GaitSym 2014 Config Reference Manual

Gait Simulation using Multibody Dynamics

William I. Sellers
Animal Simulation Laboratory, Faculty of Life Sciences, University of Manchester

GaitSym 2014 Config Reference Manual

Gait Simulation using Multibody Dynamics

INTRODUCTION

The main documentation for GaitSym 2014 is the GaitSym 2013 manual since the vast majority of that document is still current. The look and feel of the program has changed to some extent but this is largely cosmetic and is not detailed here. This document details the config file since there are a few new options and changes.

THE CONFIG FILE

This is where the complexity of the program is hidden and it is recommended that you edit an existing file whilst you get used to the format. What follows is a detailed description of the elements that make up the file but for the full details you may need to look at the source code. The documentation does not include several beta features since these may not have been properly tested and the interface has not been fully resolved. The coordinate system used is probably not important for the simulation but the graphical output makes much more sense if a right handed coordinate system is used with positive X being forward and positive Z being up. However there is an option in the preferences to set Y as up and this will help if your data is set up that way. The unit system is also probably not important as long as it is consistent. It has only been tested using SI units. The parser is very fussy and does not give particularly helpful error messages. However "-d 17", the XMLDebug option, is helpful at telling you how far the parser has got before failing. This option only works for the command line version of the software. There is a status window in the GUI version that shows the output of the parser and it can certainly be useful to see which was the last line that was parsed correctly. However there are still file format errors that will lead to the application crashing and this will be lost in the GUI version. Fortunately once the models have been constructed, most edits to the config file are fairly minor and errors should not be particularly common.

IMPORTANT: The documentation here is relatively complete. However to get a complete and accurate idea of what is going on you may need to look at the source code. All XML parsing happens in Simulation.cpp.

The basic format of the config file is as follows:

```
<?xml version="1.0"?>
<GAITSYMODE>
<STATE options />
<IOCONTROL options />
<GLOBAL options />
<ENVIRONMENT options />
<BODY options />
<JOINT options />
<GEOM options />
<MUSCLE options />
<DATATARGET options />
<REPORTER options />
<CONTROLLER options />
<DRIVER options />
</GAITSYMODE>
```

Options all take the form keyword="value" where value can be some mix of text and numbers as specified in the descriptions. Boolean values can be written as "true", "false", "1" or "0". Double precision numbers can be specified in normal decimal notation. Text is any valid XML text.

<STATE options />

Unused element produced to record some information about when in the simulation a model state file was produced.

<IOCONTROL options />

Optional. Used to specify the input styles wanted. Possible options are:

OldStyleInputs="boolean"

If set to true, this requires subsequent statements to use old style inputs. Can appear multiple times in the XML file. This option is deprecated and likely to be removed in future versions. *OldStyleOutputs* is no longer available.

SanityCheckLeft="string"
SanityCheckRight="string"
SanityCheckAxis="string"

Components of the body ID string that are used for left and right hand side bodies. If these are defined then any body IDs that differ by only these strings are compared for sanity checking. E.g. If *SanityCheckLeft="Left"* and *SanityCheckRight="Right"* then the bodies "LeftThigh" and "RightThigh" will be checked against each other. This checking compares the locations, mass properties, joints and muscle attachments mirrored across the *SanityCheckAxis* axes which is specified as *"x"*, *"y"* or *"z"*. This is only useful for the initial model building when the model is symmetrically defined. It has no effect on the model generated and simply produces warning messages in the status window if sanity check errors are detected.

<GLOBAL options />

Required. Specifies values to control the overall simulation. These are generally required but look at the example files to see suitable values. Possible options are:

IntegrationStepSize="double"

Sets the integration step size for the OpenDE integrator. Typical value is 1e-4 seconds.

GravityVector="double double double"

Sets the value and direction of gravity (X, Y, Z). Typical value is (0, -9.81, 0).

ERP="double"
CFM="double"
SpringConstant="double"
DampingConstant="double"

ERP is the error reduction parameter and is used to control how much force is applied to the model when there is error in the joints - which there always will be once the simulation has started. *CFM* is the constraint force mixing which, when non-zero, allows constraints to be violated to a limited extent. These values together specify the global damping and stiffness of the simulation system. They are very important for simulation numerical stability. Typical values are *ERP* = 0.2 and *CFM* = 1e-10. Alternatively you can specify *SpringConstant* and *DampingConstant* instead and suitable values for ERP and CFM will be calculated using the following formula:

$$ERP = IntegrationStepSize \times SpringConstant / (IntegrationStepSize \times SpringConstant + DampingConstant)$$

$$CFM = 1 / (IntegrationStepSize \times SpringConstant + DampingConstant)$$

The OpenDE documentation has more information on how exactly these values affect the simulation and on how to choose suitable values. However the recommended values generally work reasonably well.

StepType="string"

OpenDE has 2 different integrators of varying speed and accuracy: *"WorldStep"* and *"QuickStep"*. You probably want *"WorldStep"* which is the default and most accurate, however *"QuickStep"* can be very much faster to calculate.. See the OpenDE documentation for more information.

ContactMaxCorrectingVel="double"

OpenDE is a velocity based physics simulator so that all constraint satisfaction happens in the velocity domain. This value sets the maximum velocity that the simulator can generate when correcting contact interpenetration errors. Low values will lead to somewhat softer contacts where it may take several integration steps to correct for interpenetration errors. High values can cause very large forces.

ContactSurfaceLayer="double"

If set to a non-zero value this will allow this amount of contact interpenetration to occur before the contact comes to rest. Setting a small value is useful since it can reduce jittering problems due to contacts being repeatedly made and broken.

AllowInternalCollisions="boolean"

Controls whether body mounted contacts can only interact with the static environment or with each other too.

AllowConnectedCollisions="boolean"

Controls whether collisions are allowed between bodies connected by joints. This is useful of collisions are being used to apply joint limits.

BMR="double"

The metabolic rate to be added to produce gross estimates of metabolic cost. Set to 0 if net metabolic costs are wanted.

TimeLimit="double"

The simulation time limit. Set to 0 if no time limited wanted. This value can be overridden by the GUI.

MetabolicEnergyLimit="double"

The simulation metabolic energy limit. Set to 0 if no metabolic energy limited wanted.

MechanicalEnergyLimit="double"

The simulation mechanical energy limit. Set to 0 if no mechanical energy limited wanted. Mechanical energy is the mechanical work done by the muscle-tendon unit combined.

FitnessType="string"

The simulation generates a score at the end of the simulation for use in global optimisation experiments. This can either be *"DistanceTravelled"* if this is the quantity of interest, *"KinematicMatch"* if matching to existing kinematics is wanted, or *"KinematicMatchMiniMax"* if kinematic matching should use a minimax algorithm where only the largest error at each timestep is considered rather than the normal sum of errors approach where the sum of all errors is used for the score.

DistanceTravelledBodyID="string"

When calculating distance travelled the system uses the position of the centre of mass of a specified body and this is where it is specified. A common cause of crashes is if this ID string is not defined later on in the file.

There are also a number of optional GLOBAL options that are generally not very useful and may get removed in future:

OutputModelStateFilename="string"
OutputModelStateAtTime="double"
OutputModelStateAtCycle="double"
MungeModelState="boolean"

These values control the output of a model state file which records the state of the model at a particular time during the simulation. This state file can be read in for subsequent simulation. If

MungeModelState is *true* then the position of the *DistanceTravelledBodyID* is reset to zero, and all the other bodies are moved to match.

OutputKinematicsFile="string"
InputKinematicsFile="string"

These values control whether the raw kinematics of the bodies are written to an output file or read from an output file rather than being calculated by the simulation.

YUp="boolean"

If *true* then Y rather than Z is considered the up direction. This is overwritten by the Preferences pane options in the GUI version.

<INTERFACE options \>

These options control the appearance of the graphical interface. They do not affect the simulation process itself and are generally optional. Most of the values used by earlier versions of GaitSym now have no effect and are set directly in the Preferences pane options in the GUI version.

TrackBodyID="string"

The graphical output can track the X coordinate of a specific body by setting this to its ID. If the ID does not exist then the program will crash gracelessly.

<ENVIRONMENT options />

Optional but essential if you want a floor.

Plane=" double double double double"/>

Specifies the plane used as the ground surface (a, b, c, d). The plane equation is:

$$a \times x + b \times y + c \times z = d.$$

The plane's normal vector is (a, b, c), and it must have length 1. A standard Z=0 plane is specified as:

Plane="0 0 1 0".

<BODY options />

Optional but you will not have a model if you do not specify some bodies!

ID="string"

Required. Name of the element: must be unique.

GraphicFile="name"

Optional if *Density* not used. Name of OBJ file associated with this body. Can be a complete path specification or just a file name if the -g option is used. For the GUI version this path will always be appended to the default graphics folder path so absolute paths will not work as expected.

Scale="double"
Scale="double double double"

Required. Allows the graphic file to be scaled when it is read in to allow standard sized components that can be resized to match an individual. Either a single uniform scale value can be given or alternatively separate scale values for the x, y and z values.

Offset="double double double"

Required. By default the graphic file should use the centre of mass as the origin. This allows it to be offset by X, Y, Z if this is not the case.

Clockwise="boolean"

Optional. Defaults to *false*. If set to *true* the winding of the OBJ file will be reversed. Use this if you see display artefacts that can be fixed with the *BadMesh* setting since it usually means that the triangles in your objects are the wrong way round for GaitSym and fixing it up this way will mean that the display is quicker since each triangle is drawn twice when the *BadMesh* option is used. This option is also important if you are using the *Density* setting to calculate the mass properties. Calculating the mass properties requires a well formed shape with the correct winding. If your shapes will not load when using the *Density* option, or if the mass properties do not appear to be correct then try reversing the winding with this option. Not all CAD programs produce meshes with consistent winding.

BadMesh="boolean"

Optional. Defaults to *false*. If set to *true* then all triangles in the mesh are drawn twice: once as specified, and once with the winding reversed. This is useful if the mesh consists of triangles with mixed winding or if it is a single sided mesh rather than an enclosed shape. *BadMesh* will not allow the volume to be calculated from a poorly formed mesh but it will make these meshes look better. The performance penalty on a modern graphics card is not normally a problem, however transparency make not work as well.

VerticesAsSpheresRadius="double"

Optional. If specified, the body is drawn by constructing a sphere of the supplied radius around each vertex. This allows point clouds to be visualised directly. However it is generally very slow unless the point clouds are small.

Mass="double"

Required. The mass of the segment. Can be calculated from the *Density*.

MOI="double double double double double double"

Required. The elements of the inertial tensor around the centre of mass specified as I11, I22, I33, I12, I13, I23 where the matrix is:

```
[ I11 I12 I13 ]
[ I12 I22 I23 ]
[ I13 I23 I33 ]
```

Density="double"

Required. The program can optionally calculate the mass, inertial tensor and position of centre of mass by specifying a *Density* and making sure that the *GraphicFile* is a well-formed, closed triangular mesh. It copes with irregular, complex shapes but the density needs to be uniform. Set to "-1" if not being used. If the origin of the *GraphicFile* does not match the centre of mass then the position and offset are altered to accommodate this. Note you may have to alter the winding of your graphics files with the *Clockwise* option if this function does not work. Detailed instructions for using this ability are given in the section on Model Creation Workflow.

Position="string"

Required. Specify the position of the body. The origin of a body is its centre of mass. This cannot currently be altered. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of centre of mass (this is the only option if *OldStyleInputs="true"*)

- *"BodyID X Y Z"*: coordinate location of centre of mass using *BodyID* coordinate system, *BodyID* can be *World.*
- *"BodyID X1 Y1 Z1 X2 Y2 Z2"*: position such that *X1,Y1,Z1* on *BodyID* in *BodyID* local coordinates has same world coordinates as *X2,Y2,Z2* on the current body in current body coordinates. This is more useful that it might sound at first since it allows a body to be positioned by a joint constraint allowing relative position specification of jointed structures.

Quaternion="string"

Required. Specify the orientation of the body. *"string"* is an orientation string and can use the following conventions when *OldStyleInputs="false"*:

- *"W X Y Z"*: quaternion specifying the rotation of the body in world coordinates (this is the only option if *OldStyleInputs="true"*)
- *"BodyID W X Y Z"*: quaternion specifying the rotation of the body using *BodyID* coordinate system, *BodyID* can be *World.*
- Note. The quaternion can also be specified as an angle axis construction by putting a "d" or a "r" immediately after the W parameter with no intervening space. The W parameter then represents an angle in degrees ("d") or radians ("r") and the XYZ components are the axis of the rotation.

LinearVelocity="double double double"

Required. Specify the linear velocity of the centre of mass of the body (XV, YV, ZV) in world coordinates.

AngularVelocity="double double double"

Required. Specify the angular velocity of the centre of mass of the body (RXV, RYV, RZV) in world coordinates.

PositionLowBound="double double double"
PositionHighBound="double double double"
LinearVelocityLowBound="double double double"
LinearVelocityHighBound="double double double"

Optional. Specify limits for position and linear velocity. These are used to abort the simulation when it has gone crazy or to control the forward velocity of global optimisations.

LinearDamping="double"
AngularDamping="double"
LinearDampingThreshold="double"
AngularDampingThreshold="double"
MaxAngularSpeed="double"

Optional. If these values are set then body level damping is activated. This applies a force to the body that is proportional to the speed in magnitude but acts in the opposite direction. It acts rather like dynamic friction and can be used to aid simulation stability. It needs to be used with care in any simulation with aerial phases though since it will tend to support a falling body. The linear functions act on the basis of linear velocity and the angular functions depend on rotational velocity. The *MaxAngularSpeed* acts to limit the angular speed used to satisfy the model′s constraints.

<JOINT options/>

Specify the joints in the model. Optional.

ID="string"

Required. Name of the element: must be unique.

Type="string"

Required. The type of joint required. Options are *Hinge*, *FloatingHinge, Ball*, *Fixed*, *Universal*, *AMotor*, *Slider*. All joints connect two bodies identified by their *ID*. One body can be *World* if the joint is connecting to the environment.

Body1ID="string"
Body2ID="string"

Required. Set the bodies to be connected. These must already have been defined in the file, or can be specified as *World* if the joint is with the fixed environment.

Other options depend on the type of joint.

Hinge Joints

Hinge joints allow a single rotational degree of freedom between two bodies rather like the hinge on a door. Joints like the elbow or the knee are typically modeled as hinge joints.

ParamLoStop="double"
ParamHiStop="double"

Optional. Set the minimum and maximum rotations about the joint. If not set then the joint can rotate infinitely like an axle of a wheel.

HiStopTorqueLimit="double"
LoStopTorqueLimit="double"
StopTorqueWindow="integer

Optional. If set then the program calculates the torque generated by the stops and if it exceeds the specified limit, it aborts the program. *StopTorqueWindow* specifies the width of the moving average used to calculate the torque. This is useful because instantaneous values of torques are often not very useful (or very accurate) but time averaged values are.

HingeAnchor="string"

Required. Specify the location of the common anchor of the hinge joint. This is specified for the posed model. The model cannot be posed after the joint position has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate location of anchor (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of anchor using *BodyID* coordinate system, *BodyID* can be *World.*

HingeAxis="string"

Required. Specify the direction of the hinge axis. This is specified for the posed model. The model cannot be posed after the joint axis has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World.*

StartAngleReference="double"

If the starting pose is the reference pose then this should be set to 0, however if the starting pose is not the desired reference pose then a value representing this initial angle can be set. So for a human where the anatomical position is the reference position, if the starting pose has a 0.5 radian joint angle then this value should be set to 0.5.

Note. Angles for *ParamLoStop*, *ParamHiStop*, *StartAngleReference* default to radians. However they can be specified in degrees if a "d" is suffixed to the number. "r" can optionally be suffixed for radians but it is not necessary.

StopCFM="double"
StopERP="double"
StopSpringConstant="double"
StopDampingConstant="double"

Optional. Normally the joint limits use the global *CFM* and *ERP* (set in GLOBAL) values but these can be overridden on a joint by joint basis for hinge joints if this is required. These can be used to emulate a rotational spring.

StopBounce="boolean"

Optional. Can be set to true if bouncy joint stops are wanted.

FloatingHinge Joints

Floating hinges are simply joints that restrict all rotation to a single axis but do not affect linear movements, thus allowing a single rotational degree of freedom and 3 linear degrees of freedom. They are useful sky-hooks when you want to stop a body falling over. For example in my 2.5D simulations I use a *FloatingHingeJoint* to only allow the trunk to topple over forwards and backwards but not sideways, nor to rotate around a vertical axis. This effectively limits the model to 2D but allows the muscle paths and thus lengths to be fully 3D which is very useful.

The following parameters are available:

ParamLoStop="double"
ParamHiStop="double"
FloatingHingeAxis="string"
StartAngleReference="double

These definitions are exactly the same as the equivalently named hinge joint options. *FloatingHingeAxis* is equivalent to *HingeAxis* and is the only required parameter.

Ball Joints

Ball joints allow 3 rotational degrees of freedom between two bodies but restrict linear movement. A mechanical example would be magnetic compass mounted in a gimbal. Ball joints are commonly used to model hip and shoulder joints. One of the difficulties is because ball joints do not restrict any rotational degrees of freedom it is actually difficult to specify the joint angle in a way that is both easy to interpret and mathematically general. This is particularly troublesome when it comes to specifying joint limits.

The options for *Ball* joints are as follows:

BallAnchor="string"

Required. Specify the location of the common anchor of the ball joint. This is specified for the posed model. The model cannot be posed after the joint position has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate location of anchor (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of anchor using *BodyID* coordinate system, *BodyID* can be *World*.

AxisMode="string"

Optional. If using stops then ball joints can be defined in two ways: using system fixed Euler axes (*AxisMode="FixedEuler"*) or using user supplied Euler axes (*AxisMode="UserEuler"*). When using *FixedEuler* only 2 axes need to be defined since the other axis is generated automatically as perpendicular to the the two supplied axes, and of the two supplied axes, *Axis0* must be relative to *Body1* and *Axis2* is relative to *Body2* (and *Axis1* is provided by the system). The other restriction for *FixedEuler* is that the initial position must mean that the Euler angles are zero.

Axis0="double double double"
Axis1="double double double"
Axis2="double double double"

These are the underlying Euler axes for the joint. For *FixedEuler* only *Axis0* and *Axis2* are needed and *Axis0* must be relative to *Body1* and *Axis2* is relative to *Body2*. For *UserEuler* all axes are specified relative to *body1*.

> *ParamLoStop0="double"*
> *ParamHiStop0="double"*
> *ParamLoStop1="double"*
> *ParamHiStop1="double"*
> *ParamLoStop2="double"*
> *ParamHiStop2="double"*

Optional. Stops can be specified for each of the Euler angles used. These must be in the range of $-\pi/2$ to $+\pi/2$ and in practice can be very difficult to set at sensible values. This is an intrinsic problem with Euler angles: angles that are close spatially do no necessarily have close Euler angles and this can lead to odd effects.

Note. Angles for *ParamLoStopX*, *ParamHiStopX* default to radians. However they can be specified in degrees if a "d" is suffixed to the number. "r" can optionally be suffixed for radians but it is not necessary.

WARNING: The joint limits on Ball joints should be considered experimental and their implementation may change in future. It is actually very difficult to set up the correct axes and get them to work as expected. Joint limits can also be specified by attaching multiple *AMotor* joints to any other non-stopped joint and this may be preferable.

Fixed Joints

Fixed joints have no additional options. They simply fix two bodies in the pose specified. It is numerically more stable to combine two bodies into a single body than to use a fixed joint but it mostly seems to work OK and it is often very convenient. So for example in a full body model the forelimbs could be attached with fixed joints if forelimb movement was not required to avoid having to recalculate the mass properties of the torso with attached forelimbs.

Universal Joints

Universal joints can be thought of as two hinge joints attached together sharing a common reference point. They allow two degrees of rotational freedom and are thus often used for joints such as the wrist or ankle.

They take the following options:

UniversalAnchor="string"

Required. Specify the location of the common anchor of the universal joint. This is specified for the posed model. The model cannot be posed after the joint position has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate location of anchor (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of anchor using *BodyID* coordinate system, *BodyID* can be *World*.

UniversalAxis1="string"
UniversalAxis2="string"

Required. Specify the directions of the universal joint axes. These are specified for the posed model. The model cannot be posed after the joint axis has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World*.

Stops are not provided for universal joints but these are relatively straightforward to provide by aligning *AMotor* joints to the hinge axes.

AMotor Joint

An AMotor (short for angular motor) is rather unlike the other joints available in that it provides a torque source around a rotational axis between two bodies. It does not remove any degrees of freedom so it can be used in conjunction with other joints to provide functions that they do not natively provide. This includes range of motion stops and friction. The joint can also be used to specify a target angular velocity so this joint can be used to emulate a motor that can drive the model.

It takes the following options

Axis="string"

Required. Specify the direction of the AMotor axis. This is specified for the posed model. The model cannot be posed after the joint axis has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World.*

> *ParamLoStop="double"*
> *ParamHiStop="double"*

Optional. If set these restrict the movement around the axis to the given angular range. These stops are relative to the absolute angle between the two bodies unlike the case with hinge joints where they relate to the initial pose.

Note. Angles for *ParamLoStop*, *ParamHiStop* default to radians. However they can be specified in degrees if a "d" is suffixed to the number. "r" can optionally be suffixed for radians but it is not necessary.

> *MaxTorque="double"*

Optional. If specified this is the maximum torque that the motor can generate when trying to match a velocity target.

> *TargetVelocity="double"*

This is the target velocity that the AMotor will try and achieve at the next integration step size. It will apply up to a maximum of ±*MaxTorque* to achieve this velocity. One common use for this is to set the *TargetVelocity* to zero and then up to *MaxTorque* will be applied try and stop the relative motion around the axis. With a suitably chosen value for *MaxTorque* this can be used to emulate a Coulomb friction model.

Slider Joints

Slider joints allow a single rotational degree of freedom between two bodies rather like a slide rule. This joint is useful for modelling moveable elements in the environment.

> *ParamLoStop="double"*
> *ParamHiStop="double"*

Optional. Set the minimum and maximum linear distances that the slider can move. If not set then the joint can slide infinitely.

SliderAxis="string"

Required. Specify the direction of the slider axis. This is specified for the posed model. The model cannot be posed after the joint axis has been set. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World.*

StartDistanceReference="double"

If the starting pose is the reference pose then this should be set to 0, however if the starting pose is not the desired reference pose then a value representing the initial distance between the two bodies connected by the slider joint can be set. This affects the meaning of the *ParamLoStop* and *ParamHiStop* values.

StopCFM="double"
StopERP="double"
StopSpringConstant="double"
StopDampingConstant="double"

Optional. Normally the joint limits use the global *CFM* and *ERP* (set in GLOBAL) values but these can be overridden on a joint by joint basis for hinge joints if this is required. These can be used to emulate a linear spring.

StopBounce="boolean"

Optional. Can be set to true if bouncy joint stops are wanted.

<GEOM options />

Optional. Specify a contact geometry. These contacts can interact with the environment and with each other. However when there are multiple geoms on a single body make sure they do not overlap since this can cause numerical difficulties.

ID="string"

Required. Name of the element: must be unique.

Type"string"

Required. The type of geom required. Options are *Sphere*, *CappedCylinder* and *Box*.

BodyID="string"

Required. Set the body the geom is attached to. These must already have been defined in the file.

Position="string"

Required. Specify the position of the *geom*. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of geom (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of geom using *BodyID* coordinate system, *BodyID* can be *World*.

Quaternion="string"

Required. Specify the orientation of the geom. *"string"* is an orientation string and can use the following conventions when *OldStyleInputs="false"*:

- *"W X Y Z"*: quaternion specifying the rotation of the geom in world coordinates (this is the only option if *OldStyleInputs="true"*)
- *"BodyID W X Y Z"*: quaternion specifying the rotation of the geom using *BodyID* coordinate system, *BodyID* can be *World*.
- Note. The quaternion can also be specified as an angle axis construction by putting a ″d″ or a ″r″ immediately after the W parameter with no intervening space. The W parameter then represents an angle in degrees (″d″) or radians (″r″) and the XYZ components are the axis of the rotation.

CappedCylinder geoms have two other self-explanatory obligatory parameters:

Radius="double"
Length="double

Sphere geoms only have one obligatory parameter:

Radius="double"

Box geoms only have one obligatory parameter (specified as x, y, z):

Dimensions="double double double"

Note. *CappedCylinder*s are now called Capsules in the OpenDE documentation. They are defined with the axis along the Z axis and the *Quaternion* option is used to reorient them.

All geoms have a variety of option parameters that control the physics of the interaction. These are as follows:

ContactSoftERP="double"
ContactSoftCFM="double"
SpringConstant="double"
DampingConstant="double"

ContactSoftERP and *ContactSoftCFM* together specify the local damping and stiffness of the contacts created by this *geom*. They can be very important for simulation numerical stability. Their function is exactly the same as the global *ERP* and *CFM* values described in the GLOBAL section. Alternatively *SpringConstant* and *DampingConstant* values can be specified and these are converted to *ERP* and *CFM* as described in the GLOBAL section. *SpringConstant* and *ContactSoftERP* can also be specified as a pair, but not *ContactSoftCFM* and *DampingConstant*. It may be desirable for contact *geoms* to have quite low values for *SpringConstant* if a soft contact is desired.

Bounce="double"

Contacts vary in the amount of momentum lost in the collision. If the contacts are bouncy *(Bounce=1)* then no momentum should be lost. If they are non-bouncy *(Bounce=0)* then minimal momentum should be lost. The value can be set to any floating point value for interesting but largely unpredictable effects. Even with *Bounce* set to zero there is usually some momentum return since this parameter does not take into consideration any energy stored as elastic energy due to contact interpenetration

Mu="double"

This is the Coulomb model friction coefficient between the contacts. It can be set to *"infinity"* if no slip is wanted.

Abort="boolean"

Contacts can optionally abort the simulation if activated. This is often very useful for identifying situations such as where the model has fallen over, or for applying orientation limits.

<MUSCLE options />

Optional. Specify muscles that can apply forces to the model. Muscles consist of two parts that can be mixed and matched as required. The first part is the *Type* which is the model used for force generation and estimating metabolic cost and the second is the *Strap* which specifies how the path of the muscle is defined.

ID="string"

Required. Name of the element: must be unique.

Muscle Types

Type="string"

Required. Specifies the contraction and metabolic energy model. The options available are as follows. *MinettiAlexander* which is the basic contractile unit described in Minetti & Alexander 1977[1] but converted to a linear form. This version does not include any elastic elements so is numerically very robust and works well for slow movements where elastic storage is probably not too important. *MinettiAlexanderExtended* is the same contractile model but has series and parallel elastic elements added so it can be used in simulations where elastic energy storage is important. This is reasonably stable as implemented but very stiff springs can be troublesome. *MinettiAlexanderComplete* is an alternative implementation which is solved numerically rather than analytically. This makes it a little slower but it should be much better behaved. It also has optional damping in the spring elements, has length dependent force generation, and customisable activation rates and has most of the internal settings exposed. It is the recommended muscle model for new simulations, and *MinettiAlexanderExtended* should be considered deprecated. *UmbergerGerritsenMartin* is the contractile unit described in Umberger et al 2003[2]. It also has serial and parallel elastic elements and is a more traditional Hill style implementation. *DampedSpring* is a simple damped spring that can be

[1] Minetti, A. E. & Alexander, R. M. 1997 A theory of metabolic costs for bipedal gaits. Journal of Theoretical Biology 186, 467–476.

[2] Umberger, B. R., Gerritsen, K. G. M. & Martin, P. E. 2003 A model of human muscle energy expenditure. Computational Methods in Biomechanics and Biomedical Engineering 6, 99–111.

used to model a ligament, or a non-active muscle. Not all the parameters for these models are exposed in the config file and you may need to edit the source if you need to alter some of these values. Where possible I have tried to keep the same symbol names as in the original papers. The *UmbergerGerritsenMartin* implementation is based heavily on Fortran code kindly provided by Umberger.

MinettiAlexander

For the *MinettiAlexander* models the following options are required:

PCA="double"

Physiological cross section area of the muscle.

FibreLength="double"

Length of the muscle fibres.

ForcePerUnitArea="double"

The force per unit area that can be generated by the contractile element (stress).

VMaxFactor="double"

The maximum contractile speed of this muscle in lengths per second (strain rate).

ActivationK="double"

Effects the shape of the contraction curve. It is usually set to *0.17* (Minetti & Alexander 1977).

MinettiAlexanderExtended (deprecated)

For the *MinettiAlexanderExtended* models the following options are required:

PCA="double"

Physiological cross section area of the muscle.

FibreLength="double"

Length of the muscle fibres.

ForcePerUnitArea="double"

The force per unit area that can be generated by the contractile element (stress).

VMaxFactor="double"

The maximum contractile speed of this muscle in lengths per second (strain rate).

ActivationK="double"

Effects the shape of the contraction curve. It is usually set to *0.17* (Minetti & Alexander 1977).

TendonLength="double"

Length of the tendon. This value can be set to -1 which will set the value internally so the muscle is at resting length in the starting posture for the simulation. This is very handy because the exact length of a muscle-tendon unit can be difficult to calculate externally due to the wrapping operators that can be used.

SerialStrainAtFmax="double"

The serial elastic strain at the maximum force the muscle can generate.

ParallelStrainAtFmax="double"

The parallel elastic strain at the maximum force the muscle can generate. If this value is set to ≤0 then parallel elasticity is disabled.

ActivationKinetics="boolean"

Whether to include first order activation kinematics as specified by He 1991[3].

InitialFibreLength="double"

The initial fibre length used by the model. If this value is omitted or set to ≤0 then it is calculated internally to minimise the stored strain energy. This value is produced when the model's state is saved to improve the restart accuracy but its effect is generally small since when not specified (or disabled by setting to -1) the model will calculate a sensible value.

MinettiAlexanderComplete

For the *MinettiAlexanderComplete* models the following options are required:

[3] He, J., Levine, W. S. & Loeb, G. E. 1991 Feedback gains for correcting small perturbations to standing posture. IEEE Transactions on Automatic Control 36, 322-332.

PCA="double"

Physiological cross section area of the muscle.

FibreLength="double"

Length of the muscle fibres.

ForcePerUnitArea="double"

The force per unit area that can be generated by the contractile element (stress).

VMaxFactor="double"

The maximum contractile speed of this muscle in lengths per second (strain rate).

ActivationK="double"

Effects the shape of the contraction curve. It is usually set to *0.17* (Minetti & Alexander 1977).

Width="double"

The maximum length range of force production relative to the fibre length. Typical values are close to 1.

TendonLength="double"

Length of the tendon. This value can be set to -1 which will set the value internally so the muscle is at resting length in the starting posture for the simulation. This is very handy because the exact length of a muscle-tendon unit can be difficult to calculate externally due to the wrapping operators that can be used.

SerialStrainAtFmax="double"

The serial elastic strain at the maximum force the muscle can generate.

SerialStrainRateAtFmax="double"

The serial elastic strain rate that will lead to a damping force equal to the maximum force the muscle can generate. If set to zero then serial elastic damping is disabled.

SerialStrainModel="text"

Can be set to *"Linear"* or *"Square"* depending on whether the elastic elements should follow a linear or square curve for their length/tension relationship.

ParallelStrainAtFmax="double"

The parallel elastic strain at the maximum force the muscle can generate. If set to zero then parallel elasticity is disabled.

ParallelStrainRateAtFmax="double"

The parallel elastic strain rate that will lead to a damping force equal to the maximum force the muscle can generate. If set to zero then parallel elastic damping is disabled.

ParallelStrainModel="text"

Can be set to *"Linear"* or *"Square"* depending on whether the elastic elements should follow a linear or square curve for their length/tension relationship.

ActivationKinetics="boolean"

Optional. Sets whether to include first order activation kinematics as specified by He 1991[4]. If specified then the following values also need to be specified:

TActivationA="double"
TActivationB="double"
TDeactivationA="double"
TDeactivationB="double"

These are the parameters for the activation kinetics. Typical values are TActivationA="80e-3", TActivationB="0.47e-3", TDeactivationA="90e-3", and TDeactivationB="0.56e-3".

InitialFibreLength="double"

Optional. The initial fibre length used by the model. If this value is omitted or set to ≤0 then it is calculated internally to minimise the stored strain energy. This value is produced when the model's state is saved to improve the restart accuracy but its effect is generally small since when not specified (or disabled by setting to -1) the model will calculate a sensible value.

[4] He, J., Levine, W. S. & Loeb, G. E. 1991 Feedback gains for correcting small perturbations to standing posture. IEEE Transactions on Automatic Control 36, 322-332.

ActivationRate="double"

Optional. If set then this value overrides the *ActivationKinetics* setting and uses a linear activation rate model. This can be more appropriate when simple drivers are used. The value is the change of activation that can be achieved in 1 second so a value of 100 would indicate that changing from zero to full activation would take 10 ms.

StartActivation="double"

Optional. If set then this value sets the start activation of the muscle. Otherwise the activation will ramp up from zero following the selected activation dynamics.

UmbergerGerritsenMartin

For the *UmbergerGerritsenMartin* models the following options are required:

PCA="double"
FibreLength="double"
TendonLength="double"
ForcePerUnitArea="double"
SerialStrainAtFmax="double"
ParallelStrainAtFmax="double"
SerialStrainModel="string"
ParallelStrainModel="string"

These values are specified exactly as *MinettiAlexanderComplete*.

Can be set to *"Linear"* or *"Square"* depending on whether the elastic elements should follow a linear or square curve for their length/tension relationship.

VMaxFactor="double"

The maximum contractile speed of fast twitch fibres in lengths per second (strain rate). This definition is different from that used by the *MinettiAlexander*, *MinettiAlexanderExtended, and MinettiAlexanderComplete* models.

MuscleDensity="double"

The density of muscle tissue.

FastTwitchProportion="double"

The proportion of the muscle that is fast twitch.

Width="double"

The maximum length range of force production relative to the fibre length. This definition is different from that used by the *MinettiAlexanderComplete* model: this value is effectively the half width with a typical value of 0.5.

Aerobic="boolean"

If *"true"* then the aerobic model is used as opposed to the anaerobic model.

AllowReverseWork="boolean"

The muscle model in the paper allows negative work in the muscle to generate metabolic energy. This may not be wanted and certainly it gets exploited by global optimisation routines that minimise metabolic energy cost to produce almost zero cost locomotion.

DampedSpring

For the *DampedSpring* models the following options are required:

UnloadedLength="double"

The unloaded length of the spring (m).

SpringConstant ="double"

The stress generated by unit strain (N m^{-2}).

Area="double"

The cross section area of the spring (m^2).

Damping="double"

The damping constant of the spring (N m^{-2}). *Damping* is modeled by:

dampingStress = -strainRate × dampingConstant.

Strap Types

Each muscle is associated with a *Strap* which specifies the path of the muscle.

> *Strap="string"*

Specifies the strap type to use. This is one of *"TwoPoint"*, *"ThreePoint"*, *"NPoint"*, *"CylinderWrap"*, and *"TwoCylinderWrap"*. All straps require the specification of *Origin* and *Insertion* and some strap types require intermediate points too. Multipoint straps always pass through the multiple points and apply forces to the attached bodies as if the strap was a rope under tension and the points represent frictionless pulleys. They can be used to model retinacula. The *CylinderWrap* strap allows the definition of an infinitely long cylinder that the strap will wrap around when appropriate but ignore when not needed. The wrapping direction is set by either using a positive or negative value for the radius and the wrap limit is internally set to 180° which should be suitable for modelling the path of muscles around bones. *TwoCylinderWrap* is similar but it allows the muscle path to wrap around two parallel cylinders which can be very useful for muscles that act around multiple joints.

> *OriginBodyID="string"*
> *InsertionBodyID="string"*

Required for all straps. The *ID*s of the bodies to which the origin and insertion attach. These bodies must be defined before the muscle.

> *Origin="string"*
> *Insertion="string"*

Required for all straps. Position strings defining the positions of the origin and insertion. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of attachment point (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of the attachment point using *BodyID* coordinate system, *BodyID* can be *World.*

For *ThreePoint* straps the following additional definitions are required:

> *MidpointBodyID="string"*
> *Midpoint="string"*

Body ID for midpoint and position string for midpoint. Follows same conventions as origin and insertion definitions.

For *NPoint* straps the following additional definitions are required:

> *ViaPoint0BodyID="string"*
> *ViaPoint0="string"*

Body ID for via points and position string for via points. Follows same conventions as origin and insertion definitions. As many viapoints as required can be defined in pairs (*ViaPointBody0*, *ViaPoint0*), (*ViaPointBody1*, *ViaPoint1*), (*ViaPointBody2*, *ViaPoint2*), (*ViaPointBody3*, *ViaPoint3*) etc. The order that the strap wraps to is *Origin*, *ViaPoint0*, *ViaPoint1*, etc, *Insertion*.

For *CylinderWrap* straps the following additional definitions are required:

> *CylinderBodyID="string"*

The IDs of the *BODY* to which the cylinder attaches. This *BODY* must be defined before the muscle.

> *CylinderPosition="string"*

Position string defining the positions of the centre of the cylinder. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of cylinder (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of cylinder using *BodyID* coordinate system, *BodyID* can be *World*.

> *CylinderRadius="double"*

Specifies the cylinder radius.

> *CylinderQuaternion="string"*

Specifies the orientation of the cylinder. This can also be achieved by specifying the *CylinderAxis* which is usually easier. *"string"* is an orientation string and can use the following conventions when *OldStyleInputs="false"*:

- *"W X Y Z"*: quaternion specifying the rotation of the cylinder in world coordinates (this is the only option if *OldStyleInputs="true"*)

- *"BodyID W X Y Z"*: quaternion specifying the rotation of the cylinder using *BodyID* coordinate system, *BodyID* can be *World*.
- Note. The quaternion can also be specified as an angle axis construction by putting a ″d″ or a ″r″ immediately after the W parameter with no intervening space. The W parameter then represents an angle in degrees (″d″) or radians (″r″) and the XYZ components are the axis of the rotation.

Cylinder orientation can also be specified by setting the cylinder axis:

CylinderAxis="string"

"string" is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World*.

Note: Cylinders always wrap one way only following the right hand rule. If the muscle is wrapping the wrong way around a cylinder then simply negate the quaternion by negating the X, Y , Z components or negate the radius. If a cylinder wrapped muscle cannot be seen, it is likely that either the origin or insertion is within the cylinder which is not permitted.

For *TwoCylinderWrap* straps the following additional definitions are required:

CylinderBodyID="string"

The IDs of the *BODY* to which the cylinder attaches. This *BODY* must be defined before the muscle.

Cylinder1Position="string"

Position string defining the positions of the centre of the cylinder 1. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of cylinder (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of cylinder using *BodyID* coordinate system, *BodyID* can be *World*.

Cylinder1Radius="double"

Specifies the cylinder 1 radius.

Cylinder2Position="string"

Position string defining the positions of the centre of the cylinder 2. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of cylinder (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of cylinder using *BodyID* coordinate system, *BodyID* can be *World.*

Cylinder2Radius="double"

Specifies the cylinder 2 radius.

CylinderQuaternion="string"

Specifies the orientation of both cylinders. This can also be achieved by specifying the *CylinderAxis* which is usually easier. *"string"* is an orientation string and can use the following conventions when *OldStyleInputs="false"*:

- *"W X Y Z"*: quaternion specifying the rotation of the cylinder in world coordinates (this is the only option if *OldStyleInputs="true"*)
- *"BodyID W X Y Z"*: quaternion specifying the rotation of the cylinder using *BodyID* coordinate system, *BodyID* can be *World.*
- Note. The quaternion can also be specified as an angle axis construction by putting a "d" or a "r" immediately after the W parameter with no intervening space. The W parameter then represents an angle in degrees ("d") or radians ("r") and the XYZ components are the axis of the rotation.

Cylinder orientations can also be specified by setting the cylinder axis:

CylinderAxis="string"

"string" is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)

- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World*.

<DATATARGET options />

These are target values used to calculate how closely the model is matching externally provided data. They can be used for getting the model to match motion capture data for example. The current implementation is primarily for global matching techniques since it only produces a single match value but it would not be difficult to implement local matching strategies. This can be done to an extent by starting off matching a short sequence and gradually increasing the length of the simulation. Actually this works quite well because it allows earlier activation levels to be tweaked to allow better overall matching.

Type="string"

The type of data target. Currently optional (defaults to *"Scalar"*) for backward compatibility but will become compulsory in future versions. Valid types are *"Scalar"*, *"Quaternion"*, *"Vector"*. *Scalar* types return the arithmetic difference between target and value, *Vector* types return the Euclidean distance, and *Quaternion* types return the angle in radians between the target and actual quaternions.

ID="string"

Required. Name of the element: must be unique

DurationValuePairs="double double ... "
TargetTimes="double double ... "
TargetValues="double double ... "

Required. List of numbers representing the duration and the value to be matched. *DurationValuePairs* is deprecated and only works for *Scalar* types. In this version numbers are interleaved so need to be in the order *"time0 value0 time1 value0"* etc. The preferred way of specifying this information is to use *TargetTimes* followed by a list of times *"time0 time1 time3"*, and *TargetValues* followed by a list of values *"value0 value1 value2"*. These work for all *DataTarget* types.

TargetID="string"

Usually required (except for *MechanicalEnergy* and *MetabolicEnergy* targets). Name of the element that the match is being made to. The element must already exist and can be a *BODY*, *JOINT*, *GEOM* or *REPORTER*.

Intercept="double"
Slope="double"

Optional. *Intercept* and *Slope* have default values of 1. The match score is calculated using the following formula:

Score = Intercept - Slope × PositiveFunction(Target - Value)

The *PositiveFunction* depends on the *MatchType* chosen. The *Score* value will equal the *Intercept* if the difference between *Target* and *Value* is zero. The higher the value for *Slope*, the more rapidly this value will fall as the difference between *Target* and *Value* increases.

MatchType="string"

Optional. Defaults to *"Linear"* but can also be set to *"Square"*. This controls the *PositiveFunction* used. With *"Linear"* it is the absolute value, with *"Square"* it is the square of the value. Using *"Square"* means the total error term will be the standard sum of squares error term.

DataType="string"

Required by *Scalar* type. This specifies the property that is being matched. Possible options are:

- *"XP", "YP", "ZP"*, the world positions of the centre of mass
- *"Q0", "Q1", "Q2", "Q3"*, the world values of the orientation quaternion in W,X,Y,Z order
- *"XV", "YV", "ZV"*, the world values of linear velocity of the centre of mass
- *"XRV", "YRV", "ZRV"*, the world values of angular velocity of the body
- *"Angle"*, the angle of a hinge joint
- *MechanicalEnergy, MetabolicEnergy,* the total energy values of the system.

AbortThreshold="double"

Optional. If present, the simulation will abort if the match score returned by the target is less than this value.

Visible="boolean"

Optional. Can be used to disable the visibility of markers in the GUI.

<REPORTER options />

Reporters are the preferred way of generating output from a config file. However they only work for the GUI version of the file so debug command line options are still required for the command line version. They can also be used as match points for data targets. A common use for this is to set motion capture marker positions as targets and use reporters attached to limbs to calculate kinematic matching scores. The following attributes are common to all reporters:

Type="string"

Required. The type of data target. Valid types are *"Position"*, *"Torque"*, *"SwingClearanceAbort"*.

ID="string"

Required. Name of the element: must be unique

Position Reporter

These are the most commonly used reporters. They can be attached to a body and have their own location and orientation relative to the body. They will then report the global vales of their position, orientation and velocities as the simulation progresses.

This position relative to a host body is specified using the following attributes:

BodyID="string"

Required. Set the body the reporter is attached to. These must already have been defined in the file.

Position="string"

Required. Specify the position of the reporter. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of the reporter (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of the reporter using *BodyID* coordinate system, *BodyID* can be *World*.

Quaternion="string"

Required. Specify the orientation of the reporter. *"string"* is an orientation string and can use the following conventions when *OldStyleInputs="false"*:

- *"W X Y Z"*: quaternion specifying the rotation of the reporter in world coordinates (this is the only option if *OldStyleInputs="true"*)
- *"BodyID W X Y Z"*: quaternion specifying the rotation of the reporter using *BodyID* coordinate system, *BodyID* can be *World.*
- Note. The quaternion can also be specified as an angle axis construction by putting a "d" or a "r" immediately after the W parameter with no intervening space. The W parameter then represents an angle in degrees ("d") or radians ("r") and the XYZ components are the axis of the rotation.

Torque Reporter

Torque reporters are used to report the torque and moment arm that a particular muscle exerts across a particular point on a body. This is most useful when the position corresponds to a joint. The torque and moment arm are calculated using the rotational forces that the muscle applied to the body so the muscle must attach to the body. This means that when looking for moment arms and toques acting across two joint muscles, these must be calculated with respect to either the distal body or the proximal body and not the intermediate body. So, if you want to know the torque and moment arms applied by *m. rectus femoris* across the hip and the knee, you need two reporters: one attached to the pelvis and the other attached to the tibia. A reporter attached to the femur will not give you a sensible result since the muscle does not attach to this bone. The moment arms are calculated geometrically so no movement or force is required for the moment arm calculation to be correct. The torques and moment arms can also be aligned to the joint axis so that they give the effective moment arm for a muscle that acts across a hinge joint.

The required attributes are as follows:

BodyID="string"

Required. Set the body the reporter is attached to. These must already have been defined in the file.

PivotPoint="string"

Required. Specify the pivot point of the reporter. This is the point in space that the torques and moment arms are calculated around. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of the reporter (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of the reporter using *BodyID* coordinate system, *BodyID* can be *World.*

Axis="string"

Sets the primary axis of the torque triad. This is useful if the torque is being measured around a hinge joint where only a single axis is useful. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World.*

SwingClearanceAbort Reporter

This is a specialised reporter that is able to abort the simulation if the clearance of the swing limb is below a certain threshold.

It has the following additional attributes:

BodyID="string"

Required. Set the body the reporter is attached to. These must already have been defined in the file.

Position="string"

Required. Specify the position of the reporter. *"string"* is a position string and can use the following conventions when *OldStyleInputs="false"*:

- *"X Y Z"*: world coordinate location of the reporter (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate location of the reporter using *BodyID* coordinate system, *BodyID* can be *World.*

HeightThreshold="string"

VelocityThreshold="string"

These values set the swing clearance. The rule is that when the reporter is moving forward faster than *VelocityThreshold* then it must be at least *HeightThreshold* above the ground.

UpAxis="string"

Optional. Defines the up direction. *"string"* can be *"X"*, *"Y"*, or *"Z"*. Defaults to *"Z"*.

DirectionAxis="string"

Optional. Sets the direction axis for the forward velocity test. If set, this value is normalised and the dot product of this axis and the reporter velocity used to calculate the forward velocity. *"string"* is a position string using the following conventions:

- *"X Y Z"*: world coordinate direction of the axis (this is the only option if *OldStyleInputs="true"*)
- *"BodyID X Y Z"*: coordinate direction of the axis using *BodyID* coordinate system, *BodyID* can be *World.*

<CONTROLLER options />

Controllers are objects that can sit between driver and muscles to provide additional control options. The driver is set with controller as a target, and the controller is set with the muscle as a target. This allows a great deal more flexibility in terms of control functions and feedback depending on what the controller is programmed to do. There is currently only a single controller defined and its specification should be considered experimental. However all controllers will define the following attributes:

Type="string"

Required. The type of the controller. Current valid types are *"PIDMuscleLength" and "PIDTargetMatch"*.

ID="string"

Required. Name of the element: must be unique

PIDMuscleLength Controller

This controller allows a driver to specify a target length for a muscle. The controller then uses a standard PID (proportional–integral–derivative) controller mechanism to try to maintain the muscle length at that level. The behaviour of the controller is specified by the following attributes:

MuscleID="string"

Required. Set the muscle the controller is attached to. This must already have been defined in the file.

NominalLength="double"

The target length of the muscle is set relative to a nominal length specified by this value. For a muscle it is usually set to the slack length.

Kp="double"
Ki="double"
Kd="double"

These are the values that control the gains of the various aspects of the controller. The output of the controller is calculated as:

$$output = (Kp \times error) + (Ki \times error\_integral) + (Kd \times error\_derivative)$$

This is a standard implementation of a PID controller as can be found in any standard control theory textbook.

PIDTargetMatch Controller

This controller allows a driver to a data target that is used to control a muscle activation. The controller then uses a standard PID (proportional–integral–derivative) controller mechanism to try to minimise the error of the data target. The behaviour of the controller is specified by the following attributes:

MuscleID="string"

Required. Set the muscle the controller is attached to. This must already have been defined in the file.

DataTargetID="string"

The data target the will provide the input to the PID controller. The error value is the error signal from the data target.

Kp="double"

Ki="double"

Kd="double"

These are the values that control the gains of the various aspects of the controller. The output of the controller is calculated as:

$$output = (Kp \times error) + (Ki \times error\_integral) + (Kd \times error\_derivative)$$

This is a standard implementation of a PID controller as can be found in any standard control theory textbook.

<DRIVER options />

Drivers are used to apply a value to drivable elements such as muscles or controllers. All the muscle and controller types can be activated but the effect varies. For *DampedSpring* it is simply a tension multiplier, for *PIDMuscleLength* controllers it specifies the target length, but the effect is more complex for the other drivable types. The effect of drivers is simply a time dependent value change. They are optional but without drivers the muscles and controllers simply act passively with an activation of zero. All drivers have the following options:

ID="string"

Required. Name of the element: must be unique.

Type="string"

Required. This specifies the type of driver. Current options are *"Cyclic"*, *"Step"*, *"BoxCar"*, and *"StackedBoxCar"*. *Cyclic* and *Step* drivers provide a specific activation level for a specific time. *Cyclic* drivers wrap round and re-use the activation levels whereas *Step* drivers do not. *Cyclic* drivers have an optional *Phase* option to allow a phase lag to be implemented. The two BoxCar driver variants use either single or multiple box car functions and are cyclic.

TargetID="string"

Required. Name of the drivable type controlled by the driver. This must already exist.

Step Driver

DurationValuePairs="double double ... "

Required. List of numbers representing the duration and the activation level. The numbers are interleaved so need to be in the order *"duration0 activation0 duration1 activation1"* etc. For the *Step* driver *duration* is the absolute time that an activation starts at. In *Step* drivers these duration values must be increasing.

DriverRange="double double"

Optional. If set this clips the output of the driver to the specified range. The first value is the minimum and the last value is the maximum allowable output.

LinearInterpolation="double double"

Optional. If set the output of the driver ramps linearly from one value to the next. Otherwise the values are stepped and remain constant for each duration.

Cyclic Driver

DurationValuePairs="double double ... "

Required. List of numbers representing the duration and the activation level. The numbers are interleaved so need to be in the order *"duration0 activation0 duration1 activation1"* etc. For the *Cyclic* driver the *duration* is the duration of the activation not the absolute time it occurs.

DriverRange="double double"

Optional. If set this clips the output of the driver to the specified range. The first value is the minimum and the last value is the maximum allowable output.

LinearInterpolation="double double"

Optional. If set the output of the driver ramps linearly from one value to the next. Otherwise the values are stepped and remain constant for each duration.

PhaseDelay="double"

Optional. Phase delay for the activation durations. This wraps around so the last activation levels will be used for the duration of this delay before the first activation levels are used in sequence as normal.

BoxCar Driver

CycleTime="double"

Sets the cycle time for the box car function.

Delay="double"

Sets the time offset (normalised from 0 to 1) of the change from zero to the high value.

Width="double"

Sets the duration (normalised from 0 to 1) of the high value of the box car.

Height="double"

Sets the height of the box car function.

DriverRange="double double"

Optional. If set this clips the output of the driver to the specified range. The first value is the minimum and the last value is the maximum allowable output.

StackedBoxCar Driver

StackSize="integer"

Sets the number of individual boxcar functions that will be stacked together to produce the ultimate activation waveform.

CycleTimes="double double ... "

Sets the list of cycle times for the individual box car functions.

Delays="double double ... "

Sets the list of time offsets (normalised from 0 to 1) of the change from zero to the high value for the individual box car functions.

Widths="double double ... "

Sets the list of durations (normalised from 0 to 1) of the high value of the box car for the individual box car functions.

Heights="double double ... "

Sets the list of heights of the individual box car functions.

DriverRange="double double"

Optional. If set this clips the output of the driver to the specified range. The first value is the minimum and the last value is the maximum allowable output.