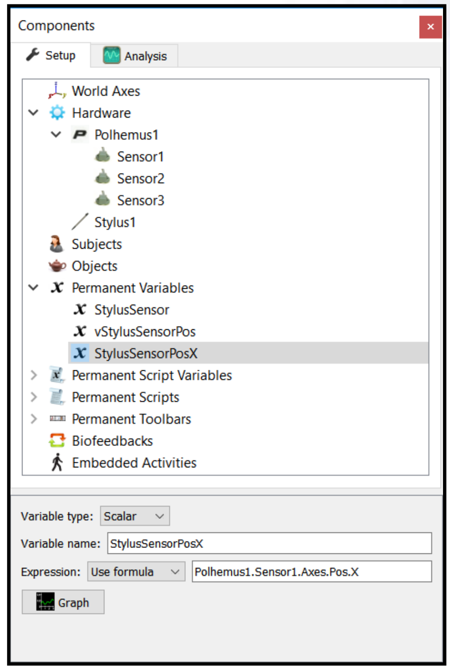


Because Polhemus.Sensor1 is an "axes" variable we know it contains both position and orientation data. In the image below, we have defined a vector variable named "vStylusSensorPos". Selecting Polhemus1 from the drop-list automatically adds fields that will result in the selection of the variable's vector components. In this case "Pos" is the vector representation of its position. The system recognizes the data is hierarchical and automatically adds the vector representation to the expression. If "Use drop-list" is changed to "Use formula", the internal form of the vector variable will be displayed. In this case "Polhemus1.Sensor1.Axes.Pos". Note that vector data types have a derivative drop-list. This provides an easy method for calculating the first and second derivatives for a variable.

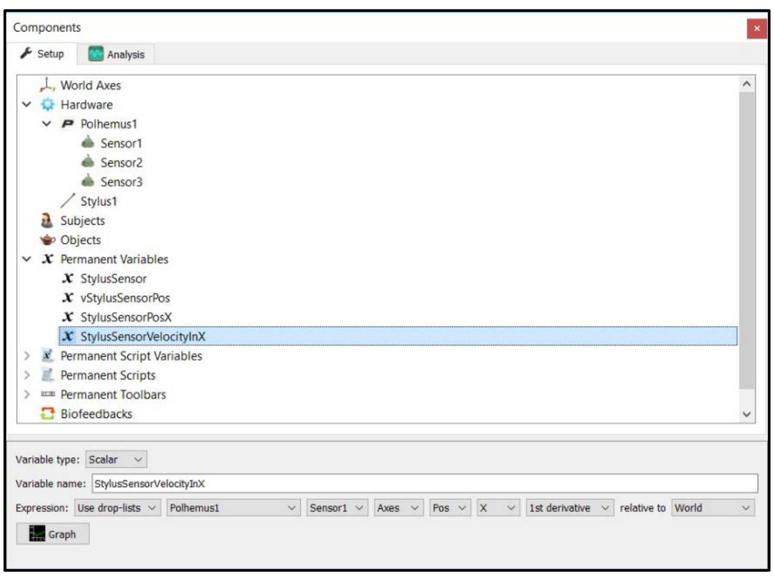


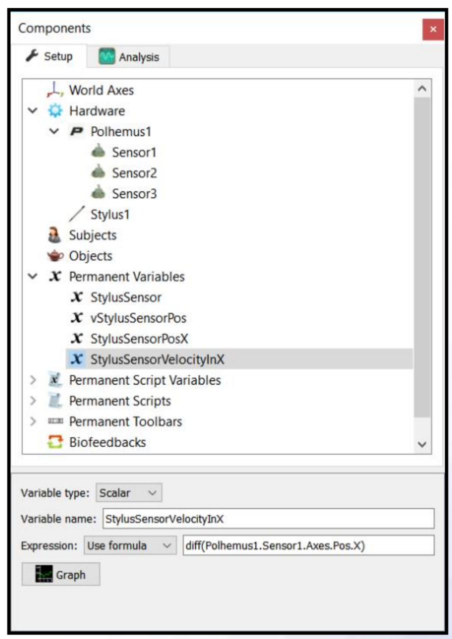
Continuing this example, we know that a vector variable contains x,y,z scalar components. Adding a scalar variable named StylusSensorPosX and from the drop-list selecting Polhemus1, we see that the expression now contains a field labeled “Pos” and a drop-list field with choice of x, y and z components or the magnitude of the position. The system automatically recognized the data as having scalar components and offered them in the drop list. Switching from drop-list to formula we can observe the internal format as “Polhemus1.Sensor1.Axes.Pos.X”.



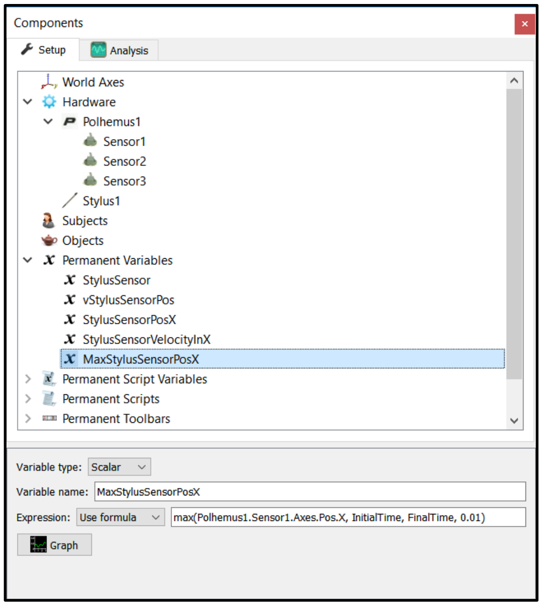


Note that scalar data types also have a derivative drop-list. As with vector data types, this provides an easy method for calculating the first and second derivatives for these variable types. Below, the first derivative of the position has been selected, thus defining velocity of the StylusSensor in the X direction of the World coordinate system. Switching from drop-list to formula we can observe the internal format as “diff(Polhemus1.Sensor1.Axes.Pos.X)”.

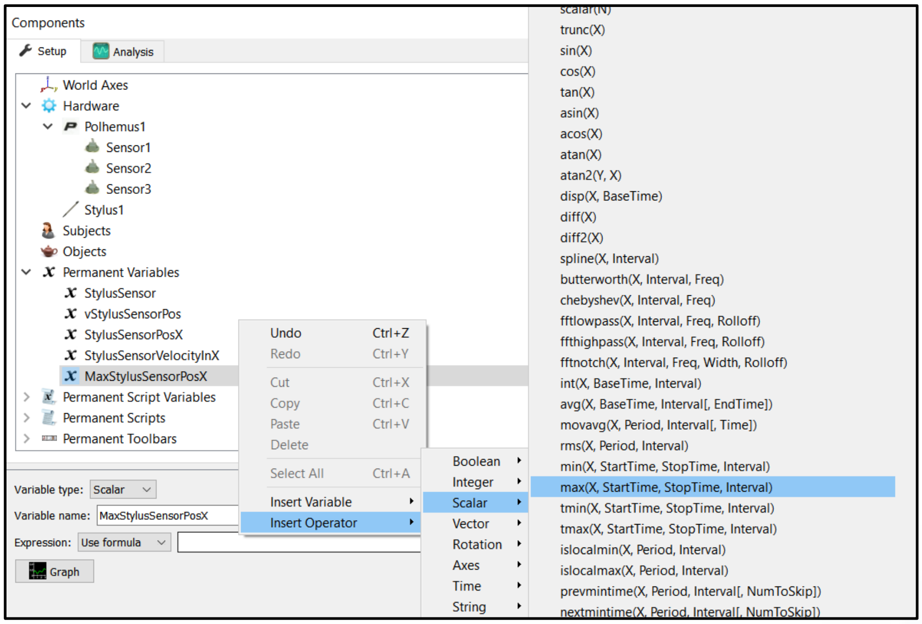




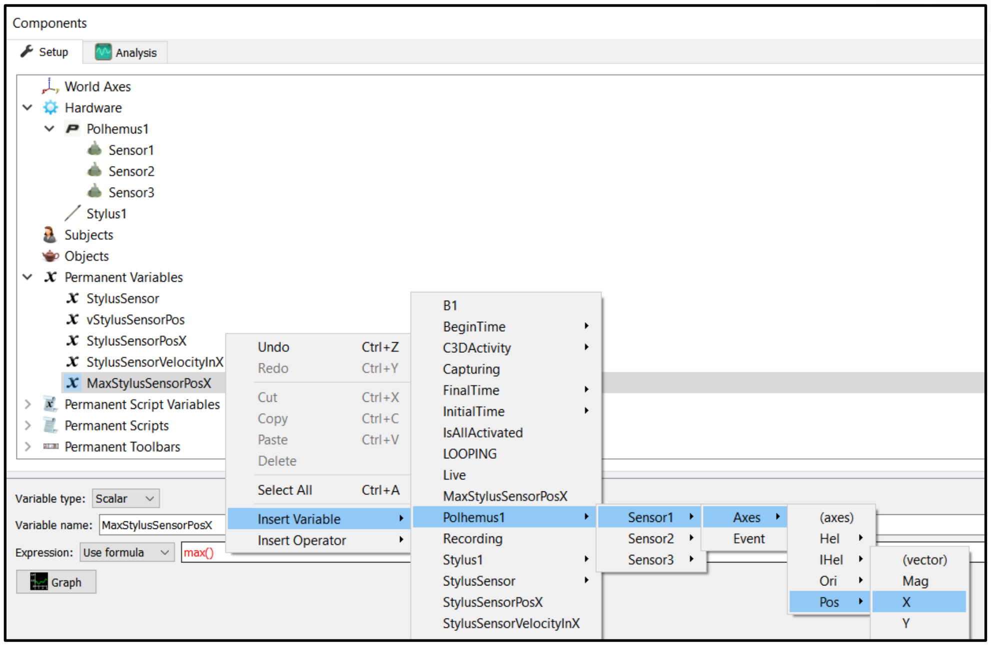
While use of the drop-list method provides a fast method for defining data, the formula method adds real power to the process. In the image below, the formula method is selected. Formulas can be typed directly into the formula edit field as shown below. In this case the operator "max()" has been entered which computes the maximum value for a variable between the specified start and stop time with readings evaluated at the specified interval.



Alternatively, operators and variables can be selected from cascading dialogs. Right clicking in the edit field provides a menu that is used to insert variables and operators. The operators are grouped by type. Selecting max(X, StartTime, StopTime, Interval) results in the code for that operator being placed in the edit field as shown below.



Right clicking again with the cursor within the max() operator parentheses, and inserting the variable Polhemus1.Sensor1.Axes.Pos.X, a Start Time, End Time, and interval, separated by commas, will complete the definition.



If, after entering, the text in the edit field remains red, there is an error which can be determined by hovering over the edit field. Operators follow standard order of operation and can be nested. However, caution should be exercised when nesting computationally intensive operators such as derivatives and integrals or multiple “if” statements as the number of computations can expand exponentially.

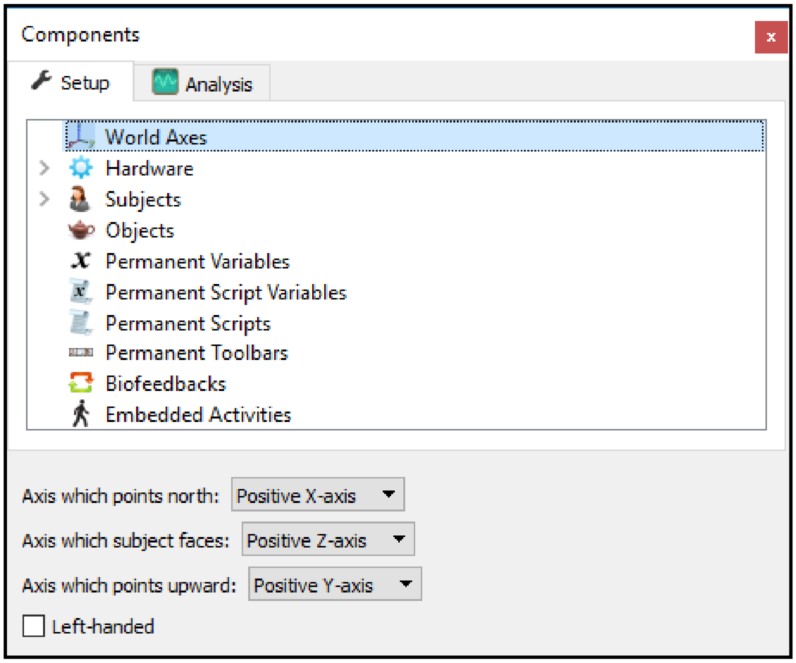
With this understanding of drop-list and formula data, we can now proceed to the components that make up The MotionMonitor xGen software.

Setup Components

Setup Components is the means by which data are generated. Setup Components include the definition of the world axes, selection of data-collection hardware, rigid bodies, styluses, objects, permanent variables and subjects. Components can be used in almost any quantity and combination.

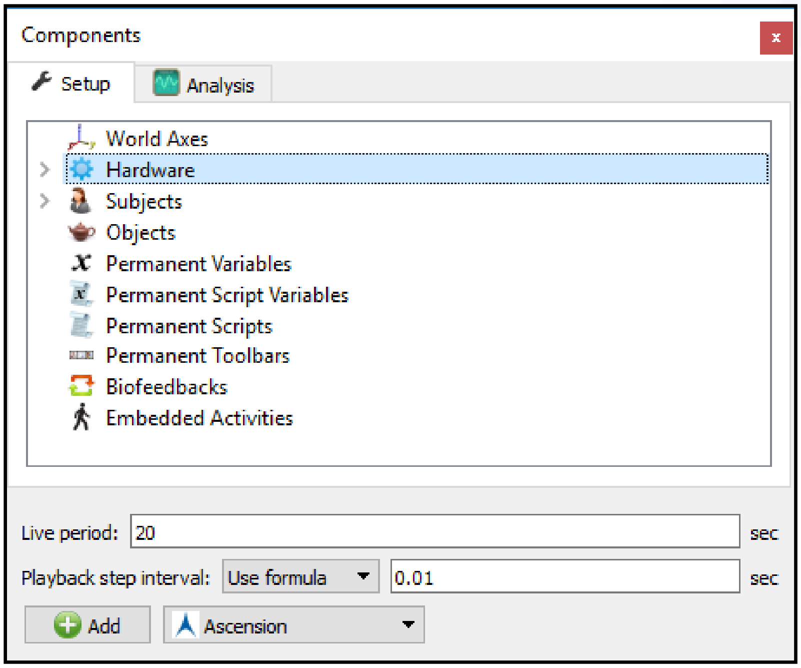
World Axes

Selecting the "World Axes" in Components Setup tab brings up the parameters panel for defining the world axes. These include the axis which points north (only required for Inertial Measurement Units or "IMU's"), the axis which the subject faces during setup, and the axis which points upward. The default world axes are right-handed but marking the check box will generate a left-handed coordinate system. Note that "axis which the subject faces" refers only to the time of setup. During data collection, the subject can move in any direction.



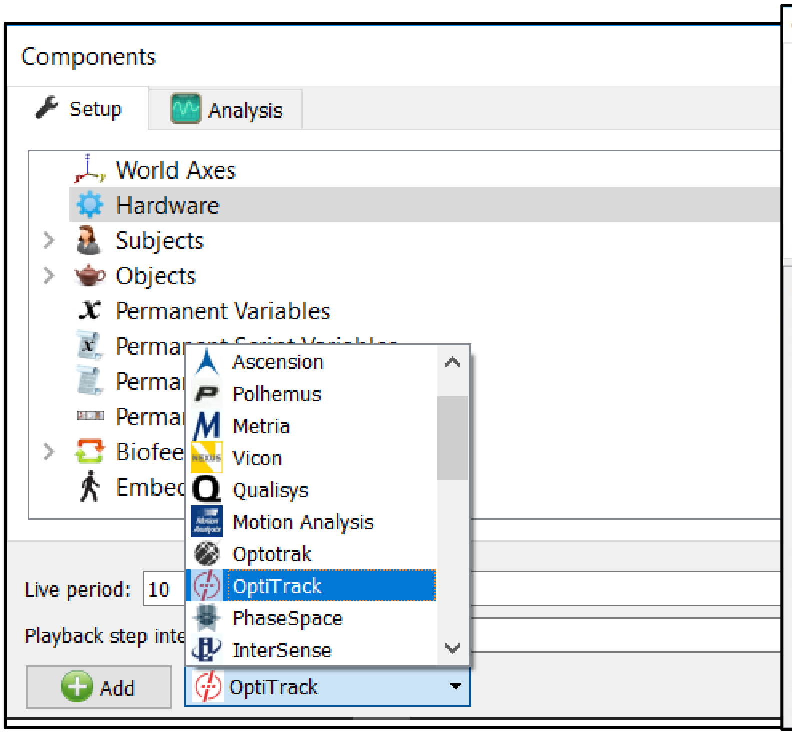
Hardware

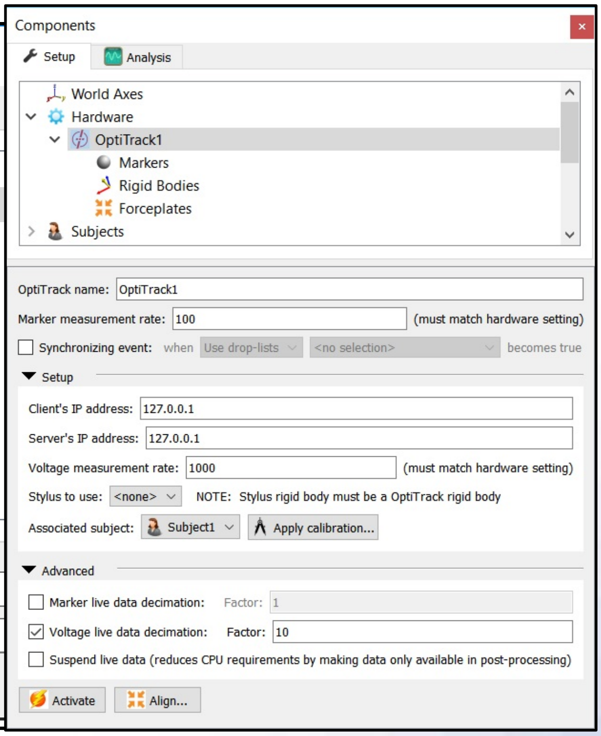
Selecting "Hardware" in the Components Setup allows the user to choose hardware devices. The "Live Period" on the hardware parameters panel specifies the maximum amount of time retained in the working memory for performing calculations displayed in the Live window. The live period has no effect on the maximum recording duration. However, a longer live period will use up more computer resources. The "playback step interval" simply establishes the period between readings displayed in the slider bar. It is often used to match the playback to the speed of video.



Clicking the "Add" button will add the hardware device selected in the drop-list to the hardware node. In the example below an OptiTrack system was selected from the drop-list. Selecting the specific hardware component will bring up its parameters panel. Your client support engineer can help identify correct settings for each hardware type.

The hardware device parameters panel is organized into multiple sections. The area at the top and buttons at the bottom are more commonly needed, and therefore always visible. The Setup and Advanced sections are typically updated less frequently and can be expanded or collapsed as needed.





The hardware name is what defines how the hardware is referenced internally and when referencing data from the hardware device when using the variable drop-lists or formula right-click cascading menus. If the hardware device name is modified, the update will propagate throughout the software, expect within scripts and previously recorded activities. One note of caution is that if you are changing the name, separate analyses would be needed for each different hardware name, possibly making post processing and batch processing more difficult.

If a new alignment of the hardware coordinate system will be performed using the “Align” button, then the “Stylus to use” must be defined using this hardware’s data for digitizing new world axes points. The use of Associated subjects is unique to OptiTrack and should be discussed with your client support engineer.

The data measurement used in live calculations and for updating graphics can be scaled down by enabling the “live data decimation” checkbox. This will result in an effective measurement rate for real time purposes being equal to the hardware’s measurement rate divided by the specified decimation factor. Similarly, the display of data in the Live window can be suspended by placing a mark in the checkbox for “Suspend live data”. The processing and display of data in a live mode uses computer resources, so these options allow the user to optimize computer resources and free up more processing power for data collection. These settings are independent for each hardware device and once an activity has been recorded data from the full measurement rate will be accessible for analysis.

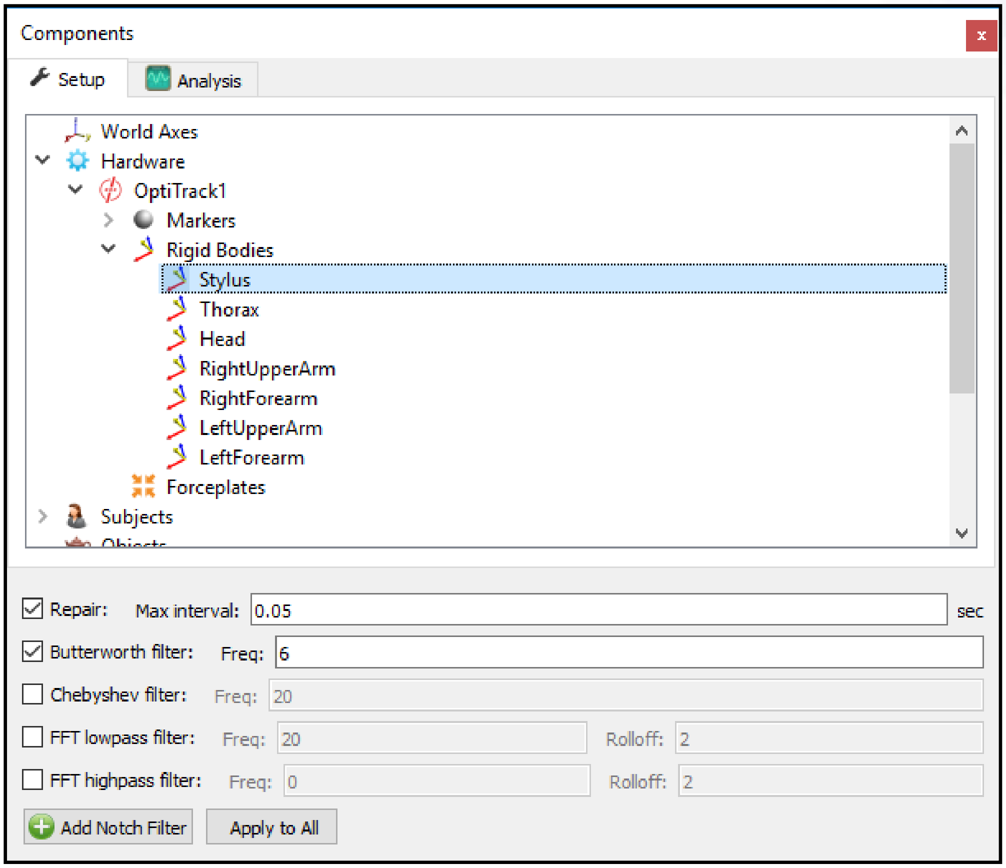
If a synchronizing device such as a handheld event marker is to be used in place of passive synchronization methods that are implemented by The MotionMonitor xGen by default, mark the “Synchronizing event:” check box and provide a definition of the event marker as a Boolean condition. The definition must be based on the hardware component being added. For example, if considering an A/D device with a handheld event marker’s signal input to A/D hardware, the synching event would occur when the A/D input channel either exceeded some voltage or fell below some voltage, depending on how the A/D hardware is configured. A signal from the OptiTrack hardware system could not be used as the synchronizing event for the A/D device. The synchronizing event specified for each hardware device will be used to override the default MotionMonitor xGen alignment for that hardware. In cases where a synchronization method is not implemented for a hardware device, the default synchronization method will still be used to align with other hardware data streams.

Hardware components can be activated by clicking the “Activate” button. This button will toggle between activate and deactivate depending on the state of the hardware. Finally, the coordinate system of hardware components can be aligned with other hardware or a user specified world co-ordinate system by clicking the “Align” button and following the screen prompts.

The data that is streamed from a hardware system will automatically appear below the hardware component. In the case of OptiTrack1, the hardware will stream i) the 3DOF vector position of “Markers”, ii) the axes data (vector position and orientation) of “Rigid Bodies”, and iii) if being collected via Optitrack, “Forceplates” center of pressure, force and moment data.

Regardless the data type, when the data is selected as shown below, a panel will appear with smoothing parameters. These can be set as appropriate for the data type and activity being collected. The smoothing parameters can be modified separately for each data type or parameters can be applied to all data within the node when the “Apply to All” button is clicked. For example, “Apply to All”, when smoothing a marker, will smooth all markers in that node, but not Rigid Bodies.

It should be noted that these parameters control smoothing at the time of collection. When a collected activity is being analyzed, these smoothing parameters can be modified as part of the analysis settings. The smoothing parameters are saved within an Analysis (*.ian) file.



Subjects

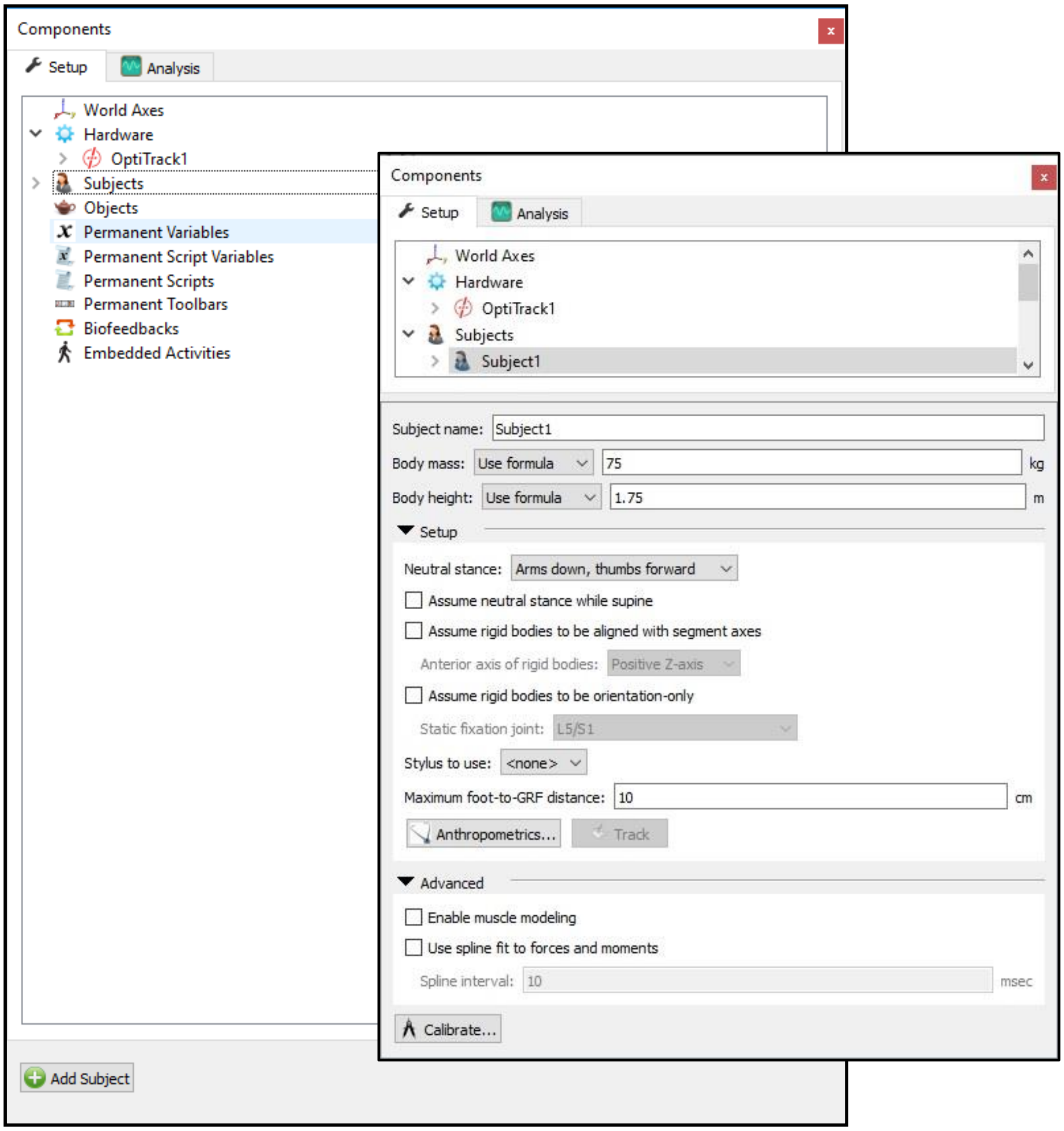
Selecting "Subjects" in the Components Setup tab allows the user to add one or more subjects to the workspace with the "Add Subject" button.

Selecting the added subject, Subject1 from the Subjects node reveals edit fields describing parameters that will be used during the calibration protocol. Most parameters are self-explanatory.

Similar to the hardware device, the Subject parameters panel is organized into multiple sections and the name can be modified. One note of caution is that if you are changing the name, separate analyses would be needed for each different subject name, possibly making post processing and batch processing more difficult.

The "Assume rigid bodies to be aligned with segment axes" checkbox can be used when the rigid body axes assigned to each segment are aligned with the longitudinal axis of the segment. Then, the "Anterior axis of rigid bodies", the axis that is directed anteriorly, can be selected from the drop-list. With these two axis directions known, the segment axes can be generated directly from the sensor data. This has more advantages during the C3D import process, because a neutral stance file would no longer be required, but it can also be applied to real time subject calibrations.

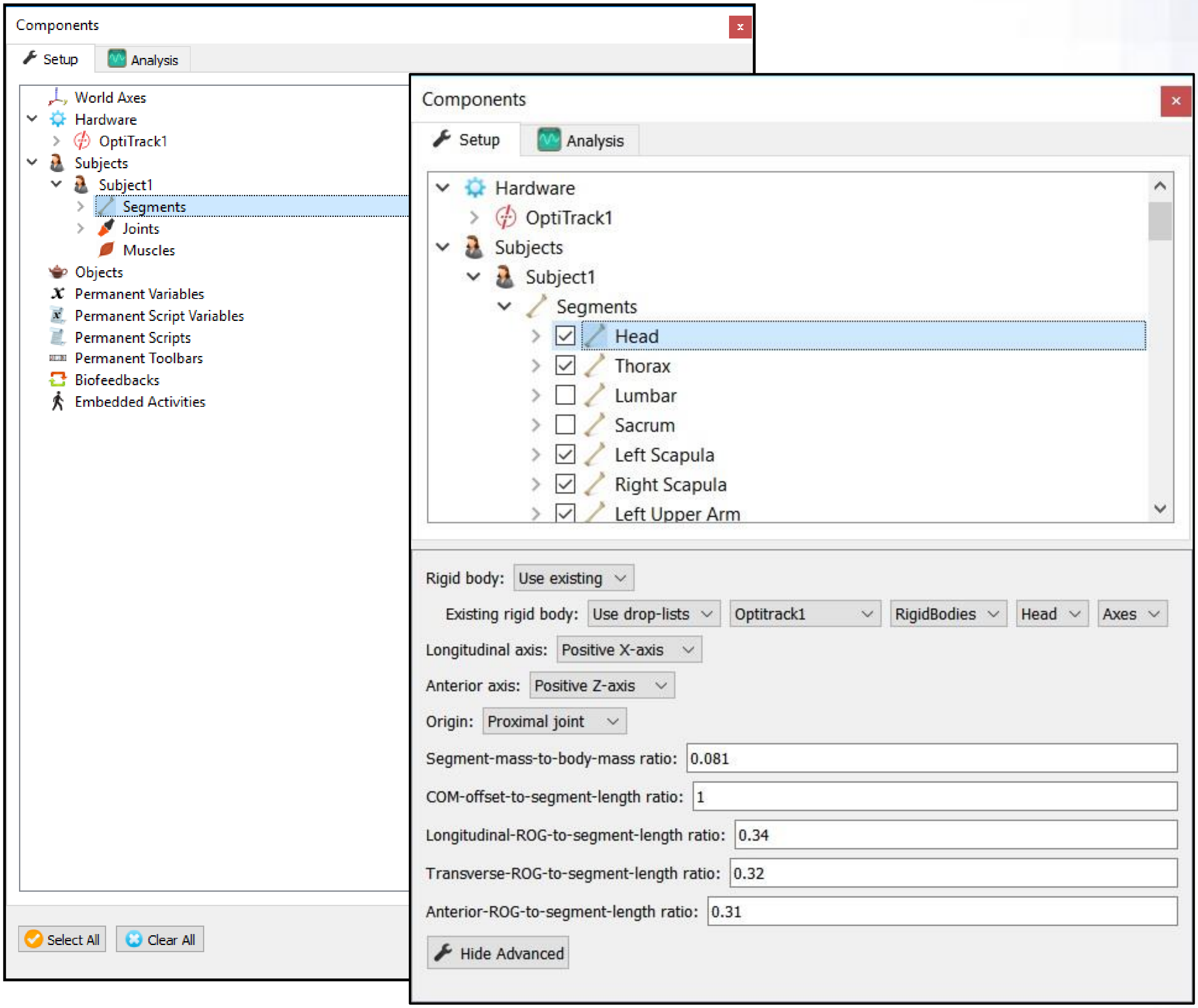
The "Assume rigid bodies to be orientation-only" checkbox can be used with sensors that provide only orientation data, such as Inertial Measurement Units. Here the subject's skeleton is built much like a rag doll. Each segment's proximal endpoint is positioned relative to its proximal segment's distal endpoint and oriented per the sensor's data. This method requires the identification of a "static fixation joint", a static starting point or fixed segment joint around which segment movement will occur.



The maximum foot-to-GRF separation field is specified to avoid spurious assignments of ground reaction forces to right and left feet. Right and left feet are assigned to forceplate ground reaction forces based on proximity and this setting assures they are within the specified distance before making the assignment.

The Use spline fit to forces and moments field is specified to restrict the inverse dynamics calculations for joint forces and moments to be performed at a specified measurement rate. The default measurement rate for these calculations is 1 msec. When enabled, this feature can reduce processing requirements in recorded activities for performing these calculations and free up computer resources.

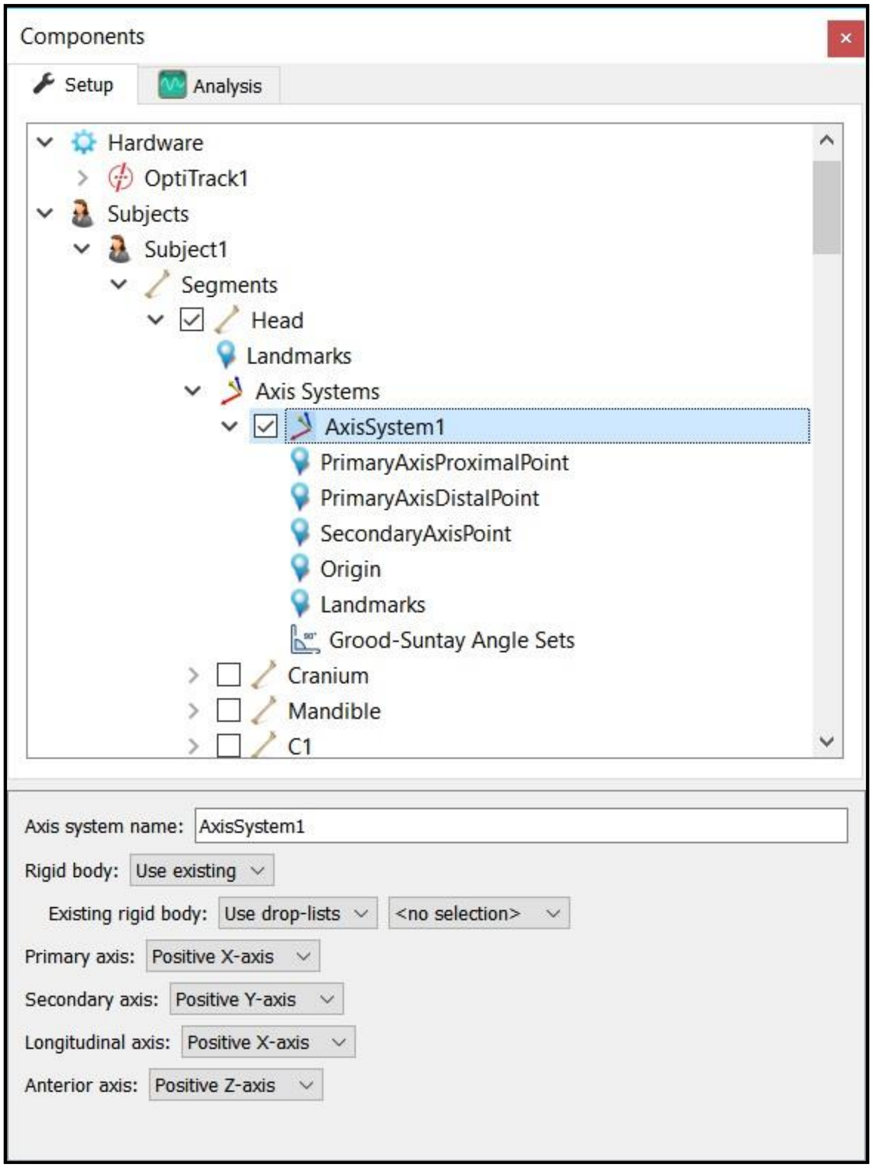
Expanding Subject1 displays the Segments, Joints and Muscles associated with that subject. Expanding the Segments node displays a list of bone segments. When a segment is enabled by placing a mark in the checkbox the Joints node is automatically populated with appropriate joints. Any number or combination of bone segments can be enabled.



The panel that appears at the bottom of each bone segment is used to associate a rigid body or an axes variable that will position and orient the segment. The “Longitudinal axes”, “Anterior axes” and “Origin” drop-lists control the layout for the segment Anatomical Axes. The anthropometrics for each segment may also be edited on this panel or from the Anthropometrics button in the Subject1 parameters panel.

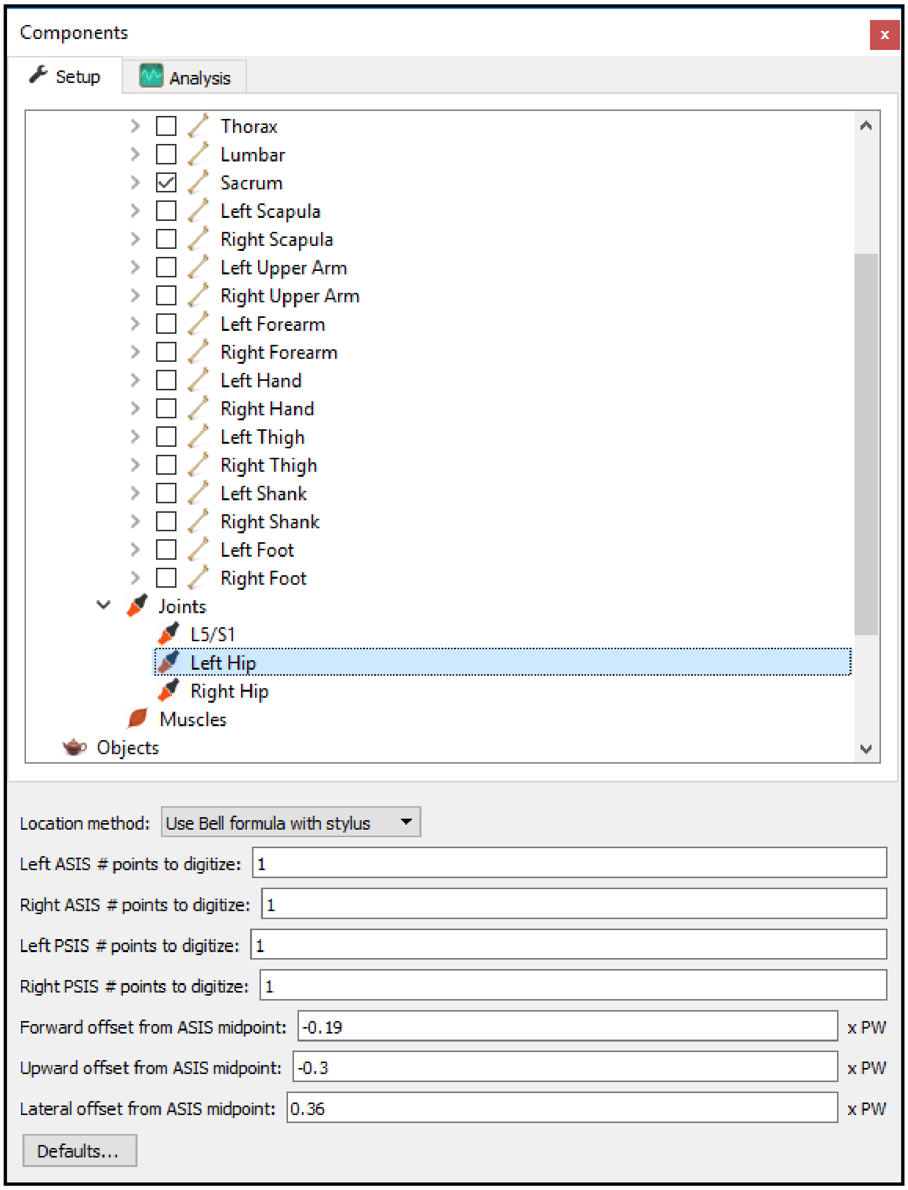
The parameter panel for the Left and Right hands have an additional option of selecting a “Force/Torque transducer assigned to this hand” from a drop-list. This allows for the calculation of top-down inverse dynamics and requires that a 6DoF force/torque device has been configured as a hardware component and then aligned.

Expanding a bone segment node reveals additional, optional parameters. Depending on the bone segment these optional parameters may be user specified axis systems or additional detail bones. Spine segments, hands and feet all have additional detail bones.



Axis systems can be added to define custom anatomically based segment coordinate system or to conform with standards in the literature, such as the ISB recommendations. Similar to with the bone segment, a valid rigid body or an axes variable that will position and orient the axes needs to be associated with the Axis system. Additionally, the four points used to define the Axis system must also have a valid definition. The "Primary axis" and "Secondary axis" drop-lists control the layout for the axes, based on the primary and secondary axis points. The "Longitudinal axis" and "Anterior Axis" drop-lists are only used with Grood-Suntay Angle Sets. For more details on additional axes systems, contact your client support engineer.

As mentioned previously, enabling a bone segment will automatically create the appropriate joints under the Joints node. In the example below, Sacrum has been enabled and it has generated L5/S1 and right and left hips as joints to be defined. Selecting the joint node will show the parameters panel with edit fields required for each type of joint. Essentially, these edit fields identify how the joint center will be located as well as any offsets that will be used to adjust the reading.

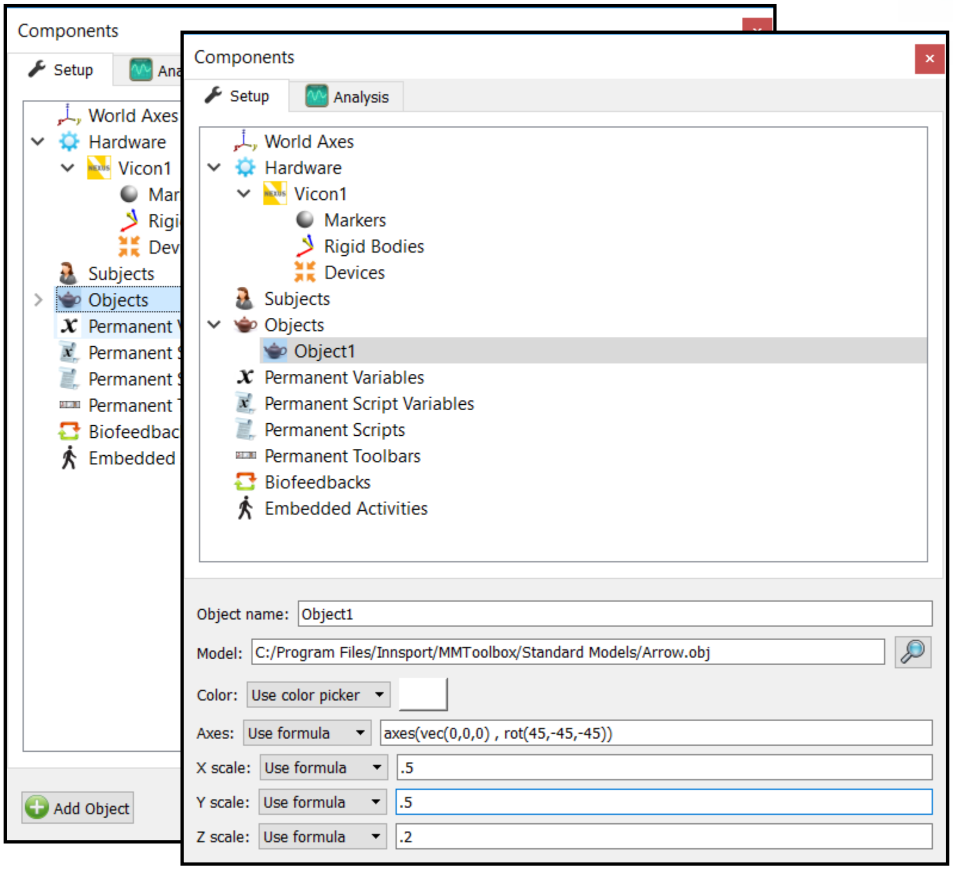


Different choices can be made for each joint. Typically, these choices include locating with a variable expression, locating with a marker, digitization with a stylus, functional methods or use of a regression offset from some set of landmarks. In the case, above, the Bell Method which is a regression based on ASIS and PSIS landmarks was selected.

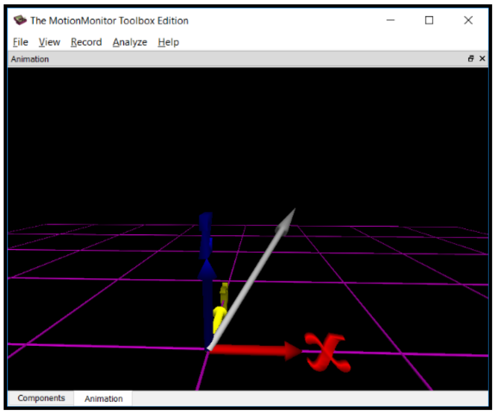
When all of the bone segments and joints have been defined, one can again select the Subject1 node and click on the Calibrate button at the bottom of the parameters panel. The system will then provide screen prompts to complete setup of the subject. If a setting was missed an informational message will be displayed as the Calibration process is performed.

Objects

Objects or 3D image files of golf clubs, baseball bats, etc. in an .obj file format can be added to the animation space by selecting the object node and clicking the "Add Object" button. Selecting the "object1" brings up the parameter panel in which the .obj file is selected, given a color, positioned in



the animation window and scaled. In this case an arrow object was selected and given a fixed position and orientation. The axes variable specifies its vector position as the origin and its orientation as rotated 45 degrees about each axis. If a Rigid Body from a hardware device was assigned to the object, the object's position and orientation can be tracked dynamically based on the motion of the rigid body.



Beyond simply adding additional visualizations to an Animation window, data from the object can also be accessed for analyses, including intersections with the object and other data being tracked.

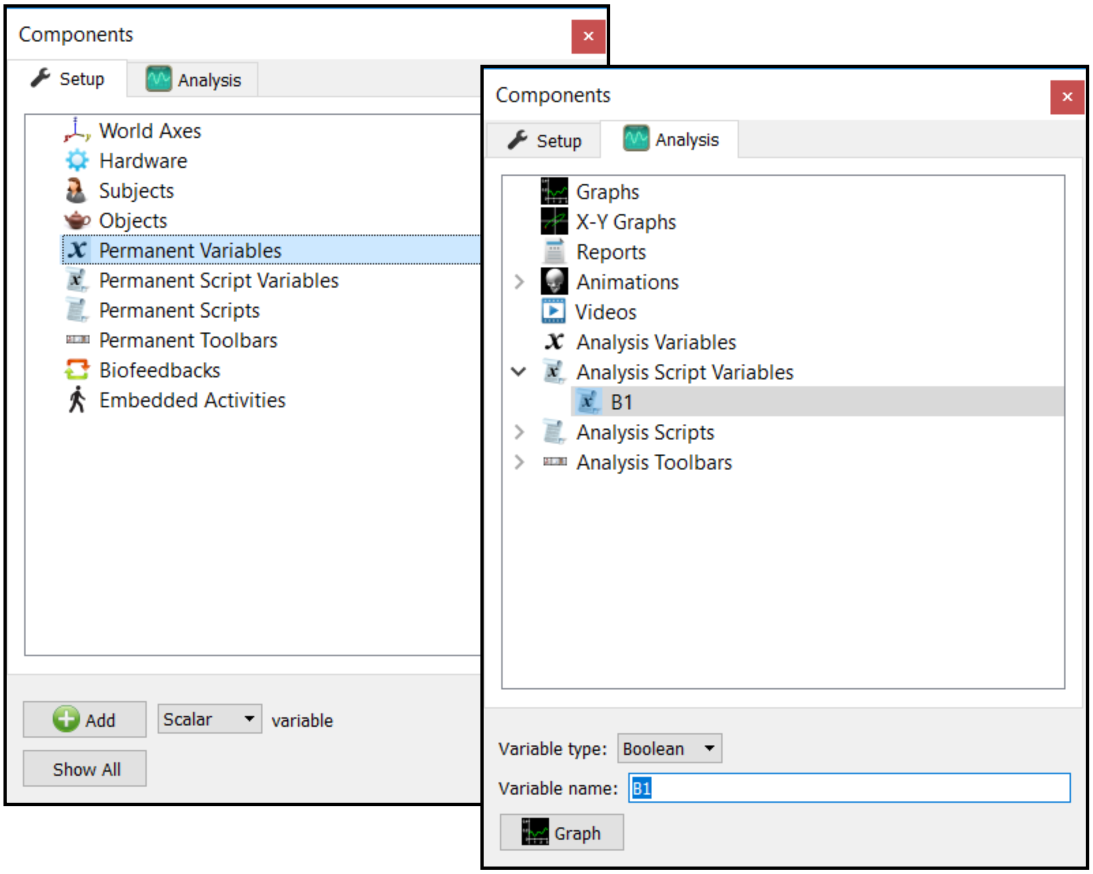
Variables

Earlier, the construction of variables was discussed. Variables can be created on both the setup (permanent variables) and analysis (analysis variables) tabs as either script or non-script variables.

Permanent Variables are variables that were known at the time the activity was collected. A Permanent Variable cannot be changed after collection is completed. Analysis variables are variables that can be created before or after collection and take their value in the present. If a modification is made to an analysis variable, anything dependent on that variable will be updated to reflect the change.

Script Variables are variables that get their value from a script. Their values can be modified only through execution of a script. Creating Script Variables consists of simply providing a type and name as shown below. Permanent Script Variables are associated with Permanent Scripts and Analysis Script Variables are associated with Analysis Scripts.

Within an Activity, Boolean type Permanent Script Variables can be used as Virtual Event Markers, who get their value through the right mouse click in a graph where the Boolean has been plotted.

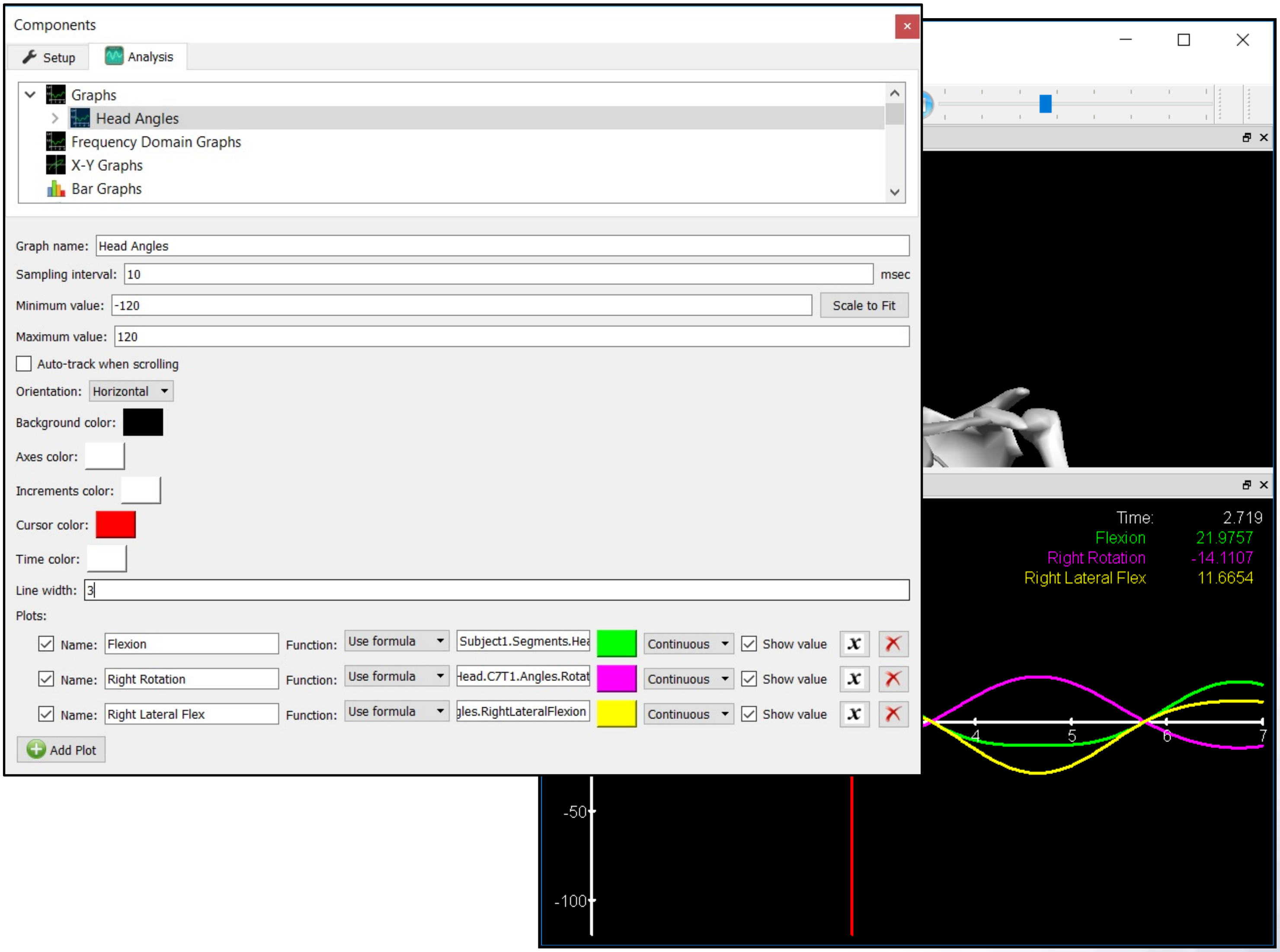


Analysis Components

The Analysis tab provides a rich array of components that can be used in different combinations to enhance the understanding of data collected with The MotionMonitor xGen. These include data definitions, graphical data, animation windows and exported data files for use in other statistical software or in The MotionMonitor Report Generator.

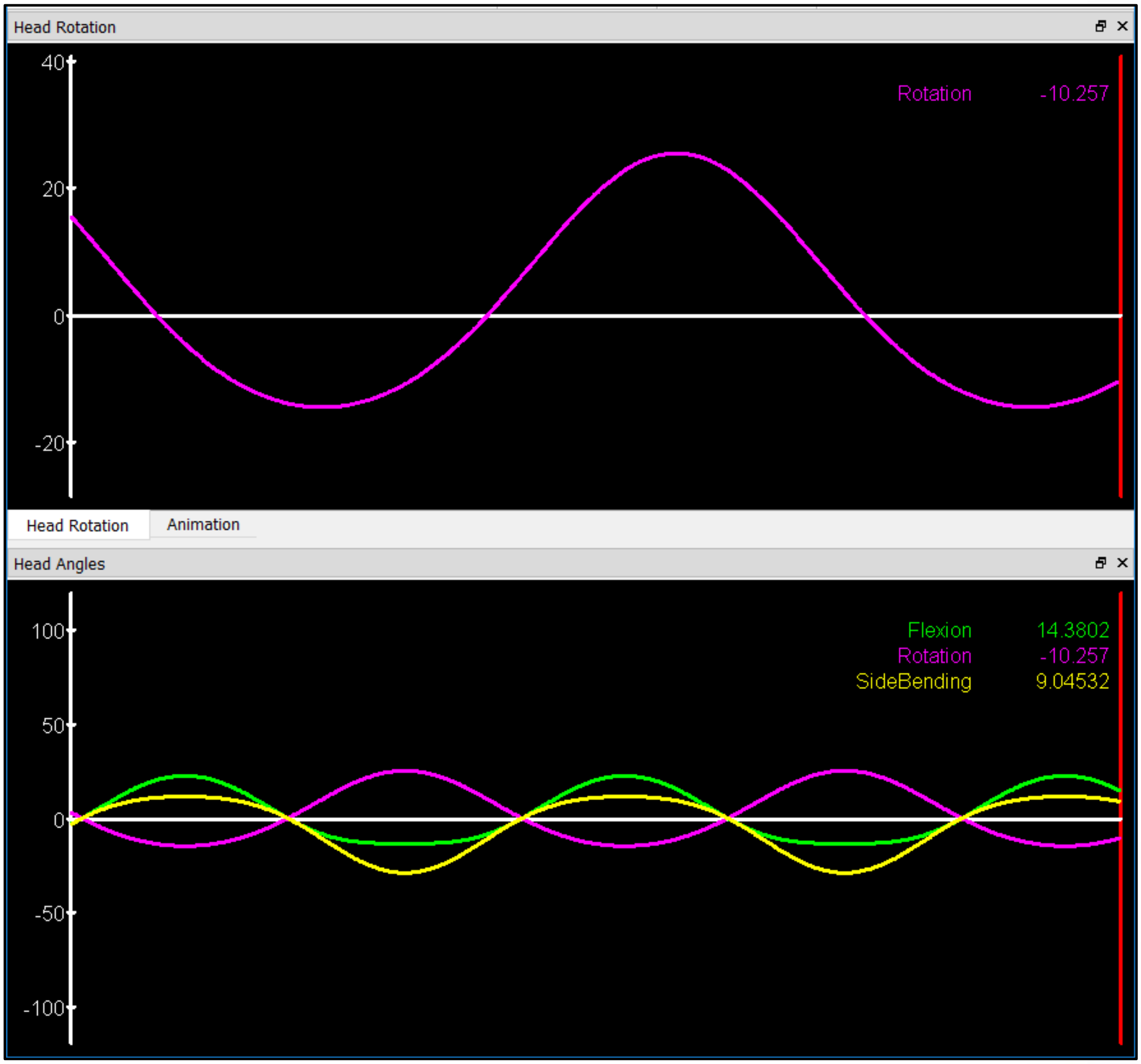
Graphs

As with other components, selecting "Graphs" and clicking on the "Add Graph" button will create a graph. When the graph is selected in the Graphs node, it will generate a parameter panel for controlling the display. The name, scaling, orientation and drawing colors are all controllable parameters. The Sampling Interval determines the resolution of the drawn graph but not the value of the underlying data. Choosing the lower of the hardware's measurement rate interval or 10 msec usually provides acceptable results. As the measurement interval is reduced, more computer resources are needed to draw the plot. Too many graph plots with too small an interval can make the software slow or non-responsive. Any number of plots can be added to a graph using the drop-list or formula methods described in previous sections. Only scalar, integer and Boolean variables can be plotted in a graph. Booleans are plotted as 1 when TRUE and 0 when FALSE.



Time series (Graphs), Frequency Domain, X-Y and Bar graph types are all supported. The parameters described above are consistent across all graph types, with some having parameters unique to that graph type.

If the graph window has been selected (Head Rotation graph in the image below), the scroll button of the mouse can be used to change the scale of the X axis. Similarly, holding the left mouse button down while scrolling the mouse will change the scale of the Y axis. Holding the left mouse button will translate the graph in either the x or y direction. Clicking the right mouse button when located in the graph window will raise a menu of available graph operations.



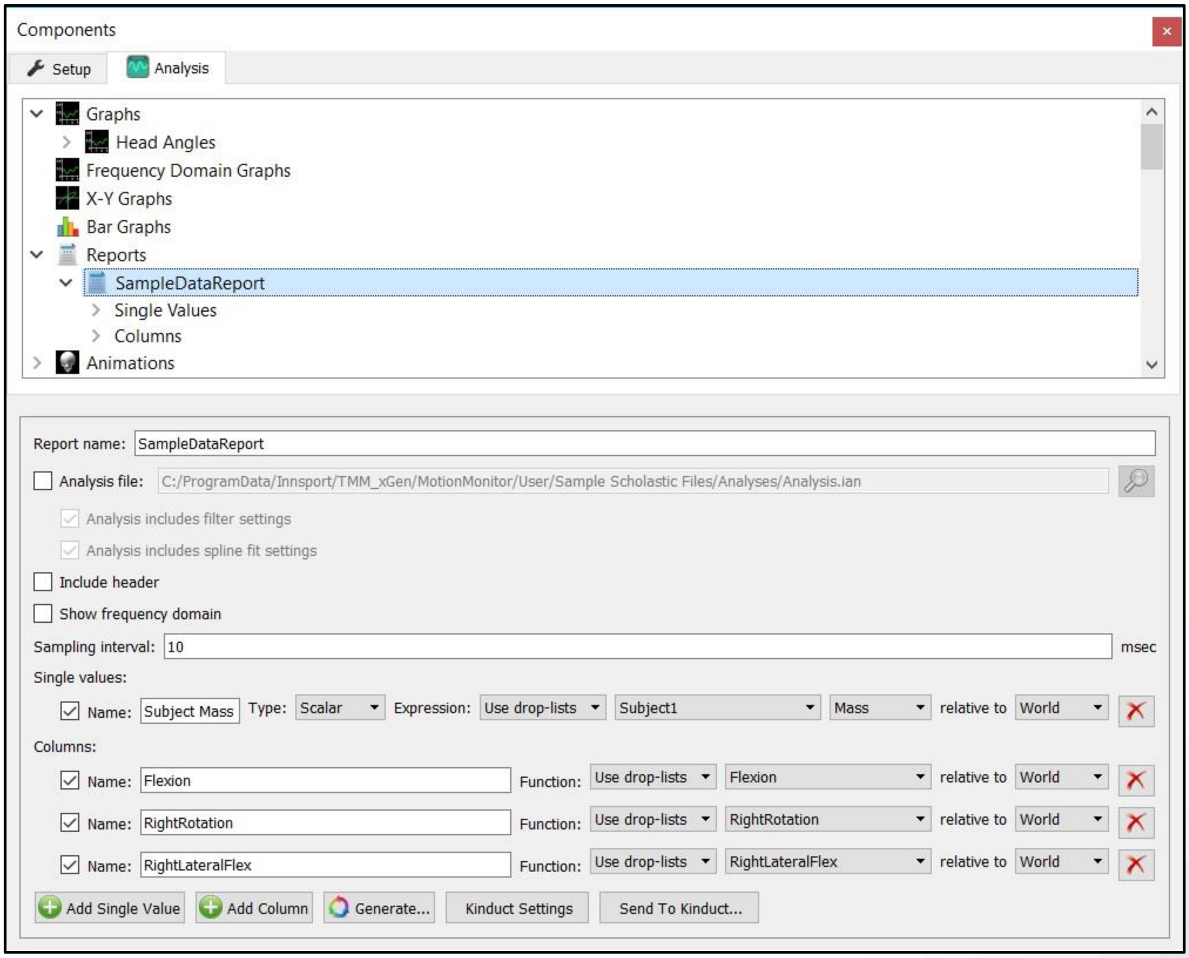
Reports

Adding Reports is a convenient way of configuring different formats for data export. After adding a report, selecting the report in the Reports node will display a parameter panel that controls the report name, existence of a header, the sampling interval which data will be exported at and the data to be exported. Data can be selected using either the drop-list or formula methods as described earlier under the "Data Representation" heading. Additionally, existing Variables may be copied from either of the Components Variables node and then pasted into the Report using the right mouse button. Clicking the "Generate" button will cause the export file to be written in flat ascii format to the user's export folder in

C:\ProgramData\Innsport\TMM_xGen\MotionMonitor\User\UserID\Export.

Because all of the selections on the Analysis tab can be saved and reloaded as an AnalysisFile.ian, it may be desirable to have the data in the report computed in accordance with a particular analysis file. In that case, the user has the option of specifying an analysis file, along with selecting whether to include components from the analysis file – including filter settings and spline filter settings, that should be applied to the data before creating the report. Data can be exported in the frequency domain by enabling the Show frequency domain checkbox.

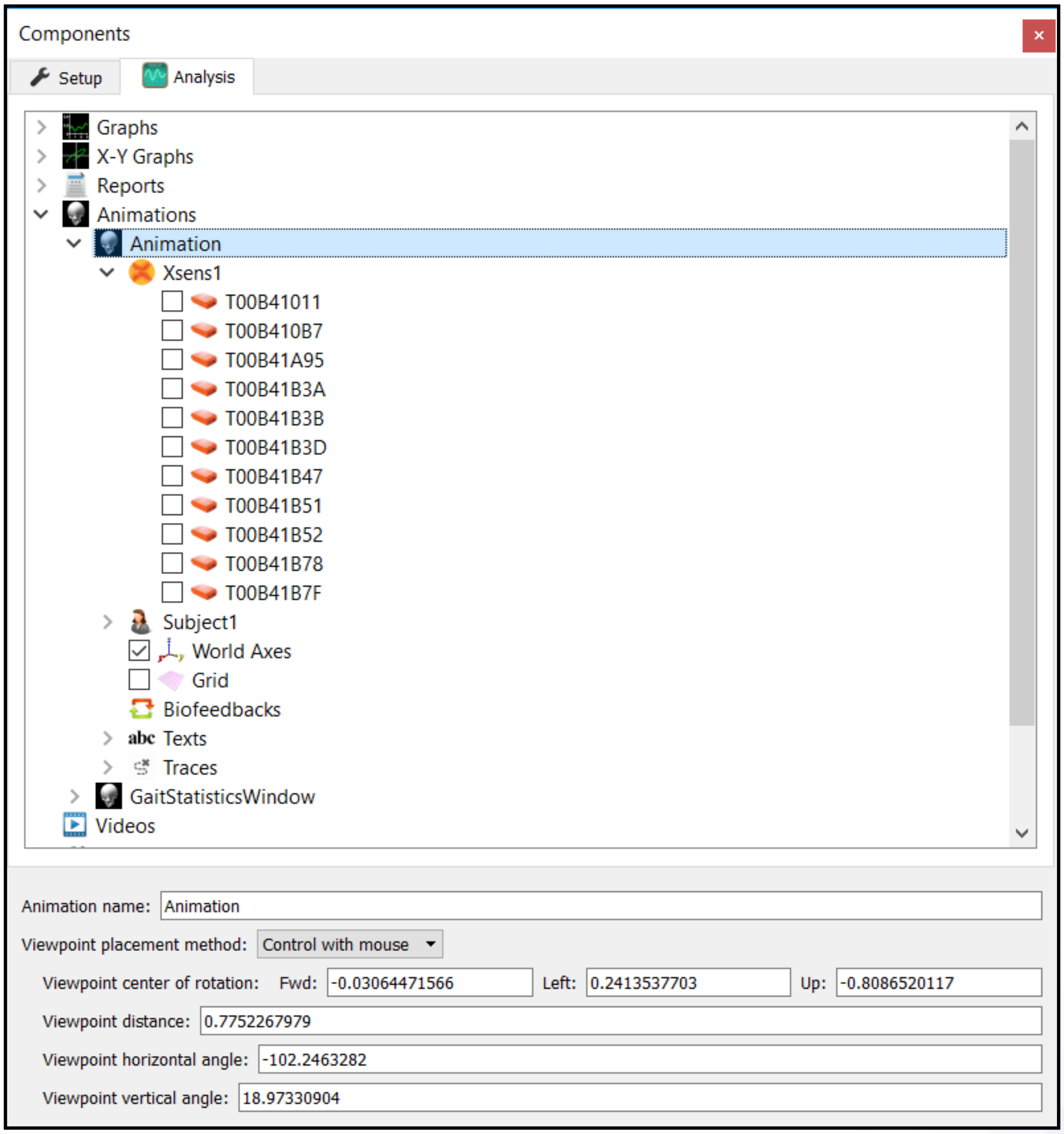
When generating a report from within an Activity, only the data from said Activity will be exported. When generating a report from within the Live Window, multiple Activities may be selected for a batch export and an analysis file can be applied.



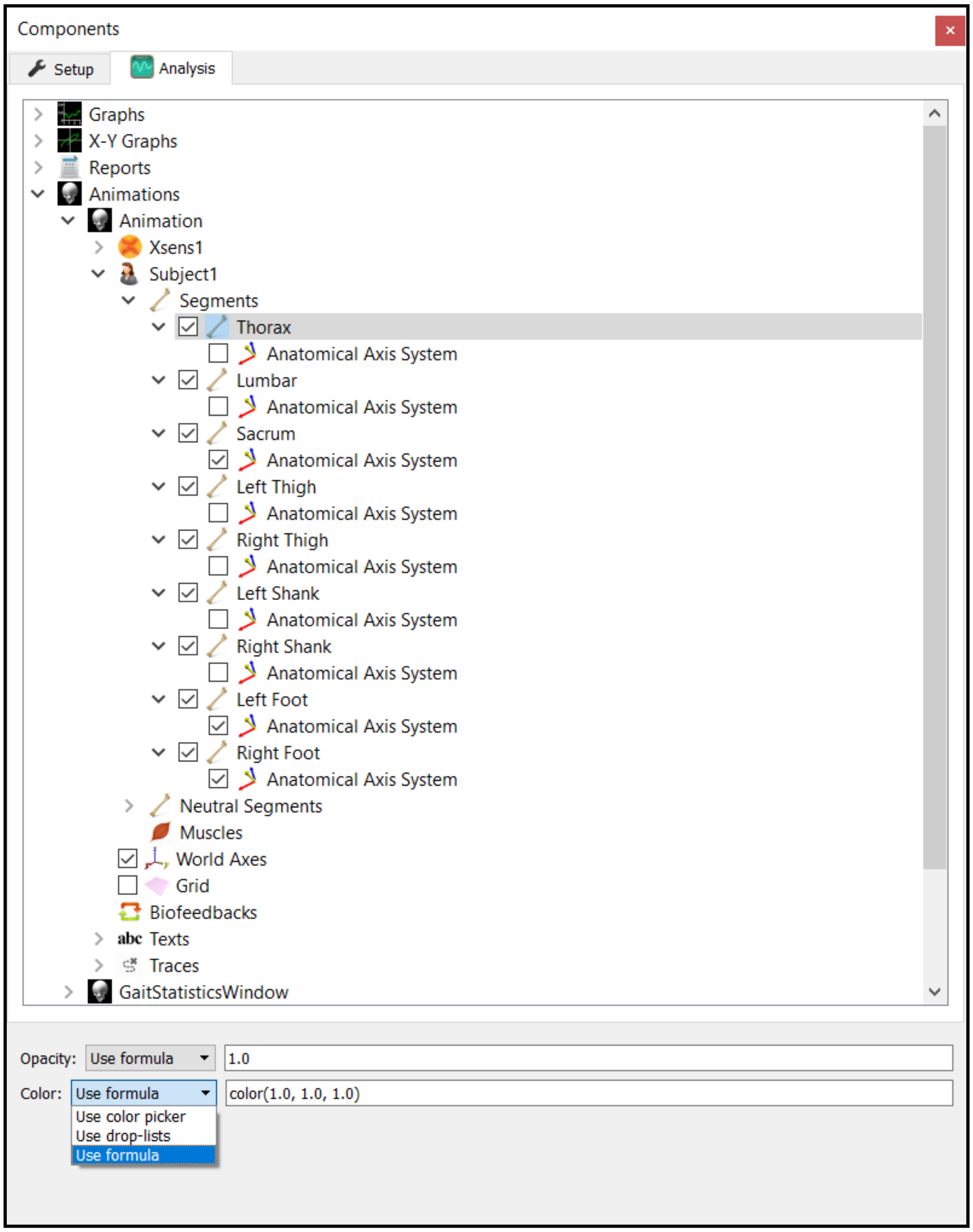
Animations

Adding an animation window to the workspace is an opportunity to have data come alive in a visual sense. Any number of animation windows can be added to the workspace to perform different functions. Selecting and expanding the Animation in the Animations node displays viewing parameters in the parameter panel. The view point into the animation window can be controlled by the mouse (translation holding the right button, rotation holding the left button and zoom using the scroll wheel) or by an expression when a fixed view point is desirable or to present the animation in a particular, dynamic visual sequence.

A list of objects available for display in the animation window automatically appears below the selected animation. Some objects can be toggled for display in the animation window simply by marking the check box. Other objects can have their visibility and/or color modified dynamically or their movement through time displayed as a trace. The stylus, hardware sensors, world axes and floor grid are examples of objects which are toggled on or off.

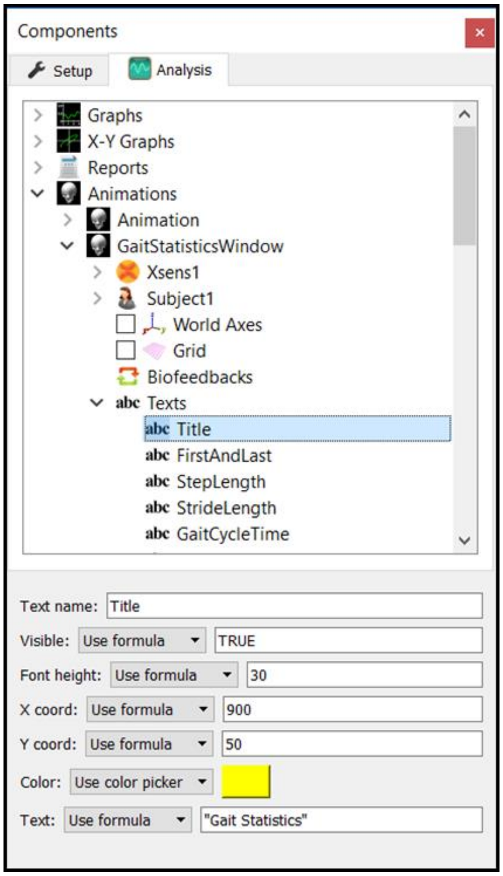


Subject Segments can be toggled on or off by marking the checkbox. In addition, the opacity and color can be modified statically or dynamically. For example, it may be desirable to change the color and opacity of a bone segment as a function of the position, velocity or acceleration of the bone segment.



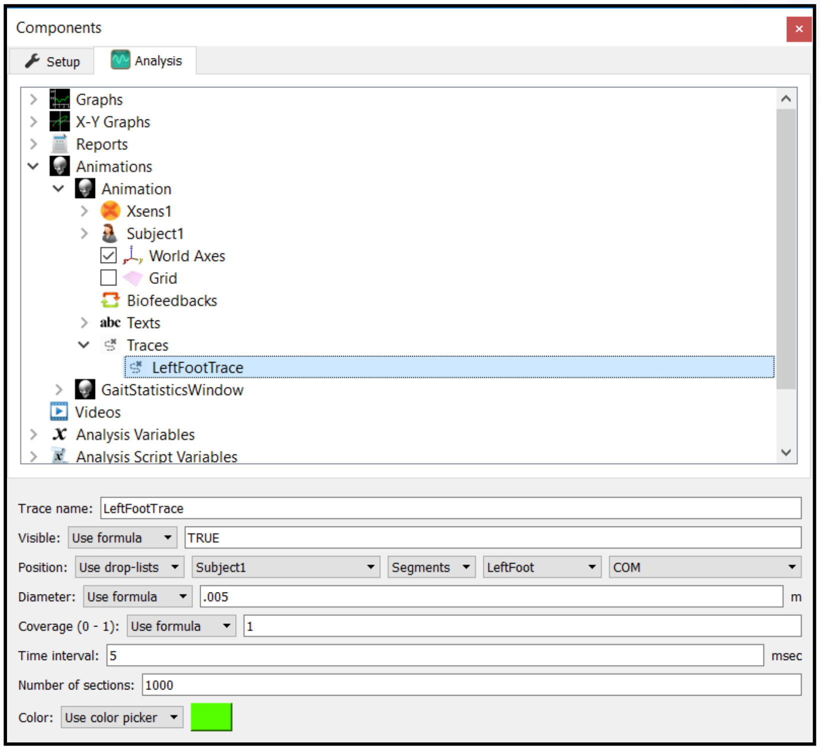
Texts provide an opportunity to present data in tabular form. Texts can also be used during data collection to guide the subject in certain behaviors or as part of biofeedback exercises. In the image below, text is being added to an animation window named “GaitStatisticsWindow”. Any number of text fields can be added to the Texts node. Selecting a particular text, displays the parameter panel.

The name of the text, the Boolean condition under which the text will be visible, the font height in pts, location, color, and text content can be entered with a formula for very powerful control of the display. The x, y location is measured from the upper left corner (0,0) of the animation window. The results of the settings above are shown in the application image below.



Traces can be used to visualize the movement of the body during different activities. The trace of the left ankle in the image below was defined by adding a trace to the Traces node under the Animation window. See the analysis components image below.

Selecting it in the trace node brings up a parameter panel with naming, visibility and color selections similar to the text parameter panel. It is also necessary to identify the object to be traced and dimensional characteristics of the trace. The diameter field is in meters and a setting of around 5 mm or .005 generally provides satisfactory results. Coverage (0-1) indicates the portion of a time interval that is to be covered with the trace. The time interval controls the length of a particular section of the trace. The number of sections indicates the total length of the trace. Because the settings all work in concert to provide an image, it is best to experiment with the settings in real time as the results will immediately appear in the animation window.


Components
Setup
Analysis
Graphs
X-Y Graphs
Reports
Animations
Animation
Xsens1
Subject1
World Axes
Grid
Biofeedbacks
Texts
Traces
LeftFootTrace
GaitStatisticsWindow
Videos
Analysis Variables
Analysis Script Variables
Trace name: LeftFootTrace
Visible: Use formula TRUE
Position: Use drop-lists Subject1 Segments LeftFoot COM
Diameter: Use formula .005 m
Coverage (0 - 1): Use formula 1
Time interval: 5 msec
Number of sections: 1000
Color: Use color picker

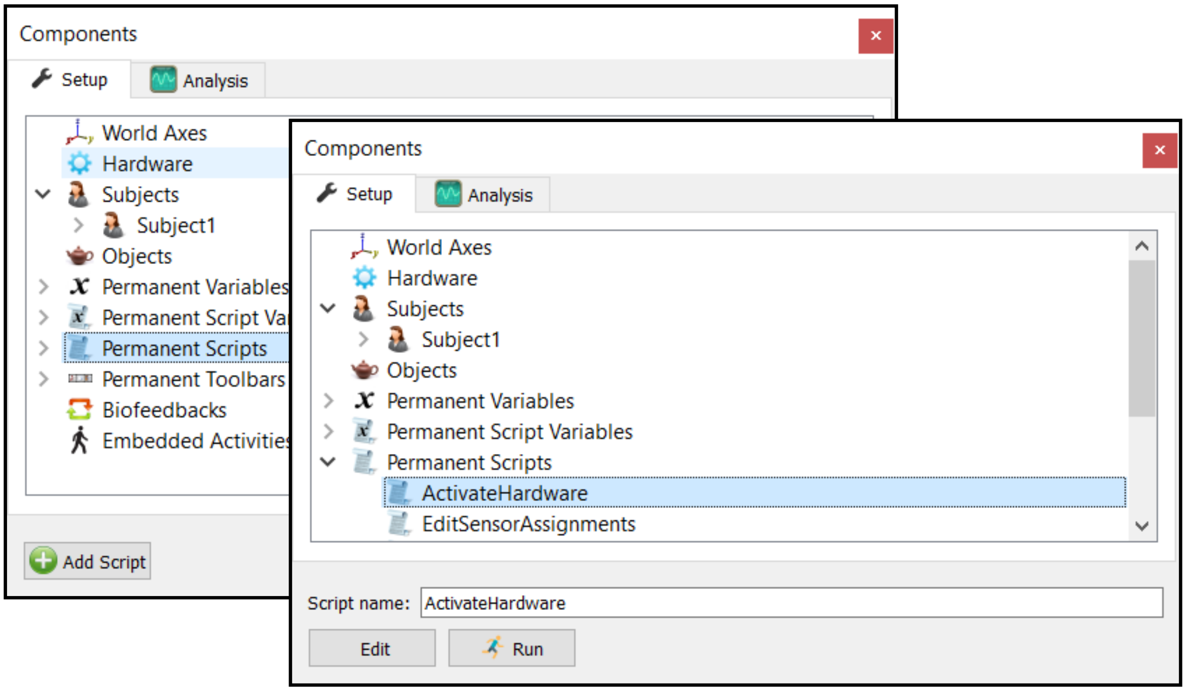

Walking*
File View Play Analyze
Animation
GaitStatisticsWindow
Gait Statistics
Statistic Right Side Left Side
Step Length 0.607 0.572
Stride Length 1.178 1.178
CycleTime (sec) 1.006 1.006
Step Speed (m/sec) 0.603 0.568
Steps per minute 119.337 119.337
Stance Time (sec) 0.667 0.663
Double Limb Stance (sec) 0.313 0.313
Stance Time (%) 0.663 0.659
Double Limb Stance (%) 0.311 0.311
RightHeelStrike
Time: 6.681
RHeelStrike 0
RightToeOff 0

Advanced Features

Advanced Features are a bit more complicated and not necessary for most users of The MotionMonitor xGen software. However, they add significant power to the software. These components make the software usable by clinicians and practitioners who may have no interest in structuring the data collection protocol. They can also be used to ensure that persons responsible for data collection follow prescribed protocols.

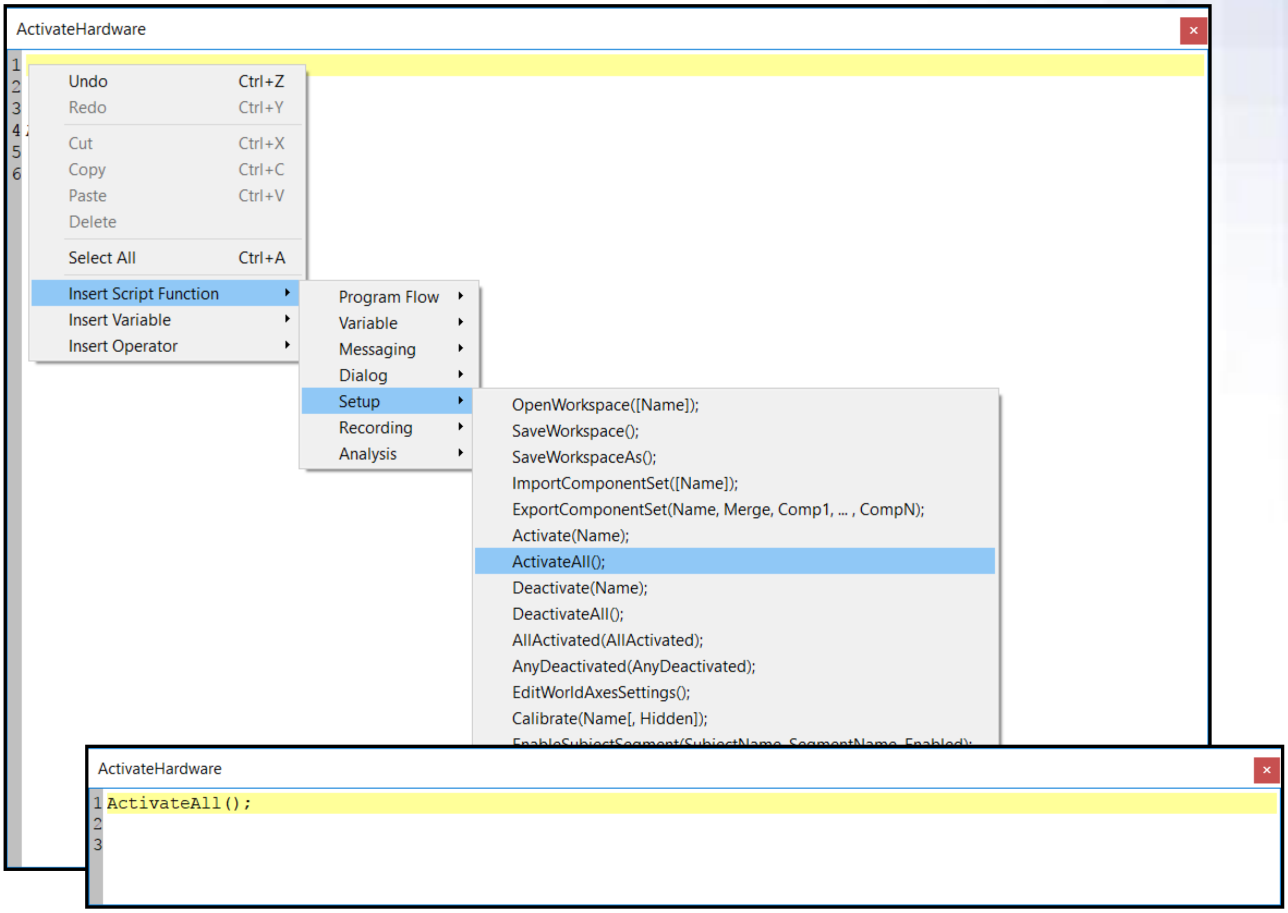
Scripts

Adding a script and selecting the script node displays an edit field for naming the script and an "Edit" and "Run" button. The Run button will execute the script. Typically, a script is executed through use of a Toolbar button, however the Run button is a handy way to test the script's function as well as to highlight any errors. Errors can be corrected immediately, and the script run again for a fast, interactive debugging of the script.

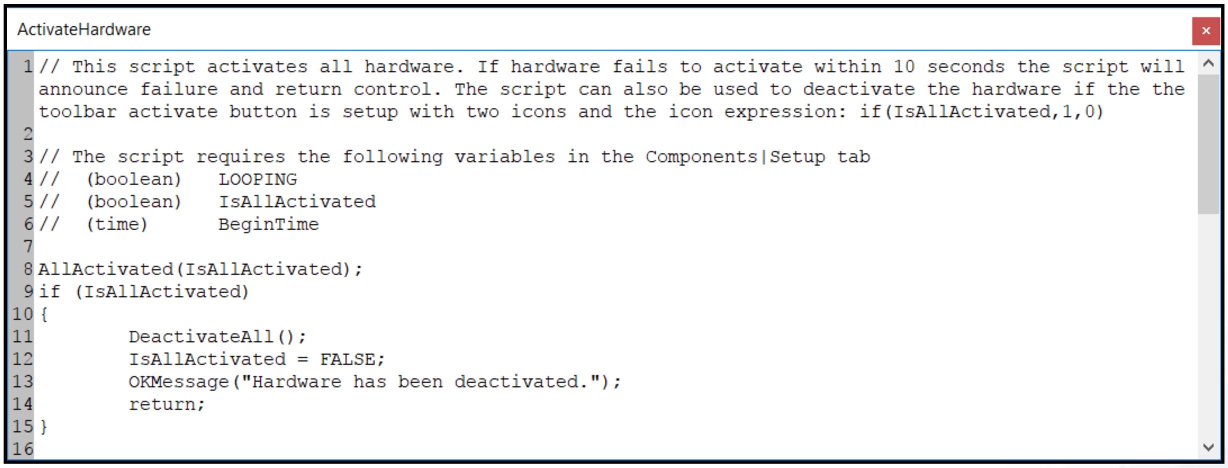


Clicking the "Edit" button will open the script window for editing. Commands can be entered directly into the script window, however, right clicking in the window is the easiest way to call up code. This eliminates the need to be knowledgeable of syntax. Right clicking provides the choice of Script Functions, Variables or Operators. Script Functions include Program Flow, Variable, Messaging, Dialog, Setup, Recording and Analysis.

Scripts can be a simple, single purpose command such as “ActivateAll()” hardware shown below.



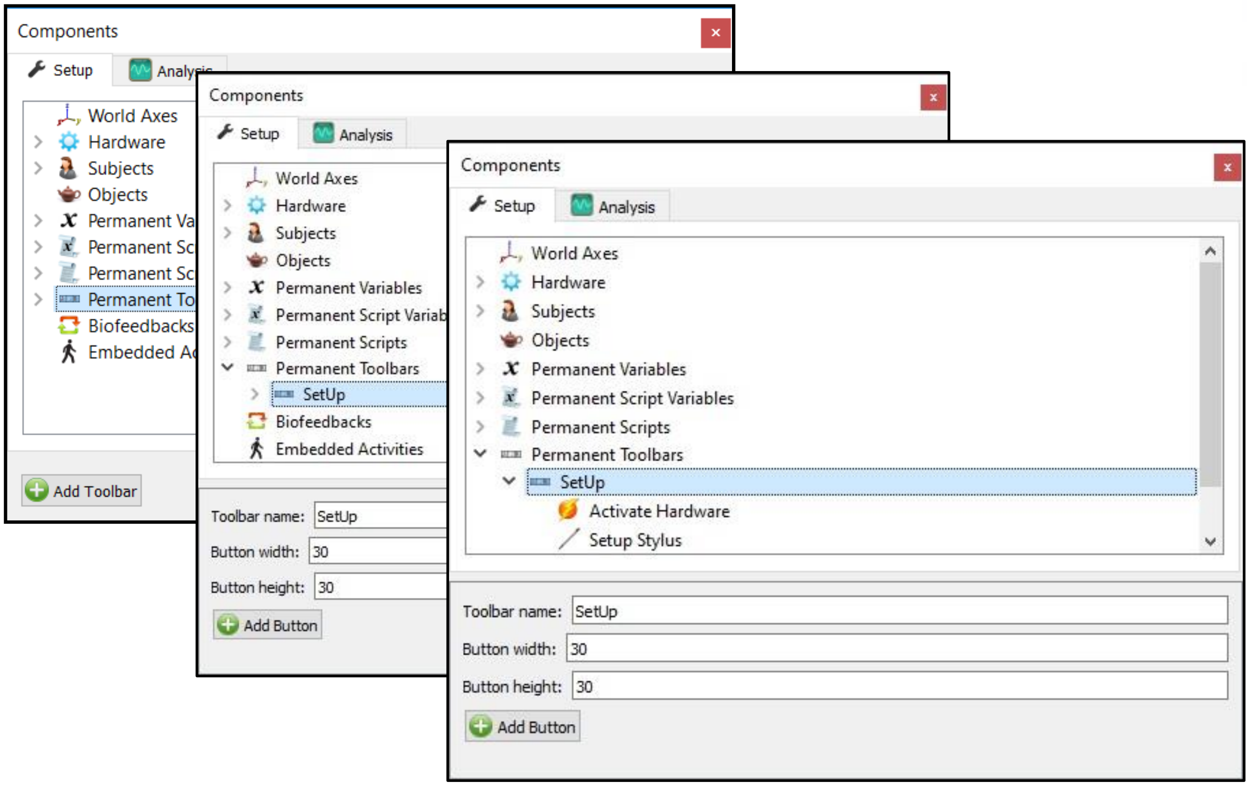
Scripts can also be more complex consisting of //comments, Program Flow statements like if{} else{} and Messaging to control execution of the script. The “Insert Script Function” can be used together with “Insert Variable” and “Insert Operator” to completely eliminate the need to know code syntax.



For more information on setting up scripts for your applications, contact your client support engineer.

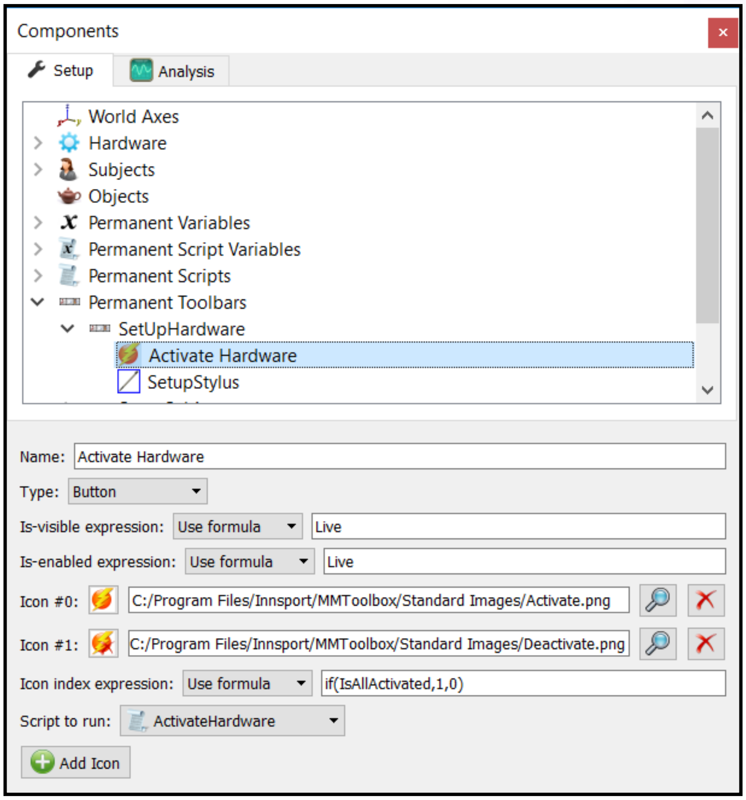
Toolbars

Toolbars are essentially an Icon or button which when pressed will execute a script. Creating Toolbars is easy and consists of first adding the toolbar to either the Setup or Analysis Toolbars, then adding any number of buttons to the toolbar. Setup Toolbars are typically associated with actions involved in setting up an experiment, capturing of data or if you don't want them to be overwritten when loading an Analysis file. Analysis Toolbars are typically used on collected data to perform different analyses. There may be many Analysis Toolbars that compute different data, display different graphs, different animation windows and export different reports. To suit the needs of an application or particular display, the button width and height for icons in a Toolbar can be set through the Toolbar parameters panel.



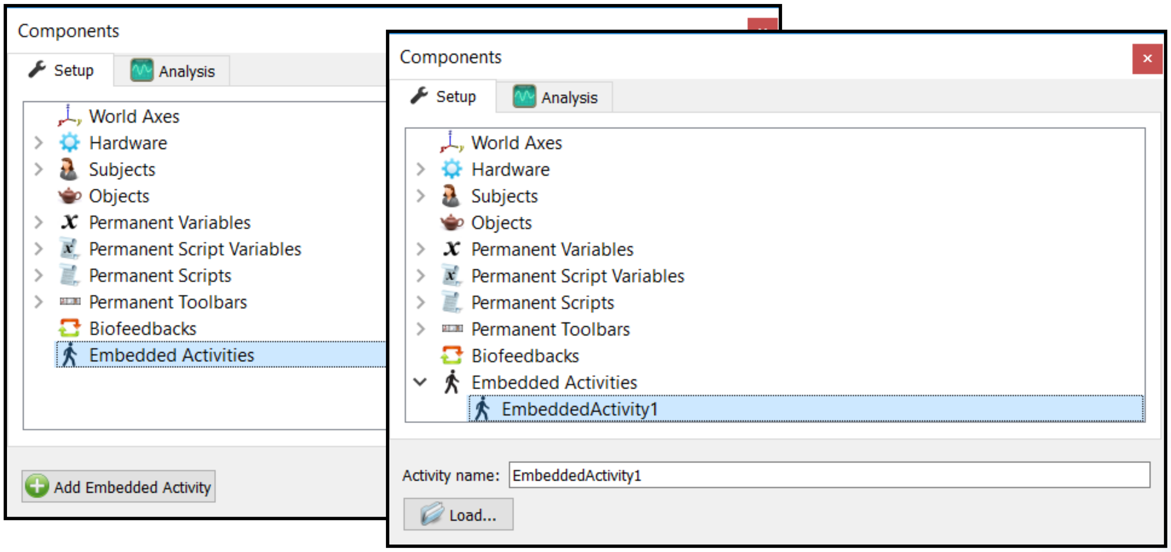
Selecting the button under the Toolbar node displays the parameter panel that can be used to define the behavior of the button when pressed. See the image below as an example.

Parameters include the button name or text to be displayed when the mouse hovers over the button. There are two Boolean fields that control when a button is visible and enabled. These can be set as constants or by formula. To always be visible, one might specify the Boolean, TRUE. In this case the global variable "Live" is used to indicate it should be displayed in the Live window but not in activity windows. The file path to the images for the icon need to be provided. In this case two icon states are being used. The first button is to indicate that the hardware is not yet active and clicking will activate the hardware. The second is to indicate that the hardware is active and is to be deactivated when clicking the button. To make the correct button state appear, the icon index is chosen in an "if" statement based on the script variable "IsAllActivated".



Embedded Activities

Embedded Activities are a method that permits the loading of data from a previously collected activity. This might be done, for example, to provide data for establishing biofeedback targets. Adding embedded activities follows the same process as other components.



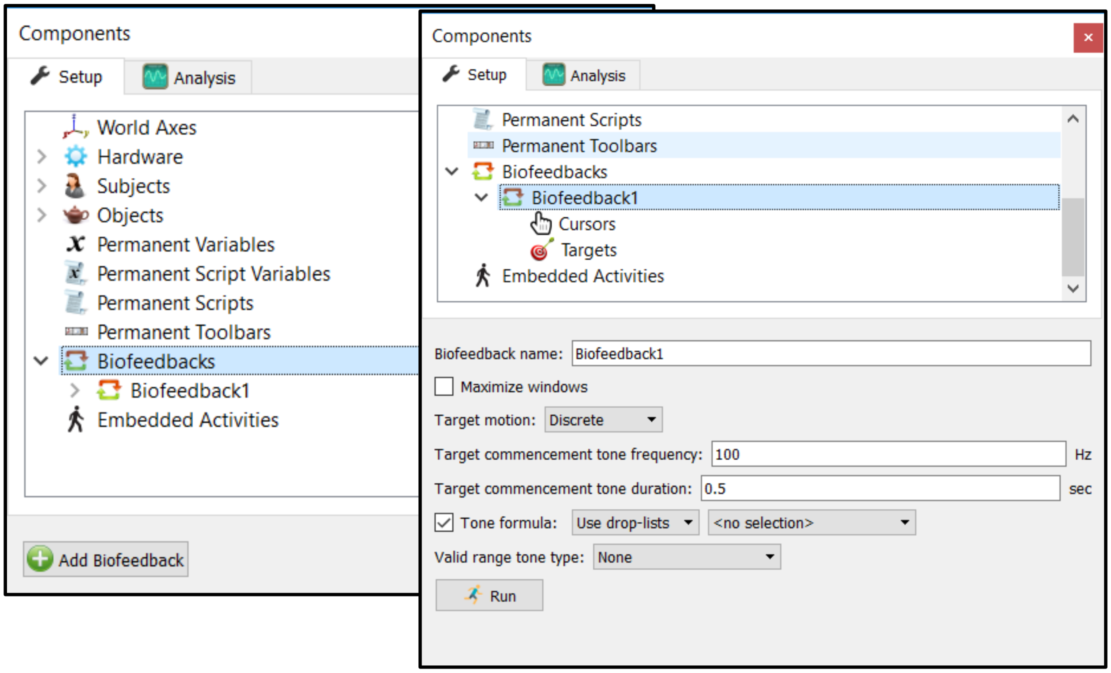
Selecting the activity under the Embedded Activities node, displays a field to name the activity and a button to load an activity. Click the "Load…" button and follow the on screen prompts to load the desired activity. For more information on using Embedded Activities, contact your client support engineer.

Biofeedbak

Biofeedback can be thought of as a target (either static, dynamic, or randomized) and a chase cursor which tracks the target. This might be something as simple as tracking a moving object with the tip of one's finger or elevating an object to a specific height as a function of one's shoulder elevation. Visual, auditory and tactile feedback based on the degree to which the cursor tracks with the target would be a typical feedback. Another form of biofeedback might be the physical or psychological response to certain stimuli in complex, stereoscopic virtual worlds.

The MotionMonitor xGen provides a clean, easy to configure interface for creating biofeedback exercises that range from the simple to complex. The simplest form can be experienced by adding a Biofeedback to the Biofeedbacks node. Its parameter panel includes fields for the name, optional characteristics of a commencement tone that signals the beginning of the session, and the tone formula to be used when the cursor is in the valid range. The tone type can be made proportional to its distance from the target through the drop-list. A checkbox provides for running the session in a maximized window. The target motion can be selected as discrete or randomized where the location within a specified range is determined by a randomly selected velocity and acceleration value.

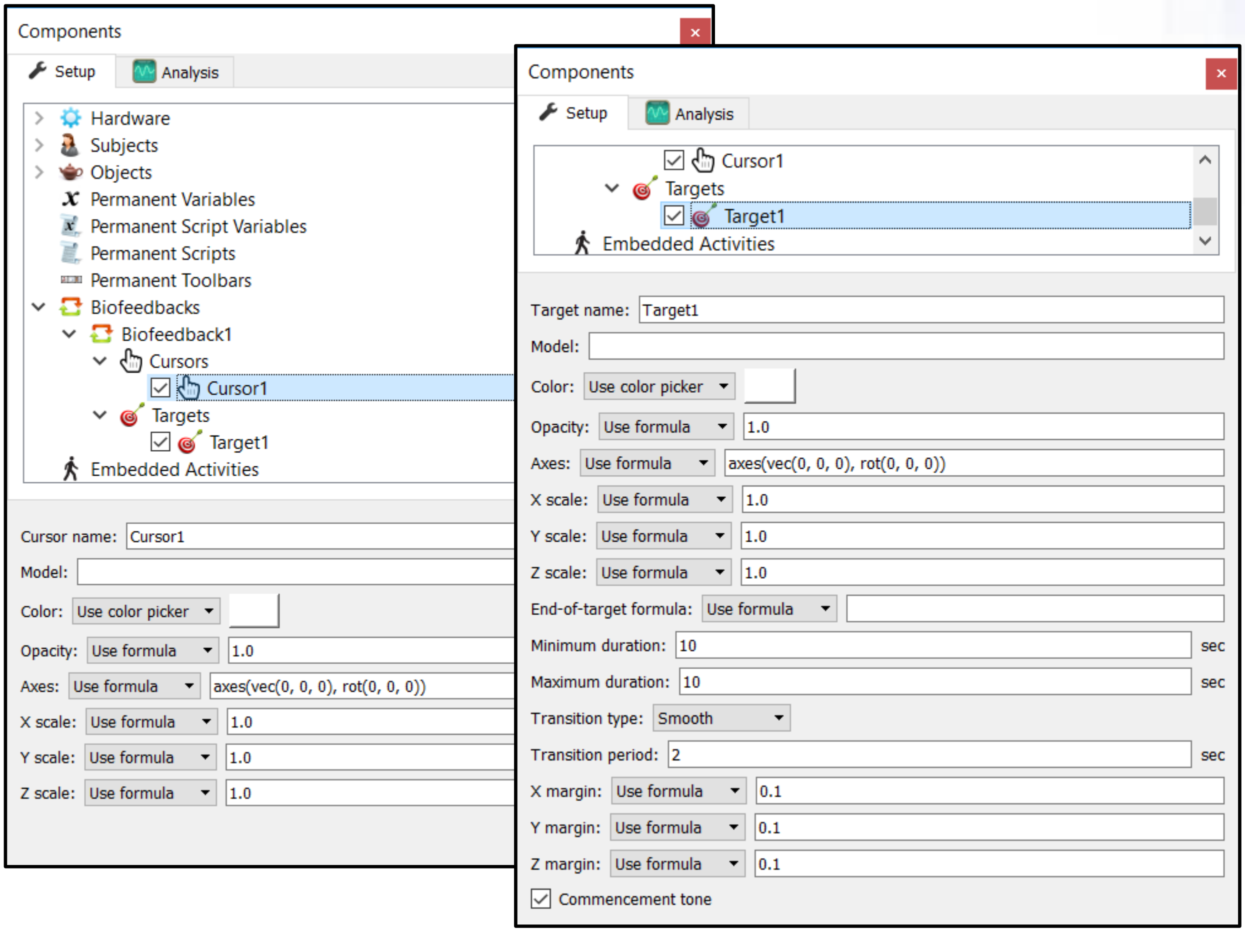
The Run button will execute the biofeedback display and begin recording. Typically the Run button is used with a Toolbar icon so that the components dialog do not need to be open.



Selecting the Cursors and Targets under the Biofeedback1 node will permit the addition of any number of cursors and targets while Biofeedback is not running.

Selecting a specific Cursor as shown below displays the parameter panel. The model file, color and opacity as well as the axes variable that will control motion of the cursor are specified in this panel.

Selecting one of the targets displays its parameter panel. Again, the target can be displayed with a model .obj file whose color and opacity are controlled by formula. Its position and orientation are controlled by the axes variable which may be a constant. The display of the target is for a random period between the minimum and maximum duration unless ended earlier by the “End of Target” Boolean. Transition type and period controls the manner and time in which the target transitions to the next target. The margin specifies the distance from the target that is considered “valid” for purposes of providing visual, auditory, or tactile feedback.



Appendix A

Variable and Script Operators

Boolean Types, which can take a value of TRUE, FALSE or INVALID (Where B1 and B2 are Boolean values)

-	INVALID	Sets the value for the Boolean variable to INVALID
-	FALSE	Sets the value for the Boolean variable to False
-	TRUE	Sets the value for the Boolean variable to True
-	B1 == B2	B1 is equal to B2
-	B1 != B2	B1 s not equal to B2
-	! B	Not B
-	B1 || B2	B1 Or B2
-	B1 && B2	B1 And B2
-	if(B, ValueIfTrue, ValueIfFalse)	If clause 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 as any variable type)
-	count(B, BaseTime, Interval)	Counts the number of times B has transitioned from FALSE to TRUE since BaseTime. Interval is a sampling frequency, in seconds. Returns an Integer.
-	prevtruetime(B, Interval[, NumToSkip])	Returns the previous time at which B transitioned to a TRUE value. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument.
-	nexttruetime(B, Interval[, NumToSkip])	Returns the next time at which B transitioned to a TRUE value. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument.

Integer Types, which are whole numbers (Where N1 and N2 are Integer values)

-	INVALID	Sets the value for an Integer variable to Invalid
-	OK	Sets the value for an Integer variable to OK
-	CANCEL	Sets the value for an Integer variable to Cancel
-	YES	Sets the value for an Integer variable to Yes

- NO	Sets the value for an integer variable to No
- N1 == N2	N1 is equal to N2
- N1 != N2	N1 is not equal to N2
- N1 < N2	N1 is less than N2
- N1 <= N2	N1 is less than or equal to N2
- N1 > N2	N1 is greater than N2
- N1 >= N2	N1 is greater than or equal to N2
- N1 + N2	N1 plus N2
- - N	Negates the sign for an Integer, N
- N1 - N2	N1 minus N2
- N1 * N2	N1 multiplied by N2
- N1 / N2	N1 divided by N2
- Switch(N, Case1, Value1, … , CaseN, ValueN, Default)	Controlling 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).

Scalar Types, which are real numbers (Where X1 and X2 are Scalar values)

- INVALID	Sets the value for a Scalar variable to Invalid
- X1 == X2	X1 is equal to X2
- X1 != X2	X1 is not equal to X2
- X1 < X2	X1 is less than X2
- X1 <= X2	X1 is less than or equal to X2
- X1 > X2	X1 is greater than X2
- X1 >= X2	X1 is greater than or equal to X2
- X1 + X2	X1 plus X2
- - X	Negates the sign for a scalar, X
- X1 – X2	X1 minus X2
- X1 * X2	X1 multiplied by X2
- X1 / X2	X1 divided by X2
- X1 % X2	Modulus operator. Remainder of X1 divided by X2 (Returns an Integer value with the same sign as X1).
- X1 ^ X2	X1 to the power of X2
- scalar(N)	Returns the Scalar value for Integer, N

- trunc(X)	Truncates the value for a Scalar, X, by removing any decimals and returns an Integer value
- sin(X)	Calculates the trigonometric sine function for a Scalar, X
- cos(X)	Calculates the trigonometric cosine function for a scalar, X
- tan(X)	Calculates the trigonometric tangent function for a Scalar, X
- asin(X)	Calculates the inverse trigonometric arcsine function for a Scalar, X
- acos(X)	Calculates the inverse trigonometric arccosine function for a Scalar, X
- atan(X)	Calculates the inverse trigonometric arctangent function for a Scalar, X
- atan2(Y, X)	Calculates the inverse trigonometric arctangent function for 2 Scalar variables, Y and X
- disp(X, BaseTime)	Calculates the change in value of a Scalar, X, from the time, BaseTime. Each point in time is evaluated relative to the value of X at BaseTime.
- diff(X)	Calculates the 1st derivative of a Scalar, X
- diff2(X)	Calculates the 2nd derivative of a Scalar, X
- spline(X, Interval)	Applies a Spline fit to a Scalar, X, without any filtering applied. Interval is a sampling frequency, in seconds.
- butterworth(X, Interval, Freq)	Applies 4th order zero phase shift Butterworth filter of a specified frequency to a Scalar, X. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz.
- chebyshev(X, Interval, Freq)	Applies 4th order zero phase shift Chebyshev filter of a specified frequency to a Scalar, X. Interval is a sampling frequency, in seconds.Freq is the cut-off frequency, in Hz.
- fftlowpass(X, Interval, Freq, Rolloff)	Applies a low-pass filter of a specified frequency to a Scalar, 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.

- ffthighpass(X, Interval, Freq, Rolloff)	Applies a high-pass filter of a specified frequency to a Scalar, 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.
- fftnotch(X, Interval, Freq, Width, Rolloff)	Applies a notch filter of a specified cut-off frequency and width to a Scalar, X. Rolloff is the width of the transition band, in Hz.
- int(X, BaseTime, Interval)	Calculates the integral of a Scalar, X, where BaseTime is the starting point for performing the integration and Interval is the sampling interval for performing the calculation
- avg(X, BaseTime, Interval[, EndTime])	Average value of X since BaseTime. Interval is a sampling frequency, in seconds. EndTime is an optional argument that will prevent further computations after the specified time.
- movavg(X, Period, Interval[, Time])	Moving-average filter for 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.
- rms(X, Period, Interval)	Calculates the RMS for a Scalar, X, where Period is the total time period (centered on the current time) and Interval is the sampling interval for performing the calculation
- min(X, StartTime, StopTime, Interval)	Identifies the minimum value for a Scalar, X, occurring during a time period defined by StartTime and Stop Time and Interval that is the step size (in seconds) for evaluating your time period.
- max(X, StartTime, StopTime, Interval)	Identifies the maximum value for a Scalar, X, occurring during a time period defined by StartTime and Stop Time and Interval that is the step size (in seconds) for evaluating your time period.

- tmin(X, StartTime, StopTime, Interval)	Identifies the time when the minimum value for a Scalar, 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
- tmax(X, StartTime, StopTime, Interval)	Identifies the time when the maximum value for a Scalar, 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
- islocalmin(X, Period, Interval)	Determines if the current value is a local min within the specified period, as determined by sampling readings at the specified interval. Period and Interval are both times, in seconds.
- islocalmax(X, Period, Interval)	Determines if the current value 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.
- prevmintime(X, Period, Interval[, NumToSkip])	Returns the previous time at which 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.
- nextmintime(X, Period, Interval[, NumToSkip])	Returns the next time at which 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.
- prevmaxtime(X, Period, Interval[, NumToSkip])	Returns the previous time at which 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.

- nextmaxtime(X, Period, Interval[, NumToSkip])
  Returns the next time at which 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.
- prevminvalue(X, Period, Interval[, NumToSkip])
  Returns the previous local-minimum value of X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument.
- nextminvalue(X, Period, Interval[, NumToSkip])
  Returns the next local-minimum value of X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument.
- prevmaxvalue(X, Period, Interval[, NumToSkip])
  Returns the previous local-maximum value of X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument.
- nextmaxvalue (X, Period, Interval[, NumToSkip])
  Returns the next local-maximum value of X. Period is the sampling period, in seconds. Interval is a sampling frequency, in seconds. NumToSkip is an optional Skip argument.
- maxfrom(X, BaseTime, Interval) Retuns the maximum value of X starting from the time, BaseTime. Interval is a sampling frequency, in seconds.
- meanfreq(X, StartTime, StopTime, Interval[, Detrend])
  Returns the mean frequency of 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.

- medianfreq(X, StartTime, StopTime, Interval[, Detrend])
  Returns the median frequency of 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.
- sampen(X, StartTime, StopTime, Interval, N, M, R)
  Entropy calculation. N is the number of samples to average, M is the pattern length, and R is the pattern-matching tolerance, given as a fraction of the standard deviation of the data points.
- higlen(X, StartTime, StopTime, Interval, K)
  Returns the Higuchi length association with decimation factor K
- higdim(X, StartTime, StopTime, Interval, Kmax[, Tol])
  Recalculates all the lengths up through Kmax, 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.
- katzdim(X, StartTime, StopTime, Interval) Returns a fractal dimension, based on Katz's method

Vector Types, which represent a 3 dimensional position or offset in space and can be characterized by an x, y and z value. Each of the x,y,z components is a scalar. (Where V1 and V2 are vector values)

- INVALID — Sets the value for a Vector variable to Invalid
- vec(X, Y, Z) — Composes a Vector, where x, y and z are Scalar values
- V1 == V2 — V1 is equal to V2
- V1 != V2 — V1 is not equal to V2
- V1 + V2 — V1 plus V2
- -V — Negates the sign for a Vector, V
- V1 – V2 — V1 minus V2
- V * Scale — Multiply a Vector, V, by a scaling factor
- V / Scale — Divide a Vector, V, by a scaling factor

-	mag (V)	Returns the magnitude of a Vector, V, which is a Scalar value
-	unit(V)	Returns the Unit Vector for a Vector, V
-	dot(V1, V2)	Performs the dot product or scalar product of V1 and V2 and returns a scalar value
-	cross(V1, V2)	Performs the cross product or vector product of V1 and V2
-	disp(V, BaseTime)	Calculates the change in value of a Vector, V, from the time, BaseTime. Each point in time is evaluated relative to the value of V at BaseTime.
-	diff(V)	Calculates the 1st derivative of a Vector, V
-	diff2(V)	Calculates the 2nd derivate of a Vector, V
-	int(V, BaseTime, Interval)	Calculates the integral of a Vector, V, where BaseTime is the starting point for performing the integration and Interval is the sampling interval for performing the calculation
-	spline(V, Interval)	Applies a Spline fit to a Vector, V, without any filtering applied. Interval is a sampling frequency, in seconds.
-	butterworth(V, Interval, Freq)	Applies Butterworth filter of a specified frequency to a Vector, V. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz.
-	chebyshev(V, Interval, Freq)	Applies Chebyshev filter of a specified frequency to a Vector, V. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz.
-	fftlowpass(V, Interval, Freq, Rolloff)	Applies a low-pass filter of a specified frequency to a Vector, 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.
-	ffthighpass(V, Interval, Freq, Rolloff)	Applies a high-pass filter of a specified frequency to a Vector, 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.
-	fftnotch(V, Interval, Freq, Width, Rolloff)	Applies a notch filter of a specified frequency and width to a Vector, V. Rolloff is the width of the transition band, in Hz.

Rotation Types, which are measures of orientation and can take the form of Euler angles, Quaternions or Cosine Matrices (Where R1 and R2 are Rotation values)

- INVALID	Sets the value for a Rotation variable to Invalid
- rot(Yaw, Pitch, Roll)	Composes a Rotation, where Yaw (Z), Pitch (Y) and Roll (X) are angles
- rot(Q0, Q1, Q2, Q3)	Composes a Rotation using quaternions where Q0, Q1, Q2, Q3 correspond to w, x, y, z, respectively.
- rot(XAxis, ZAxis)	Composes a Rotation using axes, where XAxis and ZAxis are Vector values of the cosine matrix m(0,0), m(1,0), m(2,0) and m(0,2), m(1,2), m(2,2).
- rot(M11, M12, …, M33)	Constructor for rotation variables, specifying the nine elements of a rotation matrix
- R1 == R2	R1 is equal to R2
- R1 != R2	R1 is not equal to R2
- R * V	Takes a direction/offset Vector, V, from the reference frame of Rotation, R, and places it in the world reference frame. This operator results in a Vector value
- R1 * R2	Takes R2 from the reference frame of R1 and places it in the world reference frame
- angvel(R)	Returns the angular velocity, in degrees/sec, of Rotation, R. Is analogous to performing the diff() operator for a vector.
- angacc(R)	Returns the angular acceleration, in degrees/sec^2, of Rotation, R. Is analogous to performing the diff2() operator for a vector.
- rel(V, R)	Takes a direction/offset Vector, V, from the world reference frame and places it in the reference frame of Rotation, R. This operator results in a Vector value
- rel(R1, R2)	Takes R1 from the world reference frame and places it in the reference frame of R2

Axes Types, which are essentially rigid bodies and have a representation of both position and orientation (Where A1 and A2 are Axes values)

- INVALID	Sets the value for an Axes variable to Invalid

-	axes(Pos, Ori)	Composes an Axes, where Pos is a Vector and Ori is a Rotation
-	A1 == A2	A1 is equal to A2
-	A1 != A2	A1 is not equal to A2
-	A * Pos	Takes a position Vector, Pos, from the reference frame of Axes, A, and places it in the world reference frame. This operator results in a Vector value
-	A * Ori	Takes an orientation, Ori, from the reference frame of Axes, A, and places it in the world reference frame. This operator results in an Orientation value.
-	A1 * A2	Takes A2 from the reference frame of A1 and places it in the world reference frame
-	inv(A)	Inverts Axes, A
-	relpos(Pos, A)	Takes a position Vector, Pos, from the world reference frame and places it in the reference frame of Axes, A. This operator returns a Vector value
-	reldir(V, A)	Takes a direction/offset Vector, V, from the world reference frame and places it in the orientation for the reference frame of Axes, A. This operator returns a Vector value
-	rel(Ori, A)	Takes an Orientation, Ori, from the world reference frame and places it in the reference frame of Axes, A. This operator returns a Rotation value
-	rel(A1, A2)	Takes A1 from the world reference frame and places it in the reference frame of A2
-	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.
-	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.

- 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.
- 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.
- 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 A1.
- iheln(A)	Returns the parameters of the instantaneous helical axis of axis system A. N is the direction vector of the helical axis.
- ihelt(A)	Returns the parameters of the instantaneous helical axis of axis system A. T is the amount of translation along the axis. Units are meters/second.
- ihelphi(A)	Returns the parameters of the instantaneous helical axis of axis system A. Phi is the amount of rotation about the axis. Units are degrees/second.
- spline(A, Interval)	Applies a Spline fit to an Axes, A, without any filtering applied. Interval is a sampling frequency, in seconds.
- butterworth(A, Interval, Freq)	Applies Butterworth filter of a specified frequency to an Axes, A. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz.
- chebyshev(A, Interval, Freq)	Applies Chebyshev filter of a specified frequency to an Axes, A. Interval is a sampling frequency, in seconds. Freq is the cut-off frequency, in Hz.
- fftlowpass(A, Interval, Freq, Rolloff)	Applies a low-pass filter of a specified frequency to an Axes, 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.

- fftthighpass(A, Interval, Freq, Rolloff) Applies a high-pass filter of a specified frequency to an Axes, 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.
- fftnotch(A, Interval, Freq, Width, Rolloff) Applies a notch filter of a specified frequency and width to an Axes, A. Rolloff is the width of the transition band, in Hz.

Time Types, which are time stamps of the form 7-10-2011 14:16:53:848. When used in calculations time types return time in seconds (Where T1 and T2 are Time values)

- INVALID Sets the value for a Time variable to Invalid
- time(PartSec, Sec, Min, Hour, Day, Mon, Year) Composes a Time value
- now() Sets the value of time equal to now
- T1 == T2 T1 is equal to T2
- T1 != T2 T1 is not equal to T2
- T1 < T2 T1 is less than T2
- T1 <= T2 T1 is less than or equal to T2
- T1 > T2 T1 is greater than T2
- T1 >= T2 T1 is greater than or equal to T2
- T + Secs Time, T, plus some amount of seconds
- T – Secs Time, T, minus some amount of seconds
- T1 – T2 T1 minus T2
- attime(Value, T) Returns the value of some variable, Value, at Time, T. This operator takes on whatever variable type Value is defined as
- current(Value) Returns the value of some variable, Value, at the current time. This operator takes on whatever variable type Value is defined as

String Types, which are groupings of alphanumeric characters. Within a formula, the characters are enclosed within quotation marks (Where S1 and S2 are String values)

- INVALID Sets the value for a String variable to Invalid
- “” Composes a String value by enclosing it in quotation marks
- str(N) Converts the Integer, N, to a String value
- str(X, Precision) Converts the Scalar, X, to a String value with, Precision, number of decimal places
- S1 == S2 S1 is equal to S2

- S1 != S2	S1 is not equal to S2
- S1 + S2	S1 plus S2

Color Types, which are expressions that can be defined to dynamically control the color for elements visualized in the animation window, such as an object or skeleton (Where C1 and C2 are Color values)

- INVALID	Sets the value for a Color variable to 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
- C1 == C2	C1 is equal to C2
- C1 != C2	C1 is not equal to C2

Script Program Flow Functions
Note: “Name” when used in these functions is a string or string variable

- if (Condition) {}	IF condition, where Condition, is the condition being defined and the value entered in the brackets is what the script will run while this condition is True
- else{}	ELSE condition, where the value entered in the brackets is what the script will run while the IF clause is False
- while(Condition) {}	WHILE condition, where Condition is the condition being defined and the value entered in the brackets is what the script will run while this condition is True.
- goto Label;	Allows for the script to jump to a specified label elsewhere is the script. The jump can go either forward or backward. The label however cannot be contained within an argument if the goto function is outside of this argument. For example: *goto Label1; while (Condition){Label1: <code>};*
- return;	Prevents the current sequence of commands from proceeding and returns to the invoking function
- RunScript(Name);	Calls a script within a script. There is an optional filename argument, which if absent will cause a file-browser dialog to open.

- Delay(Secs); — 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 anywhere that a break in the processing of the script is desired or after operations such as Saving, Opening and Closing files.

Script Variable Functions

- Var = Exp; — Sets a variable, Var, equal to an Expression, Exp
- 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 of any type, but it must be a regular variable, not a script variable.
- 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
- 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.

Script Messaging Functions

- OKMessage(Prompt); — Generates a Message with an OK button being displayed to the TB2 user. The Prompt should be surrounded by quotation marks.

- OKCancelMessage(Prompt, Result); Generates a Message with OK and Cancel buttons being displayed to the TB2 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 Chancel or the Close buttons are selected.
- OKCancelMessage(Prompt, StylusName, Result); Generates a Message with OK and Cancel buttons being displayed to the TB2 user and also references to a stylus, StylusName, for any digitizing. 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.
- YesNoCancel(Prompt, Result); Generates a Message with Yes, No, and Cancel buttons being displayed to the TB2 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.
- Beep(); Produces an audible “beep” sound
- PlayTone(Freq, Duration); Plays a tone at the specified frequency, Freq, and duration, Duration.
- 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 or .mp3), but without the 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.

Script Dialog Functions

- 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.
- CreateButtonControl(Name, HasBeenPressed, Text);
  Creates a button control that can be added to a dialog window with the name, Name, surrounded by quotation marks, permanent script Boolean variable for whether the button has been pressed, HasBeenPressed, and text for the button, Text, surrounded by quotation marks.
- CreateCheckboxControl(Name, Checked, Text);
  Creates a checkbox that can be added to a dialog window with the name, Name, surrounded by quotation marks, permanent script Boolean variable for whether the box has been checked, and text for the checkbox, Text, surrounded by quotation marks.
- 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, permanent script Integer variable, Integer, and MinValue and MaxValue representing the min and max values that can be entered, respectively.
- 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, permanent script Scalar variable, Scalar, and MinValue and MaxValue representing the min and max values that can be entered, respectively.

-	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
-	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.
-	CreateHorizControlGrid(Name);	Creates the horizontal grid components of a dialog window with name, Name, surrounded by quotation marks
-	CreateVertControlGrid(Name);	Creates the vertical grid components of a dialog window with name, Name, surrounded by quotation marks
-	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
-	CreateDialog(Name, Caption, ControlGridName, DefaultButtonControlName[, Width]);	Creates a dialog window with the name, Name, surrounded by quotation marks, text at the top of the window, 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.
-	OpenDialog(Name);	Opens the dialog window with the name, Name, surrounded by quotation marks
-	CloseDialog(Name);	Closes the dialog window with the name, Name, surrounded by quotation marks

-	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.

Script File Functions

-	Delete(FilePath);	Deletes the specified file, surrounded by quotation marks. 'FilePath' requires the entire file path, including file extension.
-	Rename(OldFilePath, NewFilePath);	Renames the file 'OldFilePath' to the file 'NewFilePath'. Both arguments require the entire file path, including file extension, be surrounded by quotation marks.
-	Copy(SourceFilePath, DestFilePath);	Copies the file 'SourceFilePath' to the file 'DestFilePath'. Both arguments require the entire file path, including file extension, be surrounded by quotation marks.
-	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. 'FolderPath' is the initial folder to look in and requires the entire file path, including file extension, should be surrounded by quotation marks. 'FilenameFilter' is the filter to apply (e.g. "*.iac"), 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"), and the last two arguments are string script variables that receive the file information.

- NextFilename(FolderPath, FilenameFilter, CurrentFilename, NextFilename);
  Will scan through a folder and return 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.
- NextFolderName(ParentPath, FolderNameFilter, CurrentFolderName, NextFolderName);
  Will scan through a directory and return 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.

- 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.
- 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, and the remaining items are string variables.

Script Setup Functions

- 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.
- SaveWorkspace(); Saves open workspace as the Current.iws workspace file.
- SaveWorkspaceAs(); Saves a workspace with a given name after opening a file-browser dialog
- 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.

-	ExportComponentSet(Name,Merge,Comp1,…,CompN);	
		Exports 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 added. The third argument is the list of components to be exported. This list can be of any length, and the listed names should not be contained within quotations.
-	Activate(Name);	Activates the hardware component with the given name
-	ActivateAll();	Activates all hardware listed in the Components Setup tab
-	Deactivate(Name);	Deactivates the hardware component with the given name
-	DeactivateAll();	Deactivates all hardware listed in the Components Setup tab
-	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
-	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
-	EditWorldAxesSettings();	Opens the World Axes dialog
-	Calibrate (Name [, Hidden]);	Runs the calibration procedure for the subject or hardware. The first argument is the name of the hardware to calibrate. The second argument is an optional Boolean which displays or hides the calibration dialogs

- Align(Name[, DeviceIndex]); Performs the hardware alignment procedure for the device, Name, if supported for that hardware type. The second argument is an optional Device Index for hardware devices that have multiple components that support the alignment option.
- 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.
- SetSubjectNeutralStance(SubjectName,NeutralStance);
  Allows you to select a subject's neutral stance using a script. The first argument is the subject name. 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
  4= Arms forward, Thumbs upward
- SetupSubject(Name); Performs the Subject setup procedure for the subject, Name
- EnableCameraViewfinder(CameraName,Enabled);
  Enables or disables the viewfinder (live view) for a camera. It takes two arguments: 'CameraName' is the name of the camera and 'Enabled' is a Boolean to enable or disable the viewfinder.
- LoadEmbededActivity(Name); Loads data from a previously recorded activity under the name, Name, within the current live session or recorded activity
- DeleteEmbeddedActivity(Name); Deletes embedded activity. There is an optional filename argument, which if absent will cause a file-browser dialog to open.

Script Recording Functions

- EditRecordingParams(); Opens the Recording Parameters dialog
- StartRecording(); Initiates the recording process
- StopRecording(); Terminates the recording process
- CancelRecording(); Cancels the recording process
- RunBiofeedback(Name); Runs Biofeedback of a given name
- OpenActivity([Name[, FolderPath[, FilenameFilter]]]);Opens an 'Open File' window. 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, an 'OpenFile' window 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 argument to filter when scanning for source files. To use 'FilenameFilter' pass an empty string for the Name argument, and either an empty string or a valid path for FolderPath.

-	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 left, right, top, and 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. 'FilenameFilter' is an optional argument to filter when scanning for source files. To use 'FilenameFilter' pass an empty string for the Name argument, and either an empty string or a valid path for FolderPath.
-	OpenActivityHidden(Name[, FolderPath]);	Opens an activity invisibly and with any warning messages suspended. If applicable, it will automatically reprocess any video files in the background.
-	SaveActivity();	Saves an open activity using the default name.
-	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. All arguments must be surrounded by quotation marks.
-	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.

- CloseAllActivities(); — Closes all currently open Activities regardless of whether they are visible, and does not alert the user – even if they are modified.
- 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.
- ImportC3DFile([Hidden]); — Imports C3D files. There is an optional Hidden argument, which if TRUE will bypass the Static C3D import setup and will immediately proceed to selecting dynamic C3D files for importing.
- ImportC3DFileWithoutStaticFile ([Name[, FolderPath]]); — 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.
- CloseC3DActivity(); — Closes C3D file.
- ToggleLooping(); — Toggles looping playback of an open activity that contains this script.
- Play(); — Runs playback of an open activity that contains this script
- Pause(); — Pauses playback an open activity that contains this script

- Stop(); — Stops playback of an open activity that contains this script
- AboutMotionMonitor(); — Opens 'About MotionMonitor' dialog

Script Analysis Functions

- SetRepair(Var,Enabled,MaxInterval); — Enables a data repair function for a selected variable. The first argument is the predefined variable to be interpolated. Second argument is a Boolean for the enabled status. Third argument sets the maximum frame interval for which the data will be interpolated.
- SetButterworthFilter(Var,Enabled,Freq); — Enables a Butterworth filter for a selected variable. Second argument is a Boolean for the enabled status. Final argument sets the cut-off frequency for the filter in Hz.
- SetChebyshevFilter(Var,Enabled,Freq); — Enables a Chebyshev filter for a selected variable. Second argument is a Boolean for the enabled status. Final argument sets the cut-off frequency for the filter in Hz.
- SetFFTLowpassFilter(Var,Enabled,Freq,Rolloff); — Enables a FFT low pass filter for a selected variable. The first argument is the predefined variable to be filtered. Second argument is a Boolean for the enabled status. 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.
- SetFFTHighpassFilter(Var,Enabled,Freq,Rolloff); — Enables a FFT high pass filter for a selected variable. The first argument is the predefined variable to be filtered. Second argument is a Boolean for the enabled status. 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.

-	SetFFTNotchFilter(Var,Index,Enabled,Freq,Width,Rolloff);	Sets a notch filter in recorded activities. The first argument is the predefined variable 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.
-	EditFilterSettings(Var);	Opens the Edit Filter Settings dialog for a specified variable
-	ShowGraph(Name);	Displays the graph with graph name, Name
-	HideGraph(Name);	Hides the graph with graph name, Name
-	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
-	RenameGraphPlot(GraphName,OldPlotName,NewPlotName);	Renames a plot in a specified graph. 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 plot will be renamed as
-	ShowAnimation(Name);	Displays the Animation with animation name, Name
-	HideAnimation(Name);	Hides the Animation with animation name, Name
-	ShowMusculoskeletalAnimation(Name);	Displays the Musculoskeletal Animation with animation name, Name
-	HideMuscusloskeletalAnimation(Name);	Hides the Musculoskeletal Animation with animation name, Name
-	ShowVideo(Name);	Displays the video with camera name, Name
-	HideVideo(Name);	Hides the video with camera name, Name

- GenerateReport(Name [, AnalysisName]); Generates Report. 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.
- GenerateReportAs(Name, FileName [, AnalysisName]); Generates Report with a specified name. 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.
- SendReportToKinduct(Name[, AnalysisName]); Generates Report and sends it to Kinduct using the Parameters specified within the 'Kinduct Settings' dialog for the Report. 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.
- 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.
- Spawn (ExePath[, Arg]); Spawns an executable in a specified file path (String). For example, this can be used to spawn Excel.

-	Excel(FilePath, FunctionName);	Opens the specified Excel workbook file, FilePath, in Excel and executes the Macro, FunctionName.
-	OpenAnalysis([Name]);	Opens Analysis File, Name is a string or string variable containing the name of the analysis file to open
-	OpenFilterExclusiveAnalysis([Name]);	When generating a report, this operator allows you to apply an analysis file to the export, without overwriting the filter settings saved in the activity. There is an optional filename argument, which if absent will cause a file-browser dialog to open.
-	OpenAnalysisInActivity(ActivityName[, AnalysisName]);	Should be called 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.
-	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.
-	MergeFilterExclusiveAnalysis([Name]);	When generating a report, this operator allows you to merge an analysis file to the export, without overwriting the filter settings saved in the activity. There is an optional filename argument, which if absent will cause a file-browser dialog to open.
-	SaveAnalysisAs();	Opens 'Save Analysis File' dialog

- 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 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 file. Any of these parameters will be skipped or use their default values if you pass a zero or an empty string.
  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.
- 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 file. Any of these parameters will be skipped or use their default values if you pass an empty string.

Appendix B

Working with position vectors and non-position or directional/offset vectors

Position vectors, which have an absolute location in space, like GPS coordinates. An example includes an X, Y, Z position coordinate.

Non Position or Directional/offset vectors, which don't have an absolute location in space, like the direction of North or the offset of a stylus handle to its tip. Additional examples include joint force and moment vectors.

The MotionMonitor xGen handles the appropriate computations for these data when their sources are known, such as when variables are defined through Subject drop-lists or directly form a hardware device. When custom variables are defined, is when care must taken to ensure proper variable expression definitions. In most cases, both types of vectors are handled similarly and use the same operators, however, they must be treated differently with rel() and * operators. When a position vector is made relative to a reference frame, the reference frame's position should affect the result. If the reference frame is close to the position, the resulting vector magnitude should be small; if far away, large. When a direction/offset vector is made relative to a reference frame, the reference frame's position should *not* affect the result. Only the orientation of the reference frame is relevant in that case.

For position vectors, you want to use the axes-based operators: relpos(Pos, A) or rel(Pos, A) and A * Pos. For non-position or directional/offset vectors, you want to use the rotation-based operators: rel(V, R) and R * V or axes-based operator reldir(V, A).

Examples:

reldir(V, A) or rel(V,R)

Calculating the direction of the world up vector relative to a segment axis system: reldir(WorldUp, SegmentAxes) or rel(WorldUp, SegmentAxes.Ori)

relpos(Pos, A) or rel(Pos, A)

Taking the digitized location of a bony landmark and making it relative to a segment sensor: relpos(LandmarkPos, SensorAxes) or rel(LandmarkPos, SensorAxes)

rel(A1, A2) Calculating the forearm axes relative to the upper arm axes: rel(ForearmAxes, UpperArmAxes)

Examples Cont.

rel(R1, R2)	Calculating the forearm orientation relative to the upper arm orientation: rel(ForearmOri, UpperarmOri)
rel(Ori, A)	Calculating the forearm orientation relative to the upper arm axes: rel(ForearmOri, UpperArmAxes)
R * V	Calculating the current direction of a segment's anterior axis, whose direction relative to the sensor is known: SensorAxes.Ori * AnteriorAxisRelSensor
A * Pos	Calculating the absolute position of a bony landmark, whose placement relative to the segment sensor is known: SensorAxes * LandmarkPosRelSensor
A1 * A2	Calculating the current segment Axes position and orientation, whose placement relative to the segment sensor is known: SensorAxes * SegmentAxesRelSensor
R1 * R2	Calculating the current orientation for an Axes, whose orientation relative to the segment sensor is known: SensorAxes.Ori * SensorAxesRelSensor.Ori

Note: When taking a Vector, Rotation or Axes from reference frame "X" and placing it in the world reference frame, the results will only be correct if the Vector, Rotation or Axes are actually in reference frame "X" to begin with.

Appendix C

References

Orthopaedic Angles:
These angles are projection angles and correspond to standard goniometric measurements as described in Norkin & White, A Guide to Goniometric Measurement.

Linear Velocity and Acceleration:
Computed using a 3 point parabolic fit, as described in Biomechanics and Motor Control of Human Motion, 4th edition by David A. Winter.

Angular Velocity and Acceleration:
Computed using a similar method as with linear velocity and acceleration. For velocity, the change in orientation from the first to the third value is calculated and converted into its axis-angle representation, divided by time and scaled. The computation of acceleration uses a 3 point method where the difference is divided by the time interval.

Euler or Carden Angles:
Euler sequence solutions are described in Introduction to Robotics, Mechanics and Control, Appendix B by John J. Craig. For sequences with multiple solutions, the solution where the sum of all the first angle magnitudes across the entire data series is the smallest is selected.

Helical Axes:
Spoor and Veldpaus. “Rigid Body Motion Calculated from Spatial Co-ordinates of Markers: Technical Note”, 1980. Journal of Biomechanics, Volume 13: 391-3.

Grood & Suntay Angles:
Grood E., Suntay W. "A Joint Coordinate system for the Clinical Description of Three Dimensional Motions: Application to the Knee", Trans. ASME 105:136-144, 1983.

Joint Forces & Moments:
Computed in a linked chain model as described in Appendix A of "The Influence of Dynamic Factors on Tri-axial Net Muscular Moments at the L5/S1 Joint During Asymmetrical Lifting and Lowering" by Gagnon and Gagnon in Biomechanics Vol. 25 No.8 pp. 891-901. The model uses either a "top-down" or "bottom-up" approach.

Linear & Angular Momentum:
Linear and Angular Momentum take into account the orientation of the reference frame, but do not subtract out its velocity.

Energetics:
David A. Winter, Biomechanics and Motor Control of Human Motion, 4th edition. Wiley, 2009, pg 156.

Shoulder Joint “Functional Method” (centre transformation technique (CTT)):
Ehrig, R.M, Taylor, W.R, Duda, G.N., Heller, M.O., 2006. A survey of formal methods for determining the centre of rotation of ball joints. Journal of Biomechanics 39(15), 2798-2809.

Shoulder Joint “Meskers’ Method”:
Meskers, C.G.M., Helm, F.C.T.v.d., Rozendaal, L.A., Rozing, P.M., 1998. In vivo estimation of the glenohumeral joint rotation center from scapular bony landmarks by linear regression. Journal of Biomechanics 31, 93-96.

Hip Joint "Functional Method" (centre transformation technique (CTT)):
Ehrig, R.M, Taylor, W.R, Duda, G.N., Heller, M.O., 2006. A survey of formal methods for determining the centre of rotation of ball joints. Journal of Biomechanics 39(15), 2798-2809.

Hip Joint "Bell Method":
Bell, A.L., Pedersen, D.R., Brand, R.A., 1990. A comparison of the accuracy of several hip center location prediction methods. Journal of Biomechanics 23, 617–662.

Hip Joint "Davis Method":
Davis, R.B., Ounpuu, S., Tyburski, D., Gage, J.R., 1991. A gait analysis data collection and reduction technique. Human Movement Science 10, 171–178.

Stylus Tip Calculation
Computing Stylus Tip Coordinates as described in Appendix III: Application Notes of Ascension Technology Corporation 3D Guidance trakSTAR Installation and Operation Guide Rev D. 2019.

Anthropometrics "Zatsiorsky":
Zatsiorsky, V., and Seluyanov, V. (1983). The mass and inertia characteristics of the main segments of the human body. In H. Hatsui and K Kobayashi (eds.) Biomechanics VIII-B, Human Kinetics, Champaign, IL, 1152-1159.

Anthopometrics "Dempster":
Segment Mass & Center of Mass data:
Winter, D.A, Biomechanics and Motor Control of Human Movement, Second edition. John Wiley & Sons, Inc., Toronto, 1990, 56-57.

Antrhropometrics "Radius of Gyration":
Chandler, R.F., Clauser, C.E., McConvill, J.P., Reynolds, H.M, and Young, J.W. (1975). Investigation of Inertial Properties of the Human Body, AMRL-TR-74-137, Aerospace Medical Research Laboratories, Ohio.

Zatsiorsky, V., and Seluyanov, V. (1983). The mass and inertia characteristics of the main segments of the human body. In H. Hatsui and K Kobayashi (eds.) Biomechanics VIII-B, Human Kinetics, Champaign, IL, 1155.

Default Vertebral Column Joint Center Offsets:
Chaffin, D.B. & Andersson, G.B.J., Occupational Biomechanics, Second edition. John Wiley & Sons, Inc., 1991, pg 67.