Documents

OpenSim Developer's Workshop

August 27-29, 2008,

Stanford University

Website: SimTK.org/home/opensim

OpenSim Workshop Agenda (Day 1)

DAY 1 – Wednesday, August 27, 2008

8:00am – 8:45am	OpenSim 1.6 installation and setup – *You*
8:45am – 9:00am	Welcome and goals of workshop – *Scott Delp*
9:00am – 10:00am	Participant Information and Goals – *You*
10:00am – 10:10am	BREAK
10:10am – 10:30am	Preparing your data – *Sam Hamner*
10:30am – 10:40am	Inverse dynamics and static optimization analyses – *Jeff Reinbolt*
10:40am – 11:30am	Guided analyses and exploration on your own – *Jeff Reinbolt & You*
11:30am – 12:30pm	LUNCH
12:30pm – 12:40pm	Forward dynamics simulation – *Ajay Seth*
12:40pm – 1:00pm	Guided simulation and exploration on your own – *Ajay Seth & You*
1:00pm – 1:10pm	Computed muscle control – *Ajay Seth*
1:10pm – 1:30pm	Guided simulation and exploration on your own – *Ajay Seth & You*
1:30pm – 2:00pm	Guided model editing and exploration on your own – *Jeff Reinbolt & You*
2:00pm – 2:10pm	Simulation analysis – *Ajay Seth*
2:10pm – 2:40pm	Guided analyses and exploration on your own – *Ajay Seth & You*
2:40pm – 3:00pm	BREAK
3:00pm – 3:10pm	Analysis template – *Ajay Seth*
3:10pm – 4:30pm	Creating your own analysis – *Ajay Seth & You*
4:30pm – 5:00pm	Form groups and create project plans – *Scott Delp & You*

OpenSim Workshop Agenda (Days 2 & 3)

DAY 2 – Thursday, August 28, 2008

8:30am – 9:30am	Presentations of group project plans – *You*
9:30am – 11:45am	Work on projects – *You & OpenSim Team*
11:45am – 12:45pm	LUNCH
12:45pm – 1:00pm	Open discussion of common issues – *Scott Delp & You*
1:00pm – 4:45pm	Work on projects – *You & OpenSim Team*
4:45pm – 5:00pm	Open discussion of common issues – *Scott Delp & You*
6:00pm –	STANFORD FOOTBALL

DAY 3 – Friday, August 29, 2008

8:30am – 8:45am	Open discussion of tips and comments – *Scott Delp & You*
8:45am – 11:45am	Work on projects – *You & OpenSim Team*
11:45am – 12:45pm	LUNCH
12:45pm – 2:45pm	Presentations of progress, hurdles, feedback, and future plans – *You*
2:45pm – 3:00pm	Closing remarks – *Scott Delp*
3:00pm – 4:30pm	RECEPTION

Trademarks and Copyright and Permission Notice

Acknowledgments

OpenSim was developed as a part of SimTK and funded by the Simbios National Center for Biomedical Computing through the National Institutes of Health and the NIH Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers can be found at http://nihroadmap.nih.gov/bioinformatics.

Table of Contents

1 Introduction

1.1 What is OpenSim?

OpenSim is a freely available software package that enables you to build, exchange, and analyze computer models of the musculoskeletal system and dynamic simulations of movement. OpenSim version 1.0 was introduced at the American Society of Biomechanics Conference in 2007. Since then, many people have begun to use the software in a wide variety of applications, including biomechanics research, medical device design, orthopedics and rehabilitation science, neuroscience research, ergonomic analysis and design, sports science, computer animation, robotics research, and biology and engineering education.

The software provides a platform on which the biomechanics community can build a library of simulations that can be exchanged, tested, analyzed, and improved through multi-institutional collaboration. The underlying software is written in C++ and the graphical user interface (GUI) is written in Java. OpenSim plug-in technology will make it possible to develop customized controllers, analyses, contact models, and muscle models among other things. These plug-ins can be shared without the need to alter or compile source code. You can analyze existing models and simulations and develop new models and simulations and visualize them within the GUI.

OpenSim is built using SimTK, an open-source simulation toolkit developed to create mathematical models of biological dynamics. SimTK is being developed by Simbios, an NIH National Center for Biomedical Computation based at Stanford University. Open-source, third-party tools are used for some basic functionality, including the Xerces Parser from the Apache Foundation for reading and writing XML files (xml.apache.org/xerces-c) and the Visualization Toolkit (VTK) from Kitware for visualization (www.vtk.org). Use of plug-in technology will allow low-level computational components such as integrators and optimizers to be updated as appropriate without extensive restructuring.

1.2 Capabilities

OpenSim includes a wide variety of features. You can find out about them by completing the tutorials and browsing the user guide and this handout. Some of the most useful features include:

- Scaling a Model
- Performing Inverse Kinematics Analyses
- Performing Inverse Dynamics Analyses
- Performing Static Optimization Analyses
- Generating Forward Dynamics Simulations
- Analyzing Dynamic Simulations
- Plotting Results
- Creating Snapshots and Making Animations

1.3 Model and Simulation Repository

You can create your own models of musculoskeletal structures and dynamic simulations of movement in OpenSim, as well as take advantage of computer models and dynamic simulations that other users have developed and shared. For example, you can use existing computer models of the human lower limb, upper limb, cervical spine, and whole body which have already been developed and posted at https://simtk.org/home/nmblmodels. You can also use dynamic simulations of walking and other activities that have been developed, tested and posted on Simtk.org. We encourage you to share your models and simulations with the research community by setting up a project on SimTK.org.

1.4 Compatibility with SIMM

SIMM (Software for Interactive Musculoskeletal Modeling) from Motion Analysis Corp. is a widely used software application for biomechanical simulation, surgical planning, and ergonomic analysis. The joint (*.jnt) and muscle (*.msl) files used by SIMM to describe models of the musculoskeletal system can be converted into OpenSim models (*.osim) and brought into the OpenSim framework, thus allowing users of OpenSim to build on the wealth of models built and validated in SIMM. In this way, OpenSim complements SIMM by enabling forward dynamics simulations of models without third party software or the need to compile your own source code.

1.5 Additional Resources and Help

You can learn more at the OpenSim project site at http://simtk.org/home/opensim. The project site provides a forum for users to ask questions and share expertise. You can also get additional information in the following article: Delp, S.L., Anderson, F.C., Arnold, A. S., Loan, P., Habib, A., John, C., Guendelman, E.G., Thelen, D.G., OpenSim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 11, pp. 1940-1950, 2007.

2 Preparing Your Data

2.1 Overview

This chapter describes the formats for data files that can be imported into OpenSim. Generally, you must input the following types of data into OpenSim to generate simulations:

1. Marker trajectories
2. Ground reaction forces and moments and centers of pressure

You may also import joint angles to provide additional kinematic data. Marker trajectories must be specified in .trc files, and ground reaction and center of pressure data must be specified in *.sto* or *.mot* files. Joint angles must be specified in *.sto* or *.mot* files. The *.sto* file format, which is similar to the *.mot* file format, is described below. EMG data may also be imported using *.sto* or *.mot* files, for example, to compare experimental EMG data to muscle excitations obtained from a simulation.

2.2 Laboratory Coordinates

Every set of (*x*, *y*, *z*) coordinates obtained from a motion capture system is given relative to some coordinate system. Typically, this coordinate system is called the *laboratory coordinate system*. The laboratory coordinate system is generally an inertial frame fixed to ground. Before inputting any coordinates from motion capture into OpenSim, you must to ensure that all (*x*, *y*, *z*) coordinates have been transformed from the laboratory coordinate system to the model coordinate system used in OpenSim. Although you can define an arbitrary model coordinate system, the standard convention used in OpenSim is shown in Figure 2-1.

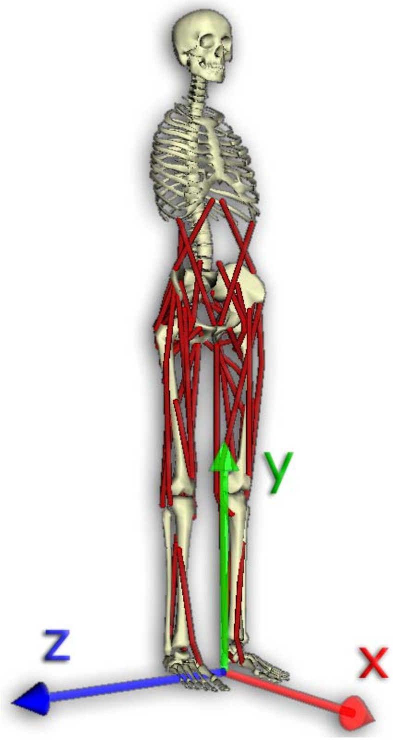


Figure 2-1. Model coordinate system.

OpenSim uses *meters* for all position and distance data. Once converted to the proper units, all (x, y, z) coordinates can be mapped from the laboratory coordinate system to the model coordinate system by a transformation. This transformation can be represented by a 3 × 3 rotation matrix which represents the orientation of the laboratory coordinate frame in the model coordinate frame. To transform the coordinates of a point ${}^{lab}\mathbf{P}$ = (x, y, z) given in the laboratory coordinate frame to its coordinates ${}^{model}\mathbf{P}$ = (x', y', z') in the model coordinate frame, you would employ the following transformation, where ${}^{model}\boldsymbol{R}^{lab}$ is the matrix whose columns are the vectors of the laboratory coordinate frame specified in the model coordinate frame:

$${}^{model}\mathbf{P} = {}^{model}\boldsymbol{R}^{lab} * {}^{lab}\mathbf{P}$$

External forces and moments are usually given in the coordinate system of a particular force sensor, such as a force plate, which may be different than the laboratory coordinate system. In this case, the force and moment data must be transformed from the appropriate force sensor's coordinate system to the model coordinate system.

2.3 File Formats

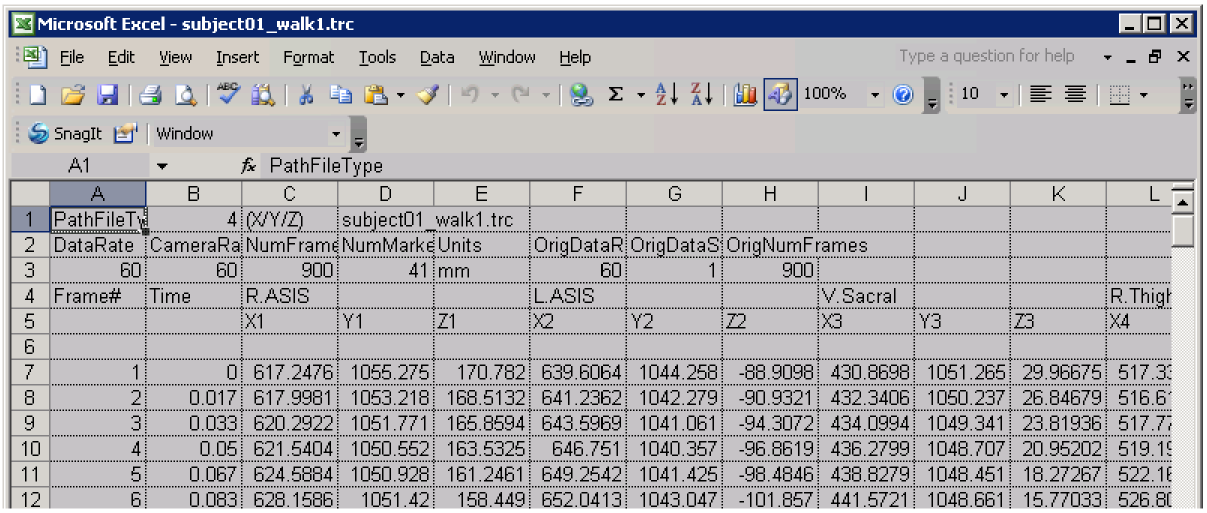
2.3.1 Marker (.trc) Files

The *.trc* (Track Row Column) file format was created by Motion Analysis Corporation to specify the positions of markers placed on a subject at different times during a motion capture trial. An example *.trc* file (subject01_walk1.trc) is provided in the *examples/Gait2354* directory, which is part of the OpenSim distribution. A fragment of this file is shown in Figure 2-2.

The first three lines of the *.trc* file is a header, followed by two rows of column labels, followed by a blank row, followed by the rows of data. Each row of data contains a frame number followed by a time value followed by the (x, y, z) coordinates of each marker. As a plain text file, a *.trc* file is commonly tab-delimited. So, for example, the fourth line in the *.trc* file in Figure 2-2 would look like this in plain text:

`Frame#`<tab>`Time`<tab>`R.ASIS`<tab><tab><tab>`L.ASIS`<tab><tab><tab>`V.Sacral`...

where <tab> indicates each tab character that would be present in the file.



Microsoft Excel - subject01_walk1.trc

A1 PathFileType

	A	B	C	D	E	F	G	H	I	J	K	L
1	PathFileTy	4	(X/Y/Z)	subject01_walk1.trc								
2	DataRate	CameraRa	NumFrame	NumMarke	Units	OrigDataR	OrigDataS	OrigNumFrames				
3	60	60	900	41	mm	60	1	900				
4	Frame#	Time	R.ASIS			L.ASIS			V.Sacral			R.Thig
5			X1	Y1	Z1	X2	Y2	Z2	X3	Y3	Z3	X4
6												
7	1	0	617.2476	1055.275	170.782	639.6064	1044.258	-88.9098	430.8698	1051.265	29.96675	517.3
8	2	0.017	617.9981	1053.218	168.5132	641.2362	1042.279	-90.9321	432.3406	1050.237	26.84679	516.6
9	3	0.033	620.2922	1051.771	165.8594	643.5969	1041.061	-94.3072	434.0994	1049.341	23.81936	517.7
10	4	0.05	621.5404	1050.552	163.5325	646.751	1040.357	-96.8619	436.2799	1048.707	20.95202	519.1
11	5	0.067	624.5884	1050.928	161.2461	649.2542	1041.425	-98.4846	438.8279	1048.451	18.27267	522.1
12	6	0.083	628.1586	1051.42	158.449	652.0413	1043.047	-101.857	441.5721	1048.661	15.77033	526.8

Figure 2-2: .trc File. The first few lines of the file are shown here.

DataRate indicates the sampling rate of the data in this *.trc* file in Hertz. **NumFrames**, indicates the number of frames (rows) of data in the whole *.trc* file. **OrigDataStartFrame**, indicates the frame number of the first frame (row) of data in the *.trc* file. Thus, the time *t* at which frame *f* was captured is *t* = (*f* – OrigDataStartFrame) / DataRate.

Note that the fourth row, which contains column labels, contains the name of each marker in the first column where the marker's coordinates appear. In the fifth row, the individual coordinates of each marker are labeled as **X1**, **Y1**, **Z1**, **X2**, **Y2**, **Z2**, etc. If this format for the fourth and fifth rows is not followed by a *.trc* file, OpenSim may fail to read the marker data correctly.

2.3.2 Motion (*.mot*) Files

The *.mot* (motion) file format was created by the developers of SIMM (Software for Interactive Musculoskeletal Modeling). The *.mot* file format is compatible with both SIMM and OpenSim. A *.mot* file consists of two parts: the motion header and the data. The motion header can come in two forms: (1) SIMM header only or (2) OpenSim and SIMM header.

(1) SIMM Header Example:

```
name subject01_walk1_grf.mot
datacolumns 19
datarows 9009
range 0.000000 15.013300
endheader
```

The first line must start with **name** followed by a space and the name of the *.mot* file. The next line should contain **datacolumns**, a space, and then the total number of columns of data in the *.mot* file (including the time column). The next line should contain **datarows**, a space, and then the total number of rows of data in the *.mot* file. The next line should contain **range**, a space, the first time value in the time column, a space, and then the last time value in the time column. Optionally, other comments could be included in subsequent lines. The final line **endheader** indicates the end of the header.

(2) OpenSim and SIMM Header Example:

```
Coordinates
nRows=500
nColumns=24

# SIMM Motion File Header:
name Coordinates
datacolumns 24
datarows 500
otherdata 1
range 0.750000 1.249000

Units are S.I. units (second, meters, Newtons, ...)
Angles are in degrees.

endheader
```

The first line is the name, which is **Coordinates** in this case, to be used to represent this *.mot* file when it is loaded into OpenSim. This does *not* have to be the name of the *.mot* file. The second line contains **nRows=** followed by the number of rows of data in the *.mot* file. The third line contains **nColumns=** followed by the number of columns of data (including the time column) in the *.mot* file. The fourth line is empty. The fifth line has a comment indicating that the SIMM motion file header is beginning, and then the following lines should have the same format as SIMM Header example.

Note that extra lines containing newline characters or comments can be included before the **endheader** line in the SIMM header section of both header types.

Immediately after the **endheader** line, the data section of the *.mot* file begins. The first line after the **endheader** line should contain tab-delimited labels for each column of (tab-delimited) data in the *.mot* file. The first column is assumed to be **time**, followed by values that vary with time such as generalized coordinates, marker coordinates, ground reaction forces and moments, centers of pressure, muscle activations, or muscle lengths. The names of these column labels should match the names used in the model with which the *.mot* file is intended to be used. The rows below this line of column labels must be the corresponding values of each of these quantities at the time represented by the first number in each row.

The time values in the time column of a *.mot* file must be uniformly spaced. An example *.mot* file (*subject01_walk1_grf.mot*) is provided in the *examples/Gait2354* directory, which is part of the OpenSim distribution.

2.3.3 Storage (*.sto*) Files

The *.sto* file format was created by the developers of OpenSim. It is very similar to the *.mot* file format, with two main differences:

- The time values in the time column of a *.sto* file do not have to be uniformly spaced
- The first column of a *.sto* file *must* contain time, whereas a *.mot* file can contain other quantities in the first column

There is only one format for the header of a *.sto* file and it is very simple, as shown below:

```
Coordinates
nRows=153
nColumns=24
endheader
```

The first line contains the name with which the *.sto* file will be referred to when it is loaded into OpenSim. The second line is **nRows=** followed by the number of rows of data in the *.sto* file. The third line is **nColumns=** followed by the number of columns of data in the *.sto* file (including the time column). The last line is **endheader**. Immediately following the **endheader** line is the data section of the *.sto* file, which is identical to the data section of a *.mot* file, except that the time column is allowed to have non-uniform spacing.

Example *.sto* files, such as *subject01_walk1_RRA_Actuation_force.sto*, are provided in the *examples/Gait2354/OutputReference/ResultsRRA* directory, which is part of the OpenSim distribution.

2.4 Representing Ground Reaction Data

You need to represent your ground reaction data in a *.mot* or *.sto* file for input into OpenSim. An example file (*subject01_walk1_grf.mot*) is given in the *examples/Gait2354* directory, which is part of the OpenSim distribution.

The first row below the header ***must*** contain the following column headings, in this order:

time	ground_**force**_**vx**	ground_**force**_**vy**	ground_**force**_**vz**	ground_**force**_**px**	ground_**force**_**py**	ground_**force**_**pz**	…
	.ground_**force**_**vx**	ground_**force**_**vy**	ground_**force**_**vz**	ground_**force**_**px**	ground_**force**_**py**	ground_**force**_**pz**	…
	ground_**torque**_**x**	ground_**torque**_**y**	ground_**torque**_**z**	ground_**torque**_**x**	ground_**torque**_**y**	ground_**torque**_**z**	

All rows below this line contain the corresponding data in each column. All data (except for the time column, column 1) must be specified in the model coordinate system. The labels **_vx**, **_vy**, and **_vz** correspond to the *x*, *y*, and *z* components of the ground reaction force vector in the model coordinate system. The labels **_px**, **_py**, and **_pz** correspond to the *x*, *y*, and *z* components of the center of pressure (COP) in the model coordinate system. The column headings correspond to**:**

Body1 Force	**Body2 Force**	**Body1 COP**	**Body2 COP**	**Body1 Torque**	**Body2 Torque**

Body1 and Body2 are specified in the *External Loads* settings of either the Inverse Dynamics or Computed Muscle Controls Tools. The examples presented in the *examples/Gait2354* directory specify Body1 as the right foot (*calcn_r*) and Body 2 as the left foot (*calcn_l*).

2.5 Marker Set

A marker set contains a list of the virtual markers that are placed on the body segments of a model. An example of a marker file is shown in Example 2-1 (next page), which can be found in the *examples/Gait2354* directory. Additionally, if a marker set is appended to a model file, it can be visualized with the model as shown in Figure 2-3.

A list of markers are enclosed inside the opening and closing tags <MarkerSet> and </MarkerSet>. Specifying a marker consists of specifying its <location>, as well as the <body> to which the marker is attached (i.e., which body its location is measured with respect to). The marker name is given by the **name** attribute of the <Marker> tag (e.g., **Sternum** for the first marker in Example 2-1).

The <fixed> property is used in the marker placement step and can be set to either **true** or **false**. If it is set to **false**, the marker will move during Scale if the *Adjust Model Markers* option is chosen to match the position of its corresponding experimental marker.

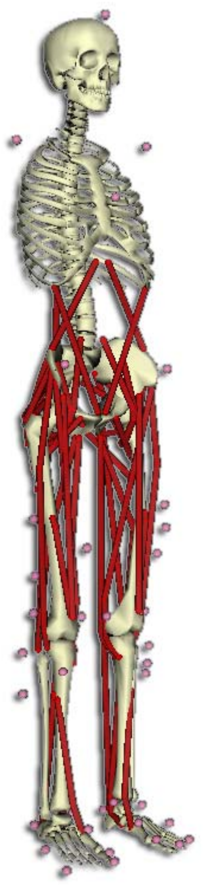


Figure 2-3. Model with Marker Set.

Example 2-1: XML file for a scale marker file
(e.g., gait2354_Scale_MarkerSet.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<MarkerSet name="gait2354_Scale">

  <objects>

    <Marker name="Sternum">
      <location> 0.07 0.3 0 </location>
      <body> torso </body>
      <fixed> false </fixed>
    </Marker>

    <Marker name="R.Acromium">
      <location> -0.03 0.44 0.15 </location>
      <body> torso </body>
      <fixed> false </fixed>
    </Marker>

    <Marker name="L.Acromium">
      <location> -0.03 0.44 -0.15 </location>
      <body> torso </body>
      <fixed> false </fixed>
    </Marker>

    <Marker name="Top.Head">
      <location>0.00084 0.657 0.0</location>
      <body> torso </body>
      <fixed> false </fixed>
    </Marker>

    <!-- . . additional <Marker> tags cut for brevity . . -->

  </objects>

</MarkerSet>
```

2.6 OpenSim Utilities

Example Matlab scripts for converting certain specific types of motion capture data into a form recognized by OpenSim are provided in the OpenSim Utilities project on Simtk.org (https://simtk.org/home/opensim-utils). We provide these utilities as examples that have been applied successfully to data sets from the Center for Gait and Motion Analysis at Gillette Children's Specialty Healthcare in St. Paul, MN, USA, and the Human Performance Laboratory at Stanford University in Stanford, CA, USA. However, it is difficult to anticipate lab-specific formats and conventions, so it is your responsibility to adapt these examples to the needs of your individual laboratories and motion capture systems.

3 Inverse Kinematics

3.1 How It Works

The Inverse Kinematics Tool goes through each time step (frame) of motion and computes generalized coordinate values which position the model in a pose that “best matches” experimental marker and coordinate values for that time step. Mathematically, the “best match” is expressed as a weighted least squares problem, whose solution aims to minimize both marker and coordinate errors.

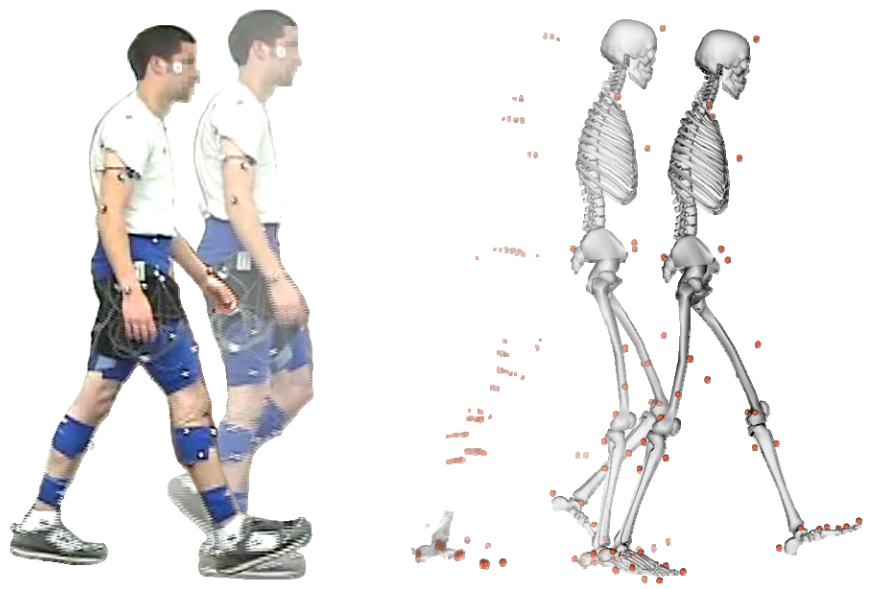


Figure 3-1: Inverse Kinematics Tool Overview. Experimental markers are matched by model markers throughout the motion by varying the generalized coordinates (e.g., joint angles) through time.

3.1.1 Marker Errors

A *marker error* is the distance between an experimental marker and the corresponding marker on the model (Figure 3-1) when the model is positioned using the generalized coordinates computed by the Inverse Kinematics solver. Each marker has a weight associated with it, specifying how strongly that marker's error term should be minimized.

3.1.2 Coordinate Errors

A *coordinate error* is the difference between an "experimental coordinate value" and the generalized coordinate value computed by the Inverse Kinematics Tool. Experimental coordinate values can be joint angles obtained directly from a motion capture system (i.e., built-in mocap inverse kinematics capabilities), or may be computed from experimental data by various specialized algorithms (e.g., defining anatomical coordinate frames and using them to specify joint frames that, in turn, describe joint angles) or by other measurement techniques that involve other measurement devices (e.g., a goniometer). A fixed desired value for a coordinate can also be a specified constant (e.g., if we know that a specific joint angle should stay at 0˚). The inclusion of experimental coordinate values is optional; the Inverse Kinematics Tool can solve for the generalized coordinates using marker matching alone.

3.1.3 Weighted Least Squares Equation

The weighted least squares problem solved by the Inverse Kinematics Tool is

$$\min_{\mathbf{q}} \left[\sum_{i \in \text{markers}} w_i \left\| \mathbf{x}_i^{\text{exp}} - \mathbf{x}_i(\mathbf{q}) \right\|^2 + \sum_{j \in \text{unprescribed coords}} \omega_j \left(q_j^{\text{exp}} - q_j \right)^2 \right]$$

$$q_j = q_j^{\text{exp}} \text{ for all prescribed coordinates } j$$

where **q** is the vector of generalized coordinates being solved for, $\mathbf{x}_\mathbf{i}^{\text{exp}}$ is the experimental position of marker *i*, $\mathbf{x}_\mathbf{i}(\mathbf{q})$ is the position of the corresponding marker on the model (which depends on the coordinate values), and q_j^{exp} is the experimental value for coordinate *j*.

3.2 Inputs

Three files are required as input by the Inverse Kinematics Tool:

> **arm26_elbow_flex.trc**: Experimental marker trajectories for a motion trial.
>
> **arm26_InverseKinematics_Tasks.xml**: Contains the inverse kinematics tasks (i.e., a specification of which virtual and experimental markers should be matched up during the inverse kinematics solution) and their relative weightings.
>
> **arm26.osim**: The current model loaded in OpenSim

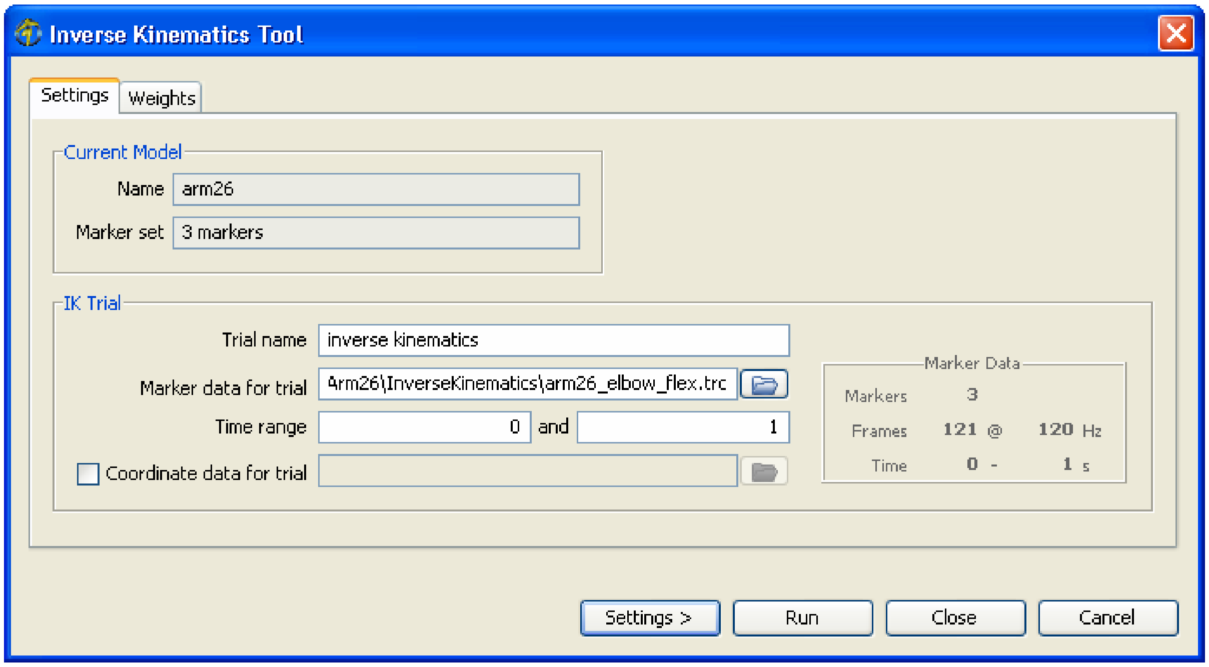
3.3 Outputs

The Inverse Kinematics Tool generates a single file:

> **arm26_InverseKinematics.mot**: Motion file containing the time histories of generalized coordinates that describe the movement of the model.

3.4 Inverse Kinematics Tool

To launch the Inverse Kinematics Tool, select **Inverse Kinematics…** from the **Tools** menu. The **Inverse Kinematics Tool** dialog (Figure 3-2) like all other OpenSim tools, operates on the *Current Model* open and selected in OpenSim (e.g., *arm26*). Inverse kinematics requires that a marker set is associated with the model and the number of markers is reported (e.g., *3 markers*). The **IK Trial** section specifies the experimental marker data that the Inverse Kinematics Tool will match with the current model. A *Trial name* can be associated with the trial to uniquely identify the resultant motion. The *Marker data for trial* field must contain the path to the marker data (in *.trc* format) and OpenSim will report the information it recognizes from the file such as the number of markers, the number of frames and sampling frequency as well as the start and end times of the data set in the **Marker Data** pane. Any subset of the time range can be specified for performing inverse kinematics in the *Time range* field, but by default the complete time range is specified. If the *Coordinate data for trial* flag is checked, then the Inverse Kinematics Tool will require coordinate values specified in a motion (.mot) file to be loaded.



Current Model

Name: arm26

Marker set: 3 markers

IK Trial

Trial name: inverse kinematics

Marker data for trial: 4rm26\InverseKinematics\arm26_elbow_flex.trc

Time range: 0 and 1

☐ Coordinate data for trial

Marker Data

Markers 3

Frames 121 @ 120 Hz

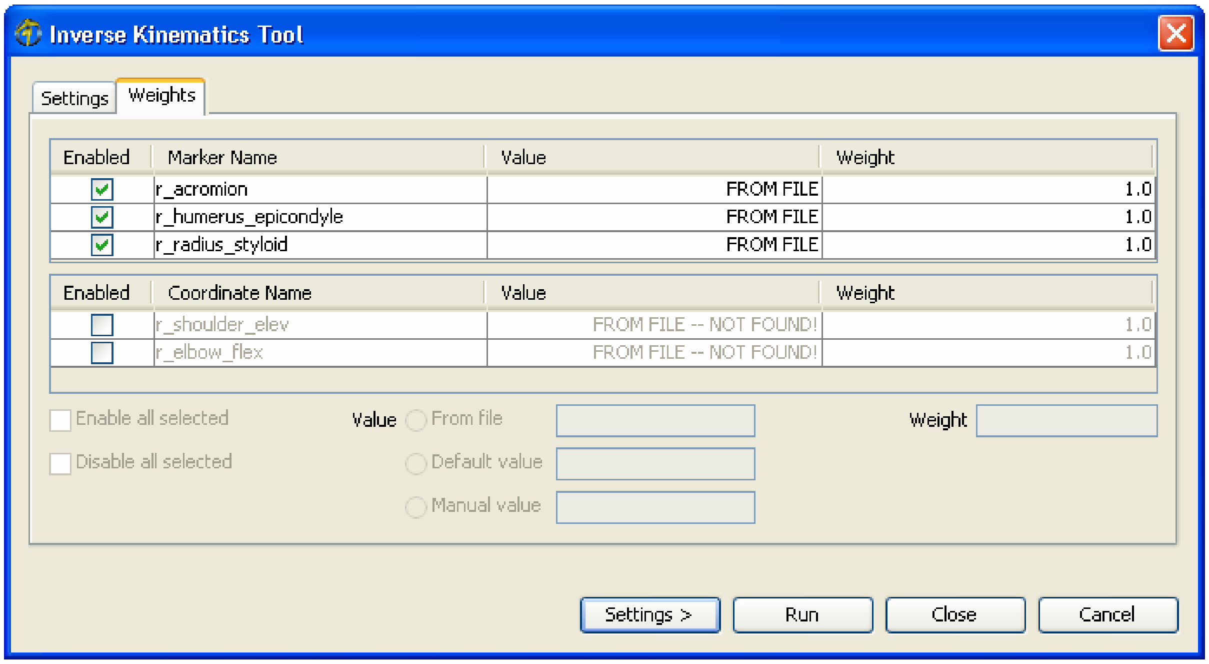
Time 0 - 1 s

Figure 3-2: Inverse Kinematics Tool Dialog

Once a marker file, and possibly a coordinate file, are specified, the behavior of the Inverse Kinematics Tool can be modified under the ***Weights*** tab (Figure 3-3). Each entry in the table represents a weight in the least-squares equation for either a marker (top table) or a coordinate (lower table). By selecting a row (or multiple rows), the entry fields below the panes become editable allowing the marker(s) or coordinate(s) to be enabled and allowing the user to specify a weight. The weight value will affect to what degree a match should be satisfied with larger weights penalizing errors for that marker or coordinate more heavily and thus attempting to match the experimental value more closely. For coordinates, the coordinate value to be matched can come from a specified motion file or set to its default or a user-specified (*manual*) constant value.

When running the Inverse Kinematics Tool from the GUI, the results from inverse kinematics are not automatically saved to file but are associated with the model under the

Motions category in the model Navigator. One can view multiple Inverse Kinematics results before saving to file. To save a motion, right click on the motion in the Navigator and select "Save as."



Enabled	Marker Name	Value	Weight
✓	r_acromion	FROM FILE	1.0
✓	r_humerus_epicondyle	FROM FILE	1.0
✓	r_radius_styloid	FROM FILE	1.0

Enabled	Coordinate Name	Value	Weight
☐	r_shoulder_elev	FROM FILE -- NOT FOUND!	1.0
☐	r_elbow_flex	FROM FILE -- NOT FOUND!	1.0

Figure 3-3: Specifying Inverse Kinematics Tool Weights

4 Inverse Dynamics

4.1 How it Works

The equations of motion for a multibody system may be written in the following form:

$$\underbrace{\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q})}_{\text{knowns}} = \underbrace{\boldsymbol{\tau}}_{\text{unknowns}}$$

where N is the number of degrees of freedom; $\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}} \in \boldsymbol{R}^N$ are the vectors of generalized positions, velocities, and accelerations, respectively; $\boldsymbol{M}(\boldsymbol{q}) \in \boldsymbol{R}^{N\times N}$ is the system mass matrix; $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) \in \boldsymbol{R}^N$ is the vector of Coriolis and centrifugal forces; $\boldsymbol{G}(\boldsymbol{q}) \in \boldsymbol{R}^N$ is the vector of gravitational forces; and $\boldsymbol{\tau} \in \boldsymbol{R}^N$ is the vector of generalized forces.

The motion of the model is completely defined by the generalized positions, velocities, and accelerations. Consequently, all of the terms on the left-hand side of the equations of motion are known. The remaining term on the right-hand side of the equations of motion is unknown. The Inverse Dynamics Tool uses the known motion of the model to solve the equations of motion for the unknown generalized forces (e.g., joint torques).

4.2 Inputs

Two files are required as input by the Inverse Dynamics Tool:

arm26_InverseKinematics.mot: Motion file containing the time histories of generalized coordinates that describe the movement of the model. This file may be generated by the Inverse Kinematics Tool.

arm26.osim: The current model loaded in OpenSim.

4.3 Outputs

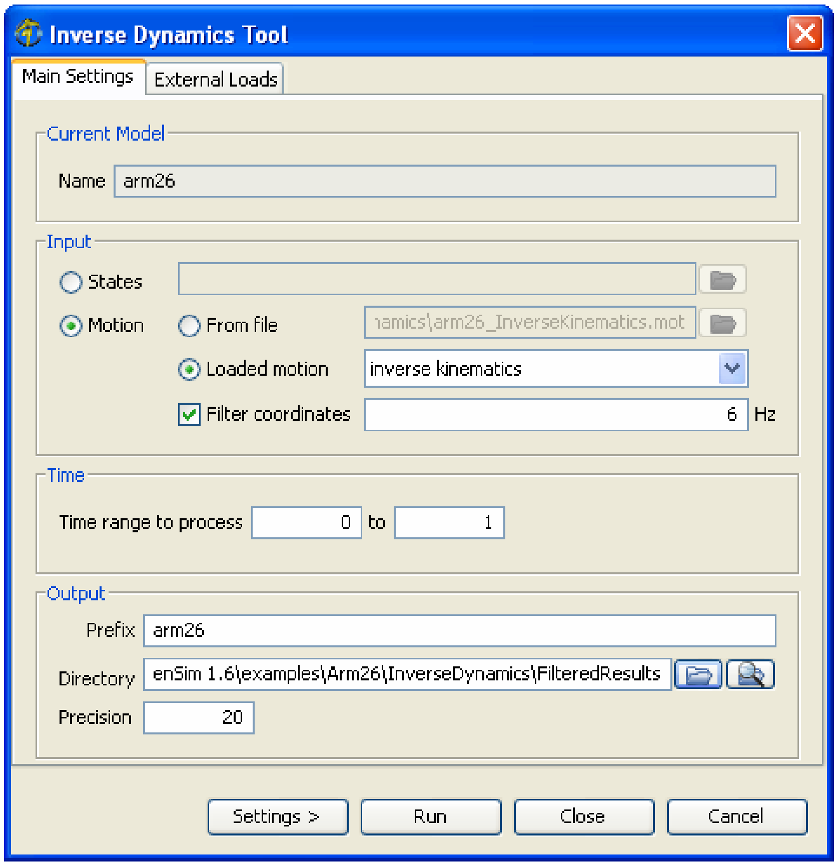
The Inverse Dynamics Tool generates a single file in a specified folder:

> **arm26_InverseDynamics_force.sto**: Storage file containing the time histories of the net forces and torques at each joint.

4.4 Inverse Dynamics Tool

To launch the Inverse Dynamics Tool select **Inverse Dynamics…** from the **Tools** menu. The **Inverse Dynamics Tool** dialog (Figure 4-1) like all other OpenSim tools operates on the *Current Model* open and selected in OpenSim (e.g., *arm26*). The Inverse Dynamics Tool is controlled by a dialog with two tabbed panes. The **Main Settings** pane specifies parameters relating to the input kinematics of the current model, the time range for the analysis, and the output of the results. The **External Loads** pane specifies parameters relating to the external loads applied to the model during the analysis.

The Main Settings pane (Figure 4-1) is organized into four main sections entitled **Current Model**, **Input**, **Time**, and **Output**. The **Current Model** section displays an uneditable name for the current model being used for the inverse dynamics analysis. The **Input** section displays editable information specifying the kinematics (e.g., states or motion) describing the movement of a model. The **Time** section displays editable information specifying the start and end time for the inverse dynamics analysis. The **Output** section displays editable information specifying the prefix appended to the resulting output file, the directory to which the file is saved, and the precision (number of decimal places) used when writing results. You may use the 📂 button to browse for a directory to save the output files, and the 🔍 button to open an explorer to the specified directory.



Current Model

Name: arm26

Input

States

[✓ Motion]

From file: ıamics\arm26_InverseKinematics.mot

[✓ Loaded motion]: inverse kinematics

[✓ Filter coordinates]: 6 Hz

Time

Time range to process: 0 to 1

Output

Prefix: arm26

Directory: enSim 1.6\examples\Arm26\InverseDynamics\FilteredResults

Precision: 20

Figure 4-1: Dialog for the Inverse Dynamics Tool. The main settings pane.

5 Static Optimization

5.1 How it Works

As described in Chapter 4 covering Inverse Dynamics, the motion of the model is completely defined by the generalized positions, velocities, and accelerations. The Static Optimization Tool uses the known motion of the model to solve the equations of motion for the unknown generalized forces (e.g., joint torques) subject to one of the following constraints:

$$\underbrace{\sum_{m=1}^{nm}\left(a_m F_m^0\right) r_{m,j} = \tau_j}_{\text{ideal force generators}} \quad \text{or} \quad \underbrace{\sum_{m=1}^{nm}\left[a_m f\left(F_m^0, l_m, v_m\right)\right] r_{m,j} = \tau_j}_{\text{constrained by force-length-velocity properties}}$$

while minimizing the objective function:

$$J = \sum_{m=1}^{nm}\left(a_m\right)^p$$

where *nm* is the number of muscles in the model; a_m is the activation level of muscle *m* at a discrete time step; F_m^0 is its maximum isometric force; l_m is its length; v_m is its shortening velocity; $f\left(F_m^0, l_m, v_m\right)$ is its force-length-velocity surface; $r_{m,j}$ is its moment arm about the j th joint axis; τ_j is the generalized force acting about the j th joint axis; and *p* is a user defined constant.

5.2 Inputs

Two files are required as input by the Static Optimization Tool:

arm26_InverseKinematics.mot: Motion file containing the time histories of generalized coordinates that describe the movement of the model. This file was generated by the Inverse Kinematics Tool.

arm26.osim: The current model loaded in OpenSim.

5.3 Outputs

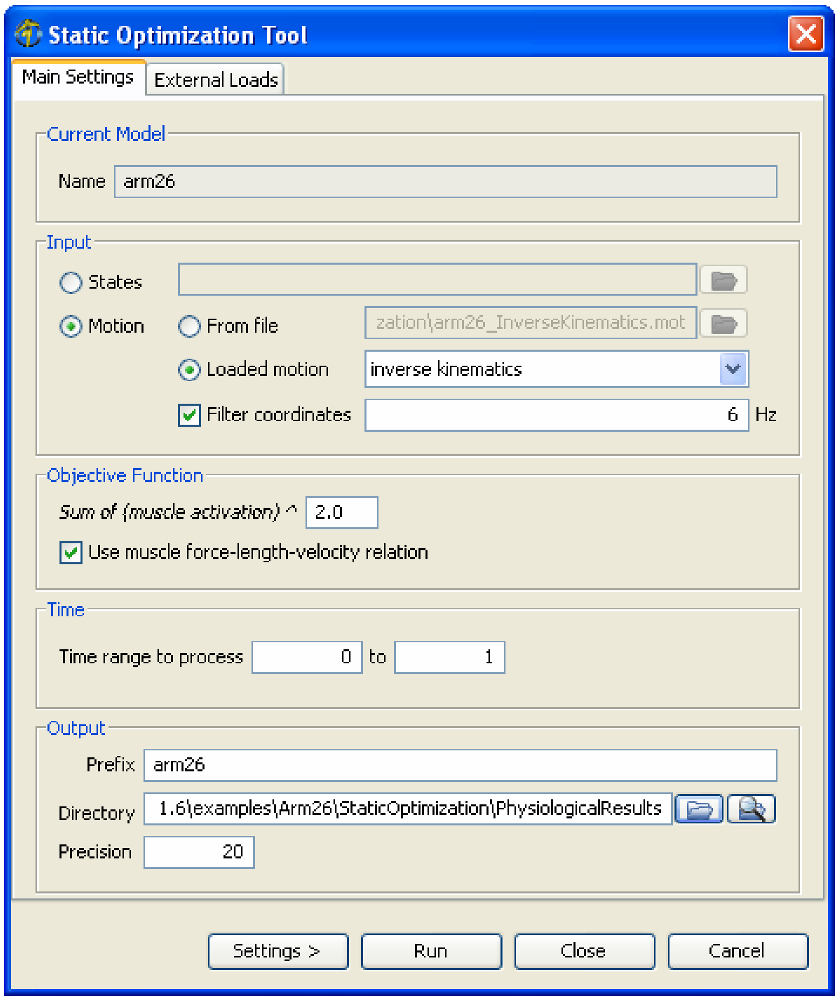
The Static Optimization Tool generates three files in a specified folder:

arm26_StaticOptimization_controls.xml: Contains the time histories of muscle activations. These controls were minimized by the Static Optimization Tool.
arm26_StaticOptimization_activation.sto: Storage file containing the time histories of muscle activations.
arm26_StaticOptimization_force.sto: Storage file containing the time histories of muscle forces.

5.4 Static Optimization Tool

To launch the Static Optimization Tool, select **Static Optimization…** from the **Tools** menu. The **Static Optimization Tool** dialog (Figure 5-1) like all other OpenSim tools operates on the *Current Model* open and selected in OpenSim (e.g. *arm26*). The Static Optimization Tool is controlled by a dialog with two tabbed panes. The **Main Settings** pane specifies parameters relating to the input kinematics of the current model, the time range for the analysis, and the output of the results. The **External Loads** pane specifies parameters relating to the external loads applied to the model during the analysis.

The Main Settings pane (Figure 5-1) is organized into five main sections entitled **Current Model**, **Input**, **Objective Function**, **Time**, and **Output**. The **Current Model** section displays an uneditable name for the current model being used for the static optimization analysis. The **Input** section displays editable information specifying the kinematics (e.g., states or motion) describing the movement of a model. The **Objective Function** section displays editable information specifying the power to which the muscle activations should be raised and whether or not to use the muscle force-length-velocity properties. The **Time** section displays editable information specifying the start and end time for the Static Optimization analysis. The **Output** section displays editable information specifying the prefix appended to the resulting output file, the directory to which the file is saved, and the precision (number of decimal places) used when writing results. You may use the 📂 button to browse for a directory to save the output files, and the 🔍 button to open an explorer to the specified directory.



Current Model
Name: arm26

Input
States
[✓ Motion] From file: zation\arm26_InverseKinematics.mot
[✓ Loaded motion]: inverse kinematics
[✓ Filter coordinates]: 6 Hz

Objective Function
Sum of (muscle activation) ^ 2.0
[✓ Use muscle force-length-velocity relation]

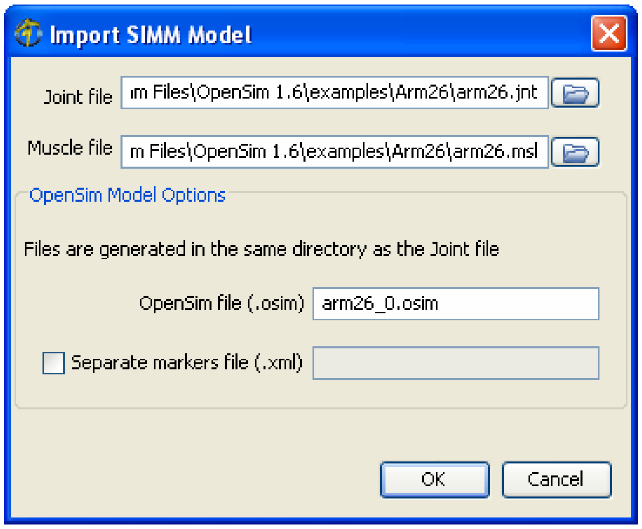
Time
Time range to process 0 to 1

Output
Prefix: arm26
Directory: 1.6\examples\Arm26\StaticOptimization\PhysiologicalResults
Precision: 20

Figure 5-1: Dialog for the Static Optimization Tool. The main settings pane.

6 Example: Inverse Kinematics, Inverse Dynamics, and Static Optimization

6.1 Importing a SIMM Model of the Upper Extremity



Joint file: ım Files\OpenSim 1.6\examples\Arm26\arm26.jnt

Muscle file: m Files\OpenSim 1.6\examples\Arm26\arm26.msl

Files are generated in the same directory as the Joint file

OpenSim file (.osim): arm26_0.osim

☐ Separate markers file (.xml)

Figure 6-1: Dialog for Importing a SIMM Model

1. **Open Import SIMM Model Dialog**. To import a SIMM model into OpenSim, select **Import SIMM Model...** from the **File** menu.
2. **Specify Joint File**. Browse to a the Arm26 example directory (e.g., ..*OpenSim 1.6\examples\Arm26*) and select the joint file (e.g., *arm26.jnt*).
3. **Specify Muscle File**. Continue by selecting a muscle file (e.g., *arm26.msl*).
4. **Specify OpenSim File**. Enter a name (e.g., *arm26_0.osim*) for the OpenSim model to be generated.

5. **Specify Dynamics Engine**. Select **Simbody engine** (for dynamics) from the drop down list.
6. **Close Import SIMM Model Dialog**. Click the **OK** button.

6.2 Performing Inverse Kinematics

1. **Open Inverse Kinematics Tool**. To open the tool (Figure 3-2), select **Inverse Kinematics...** from the **Tools** menu.
2. **Specify Trial Name**. Begin in the **Settings** pane and enter a name (e.g., *inverse kinematics*) for the trial.
3. **Specify Marker Data for Trial**. Browse to the inverse kinematics directory (e.g., *..\Arm26\InverseKinematics*) and select the marker file (e.g., *arm26_elbow_flex.trc*).
4. **Specify Time Range**. Enter a range from 0 to 1 seconds corresponding to the interval in the marker file.
5. **Confirm Weights**. Move to the **Weights** pane and confirm that all marker weights are 1.
6. **Save Settings**. Use the **Settings >** button to save your settings to a setup file (e.g., *arm26_Setup_InverseKinematics.xml*).
7. **Run and Close Inverse Kinematics Tool**. You will see the model begin to move as the optimization flexes the elbow to best match marker trajectories. When the analysis has completed by reaching the end of the specified time range, the **Motions** branch under the model in the **Navigator** will be populated by the inverse kinematics motion.

6.3 Viewing Inverse Kinematics Results

1. **View Motion**. Use the motion viewer controls to play back the inverse kinematics motion.
2. **Save Results from Inverse Kinematics**. Right click the new motion in under the model in the **Navigator** and select **Save As** to save the file (e.g., *arm26_InverseKinematics.mot*).

3. **Plot Joint Angles from Inverse Kinematics**. From the resulting motion (e.g., *inverse kinematics*), plot *r_shoulder_elev* and *r_elbow_flex* versus *time*.

6.4 Performing Inverse Dynamics

1. **Open Inverse Dynamics Tool**. To open the tool (Figure 4-1), select **Inverse Dynamics…** from the **Tools** menu.
2. **Specify Unfiltered Input Motion**. Use the radio buttons to select the **Loaded motion** (e.g., *inverse kinematics*) and uncheck the **Filter coordinates** option.
3. **Specify Time Range**. Enter a range from 0 to 1 seconds corresponding to the interval in the motion.
4. **Specify Output Directory**. Set the output **Directory** (e.g., *..\InverseDynamics\UnfilteredResults*), so that you are able to compare the results of inverse dynamics analyses using unfiltered and filtered input motions.
5. **Save Settings**. Use the **Settings >** button to save your settings to a setup file (e.g., *arm26_Setup_InverseDynamics.xml*).
6. **Run Inverse Dynamics Tool**. You will see the model begin to move as the analysis flexes the elbow while computing joint torques. When the analysis has completed by reaching the end of the specified time range, the specified output directory will be populated by a storage file (e.g., *arm26_InverseDynamics_force.sto*).
7. **Specify New Filtered Input Motion**. Check the **Filter coordinates** option and enter a cutoff frequency of 6 Hz.
8. **Specify New Output Directory**. Rename the output **Directory** (e.g., *..\InverseDynamics\FilteredResults*).
9. **Run and Close Inverse Dynamics Tool**.

6.5 Comparing Inverse Dynamics Results

1. **Plot Noisy Joint Torques from Inverse Dynamics**. From the resulting file (e.g., *..\UnfilteredResults\arm26_InverseDynamics_force.sto*), plot *r_shoulder_elev* and *r_elbow_flex* versus *time*. Leave the **Plotter** dialog open to compare subsequent joint torques.

2. **Plot Smooth Joint Torques from Inverse Dynamics**. From the resulting file (e.g., *..\FilteredResults\arm26_InverseDynamics_force.sto*), plot *r_shoulder_elev* and *r_elbow_flex* versus *time*. Compare with noisy joint torques.

6.6 Performing Static Optimization

1. **Open Static Optimization Tool**. To open the tool (Figure 5-1), select **Static Optimization…** from the **Tools** menu.
2. **Specify Filtered Input Motion**. Use the radio buttons to select the **Loaded motion** (e.g., *inverse kinematics*), check the **Filter coordinates** option, and enter a cutoff frequency of 6 Hz.
3. **Specify Nonphysiological Objective Function**. Raise ***Sum of (muscle activation)*** to a power of 2.0 and uncheck the **Use muscle force-length-velocity relation** option.
4. **Specify Time Range**. Enter a range from 0 to 1 seconds corresponding to the interval in the motion.
5. **Specify Output Directory**. Set the output **Directory** (e.g., *..\StaticOptimization\NonphysiologicalResults*), so that you are able to compare the results of static optimization analyses using nonphysiological and physiological objective functions.
6. **Save Settings**. Use the **Settings >** button to save your settings to a setup file (e.g., *arm26_Setup_StaticOptimization.xml*).
7. **Run Static Optimization Tool**. You will see the model begin to move as the analysis flexes the elbow while computing muscle activations. When the analysis has completed by reaching the end of the specified time range, the specified output directory will be populated by a controls file (e.g., *arm26_StaticOptimization_controls.xml*) and two storage files (e.g., *arm26_StaticOptimization_activation.sto* and *arm26_StaticOptimization_force.sto*).
8. **Specify New Physiological Objective Function**. Check the **Use muscle force-length-velocity relation** option.
9. **Specify New Output Directory**. Rename the output **Directory** (e.g., *..\StaticOptimization\PhysiologicalResults*).

10. **Run and Close Static Optimization Tool**.

6.7 Comparing Static Optimization Results

1. **Plot Nonphysiological Muscle Activations from Static Optimization**. From the resulting file (e.g., ..\*NonphysiologicalResults\arm26_StaticOptimization_activation.sto*), plot *BIClong* versus *time*. Leave the **Plotter** dialog open to compare subsequent activations.
2. **Plot Physiological Muscle Activations from Static Optimization**. From the resulting file (e.g., ..\PhysiologicalResults\arm26_StaticOptimization_activation.sto), plot *BIClong* versus *time*. Compare with nonphysiological muscle activation.

7 Forward Dynamics

7.1 How it Works

The Forward Dynamics Tool uses the model together with the initial states and controls to run a muscle-driven forward dynamics simulation. A forward dynamics simulation is the solution to the differential equations that define a dynamical system (model). For example, the dynamics of musculoskeletal models typically take the form of :

$$\ddot{q} = \left[\mathbf{M}(q)\right]^{-1}\left\{\boldsymbol{\tau}(a,l,\dot{l}) - \mathbf{C}(q,\dot{q}) + \mathbf{G}(q)\right\} \qquad \text{Multibody dynamics}$$

$$\dot{l} = \Lambda(a,l,q) \qquad \text{Muscle contraction dynamics}$$

$$\dot{a} = \mathrm{A}(a,x) \qquad \text{Muscle activation dynamics}$$

describing the acceleration, $\ddot{q}$, of the bones as rigid bodies due to muscle loads, $\boldsymbol{\tau}$, centrifugal, Coriolis, **C**, and gravity forces, **G**, as a function of joint angles, *q*, and their velocities, $\dot{q}$. The muscle loads are a function of muscle activations, *a*, and muscle fiber lengths and velocities, *l* and $\dot{l}$. In turn, fiber velocity and muscle activation rates are governed by muscle contraction and activation dynamics that are dependent on the current muscle and model kinematic state as well as the input excitations, *x*, generally termed the model's controls.

A 5th order Runge-Kutta-Feldberg integrator is used to solve (numerically integrate) the dynamical equations for the trajectories of the states for the musculoskeletal model over a definite interval in time. The Forward Dynamics Tool is an open-loop system that applies actuator controls with no feedback or correction mechanism to ensure that states follow a desired trajectory.

7.2 Inputs

Three files are required as input by the Forward Dynamics Tool:

arm26_StaticOptimization_controls.xml: Contains the time histories of muscle activations computed by the static optimization tool.

arm26_InitialStates.sto: Contains the model initial states including joint angles, joint velocities, muscle activations, and muscle fiber lengths. These states are used by the forward dynamics tool to set the initial states of the model for forward integration.

arm26.osim: The current model loaded in OpenSim.

7.3 Outputs

The **Forward Dynamics Tool** generates the complete state trajectory over the interval of integration and writes them to a file in a specified directory. Any analyses associated with the model to compute derived quantities (like joint powers or induced accelerations) will also generate files in the output directory.

At a minimum Forward Dynamics Tool generates three files in a specified folder:

arm26_states.sto: Contains the time histories of all the states of the model including joint angles (coordinates) and velocities and muscle states such as activation and fiber length.

arm26_states_degrees.mot: Is a SIMM compatible motion file that has joint states in degrees.

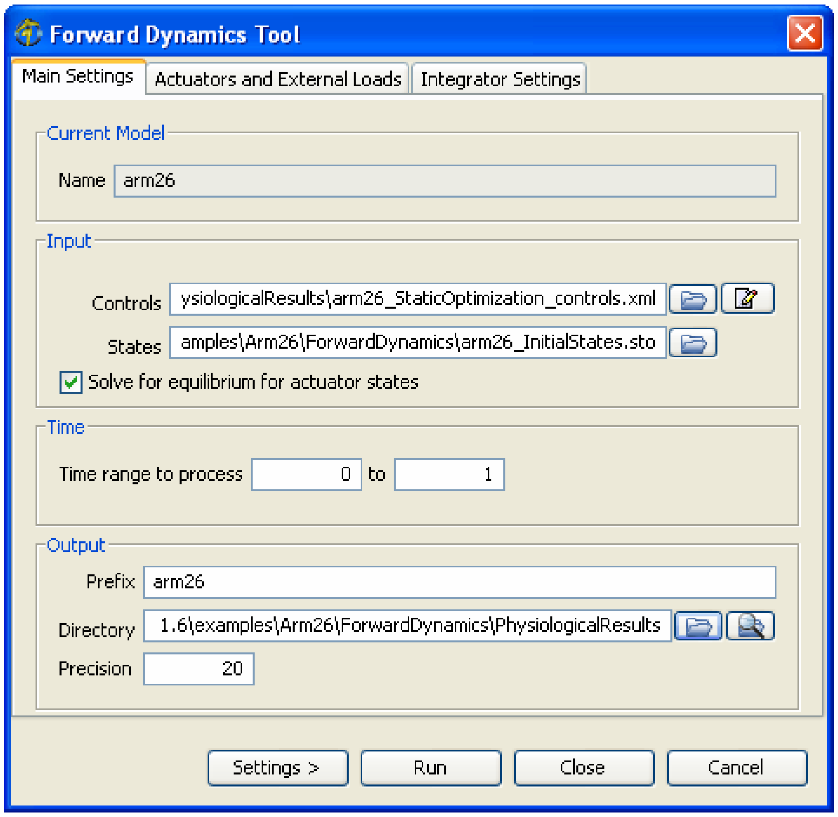
arm26_controls.sto: The control values used by the model at each integration step.

Additional files relating to Kinematics (joints trajectories), BodyKinematics (center-of mass trajectories), and Actuator (forces, powers) analyses are automatically associated with a forward simulation.

7.4 Forward Dynamics Tool

To launch the Forward Dynamics Tool, select **Forward Dynamics…** from the **Tools** menu. The Forward Dynamics Tool is controlled by a dialog with three tabbed panes (Fig. 6-1). The **Main Settings** pane specifies parameters relating to the controls and states that will be input into the model, the time range for the simulation, and the output of the results. The **Actuators and External Loads** pane specifies the actuator set and the external loads applied to the model during the simulation. The **Integrator Settings** pane specifies integrator step sizes and tolerances used to solve the simulation. The controls and states (if available) from *Static Optimization* or *Computed Muscle Control* can be used directly as input to the Forward Dynamics Tool.

The Main Settings pane (Fig. 6-1) is organized into four main sections entitled **Current Model**, **Input**, **Time**, and **Output**. The **Current Model** section displays uneditable information about the current model being used for analysis by the Forward Dynamics Tool. The **Input** section displays editable information specifying the controls and initial states to be used to run the forward simulation. You may use the button to browse for the controls and initial states files. The **Time** section displays editable information specifying the start and end time for the forward simulation. The **Output** section displays editable information specifying the prefix appended to all of the resulting output files, the directory to which the files are saved, and the precision (number of decimal places) used when writing results. You may use the button to browse for a directory to save the output files, and the button to open an explorer to the specified directory.



Current Model

Name: arm26

Input

Controls: ysiologicalResults\arm26_StaticOptimization_controls.xml

States: amples\Arm26\ForwardDynamics\arm26_InitialStates.sto

✓ Solve for equilibrium for actuator states

Time

Time range to process: 0 to 1

Output

Prefix: arm26

Directory: 1.6\examples\Arm26\ForwardDynamics\PhysiologicalResults

Precision: 20

Figure 7-1: Dialog for the Forward Dynamics Tool. The main settings pane.

8 Example: Forward Dynamics

8.1 Generating a Forward Dynamics Simulation of the Upper Extremity

1. **Open Forward Dynamics Tool**. To open the tool (Figure 7-1), select **Forward Dynamics…** from the **Tools** menu.
2. **Specify Nonphysiological Controls**. In the Forward Dynamics Tool dialog, select the **Controls** file (e.g., *arm26_StaticOptimization_controls.xml*) from the static optimization results that do not use the muscle force-length-velocity relation. These results are located in your specified output directory (e.g., *..\StaticOptimization\NonphysiologicalResults*).
3. **Specify Initial States**. An initial **States** file is provided in the Forward Dynamics directory (*..\ForwardDynamics\arm26_InitialStates.sto*) and the Forward Dynamics Tool should be set to use this file.
4. **Solve for Equilibrium**. This option makes sure that the initial actuator states (muscle activation, fiber length) are in static equilibrium, that is the rate of contraction is zero. This is a useful option for setting initial states when one does not have reliable estimates and the model is starting from rest. It utilizes the input state to determine the position of the model and initial activation of the muscles from which the initial muscle fiber lengths are computed.
5. **Specify Time Range**. The **time range** for the forward simulation is specified and these should be set from 0 to 1 sec to correspond to the interval upon which the controls from static optimization were computed.
6. **Specify Output Directory**. Set the output **Directory** (e.g., *..\ForwardDynamics\NonphysiologicalResults*), so that you are able to compare the results of a forward simulation using nonphysiological and physiological controls earlier by static optimization.
7. **Specify Model's Actuators**. Set the actuator settings to **Append** (rather than replace).
8. **Save Settings**. Use the **Settings >** button to save your settings to a setup file (e.g., *arm26_Setup_ForwardDynamics.xml*).

9. **Run Forward Dynamics Tool.** You will see the model begin to move as muscles contract and accelerate the model. When the simulation has completed by reaching the end of the specified time range, the specified output directory will be populated by states files (e.g., *arm26_states_degrees.mot*) and the corresponding motion file (e.g., *arm26_states*) will be associated with the model in the GUI.

8.2 Analyzing your Forward Dynamics Simulation in the GUI

1. **View Motion**. Use the motion viewer controls to play back the forward dynamics motion.
2. **Plot Nonphysiological Joint Angles from Forward Dynamics**. Browse to the output directory specified above (e.g., *..\ForwardDynamics\NonphysiologicalResults\arm26_states_degrees.mot*). Plot *r_shoulder_elev* and *r_elbow_flex* versus *time*. Leave **Plotter** dialog open to compare subsequent joint angles.
3. **Plot Joint Angles from Inverse Kinematics**. Use the motion loaded in the model (e.g., *inverse kinematics*) or browse to the inverse kinematics directory (e.g., *..\InverseKinematics\arm26_InverseKinematics.mot*). Plot *r_shoulder_elev* and *r_elbow_flex* versus *time*. How do they compare? Leave **Plotter** dialog open to compare subsequent joint angles.

8.3 A Forward Dynamics Simulation with Different Controls

1. **Specify New Physiological Controls**. In the Forward Dynamics Tool dialog, select the **Controls** file (e.g., *arm26_StaticOptimization_controls.xml*) from the static optimization results that use the muscle force-length-velocity relation. These results are located in your specified output directory (e.g., *..\StaticOptimization\PhysiologicalResults*).
2. **Specify New Output Directory**. Rename the output **Directory** (e.g., *..\ForwardDynamics\PhysiologicalResults*).
3. **Run Forward Dynamics Tool**.

8.4 Comparing Forward Dynamics Results

1. **View New Motion**. Use the motion viewer controls to playback the forward dynamics motion.
2. **Plot Physiological Joint Angles from Forward Dynamics**. Browse to the output directory specified above (e.g., *..\ForwardDynamics\PhysiologicalResults\arm26_states_degrees.mot*). Plot *r_shoulder_elev* and *r_elbow_flex* versus *time*. Leave **Plotter** dialog open to compare subsequent joint angles.

   How do the joint angles from the physiological controls compare to the previous nonphysiological motion? Why do the controls estimated from static optimization not sufficiently flex the elbow and extend the shoulder too much? To explore this question, use plot muscle states (activation and fiber lengths) and compare forces between the forward simulation and the static optimization activations (controls) and forces.

8.5 Modifying the Muscle Controls

1. **Open Excitation Editor**. To open the editor, select **Excitation…** from the **Edit** menu.
2. **Load Controls**. In the Excitation Editor dialog, load the controls file (e.g., *arm26_StaticOptimization_controls.xml*) from the static optimization results that use the muscle force-length-velocity relation. These results are located in your specified output directory (e.g., *..\StaticOptimization\PhysiologicalResults*). Select muscle excitations to edit (e.g., *BIClong*).
3. **Modify Controls**. The controls appear as individual graphs with moveable control nodes, which enable you to reshape the controls as desired. To select an individual control node, hold the **Ctrl** key and click the control node of interest. To select multiple control nodes, hold the **Ctrl** key and drag a box (*from top left to bottom right*) around control nodes of interest. Use the left mouse button to drag selected control node(s) to new location. Increase the excitation to Biceps Long (*BIClong*) by 25% or so.

4. **Save Modified Controls**. Use the **Save As** button to save the modified controls to a new file (e.g., *..\ForwardDynamics\arm26_Modified_controls.xml*).
5. **Specify New Modified Controls**. In the Forward Dynamics Tool dialog, select the **Controls** file (e.g., *..\ForwardDynamics\arm26_Modified_controls.xml*) saved above.
6. **Specify New Output Directory**. Rename the output **Directory** (e.g., *..\ForwardDynamics\ModifiedPhysiologicalResults*).
7. **Run and Close Forward Dynamics Tool**.
8. **View New Motion**. Use the motion viewer controls to play back the forward dynamics motion. Was performance improved?

8.6 Experiment On Your Own

1. **Experiment with Different Controls**. Repeat section 8.5 with other muscle controls.
2. **View Results Simultaneously**. Multiple models can be open at once to visualize simulation results simultaneously. Open another model (e.g., *arm26.osim*) by selecting **Open model...** from the **File** menu. The new model will appear offset from the original model. You can associate previously computed states (e.g., *arm26_states_degrees.mot*) to this model by selecting **Load motion...** from the **File** menu.
3. **Make a Movie**. Use the camera tool to take snapshots or use the movie-camera to generate animations. The camera dolly allows the view point of the movie-camera to change during when the animation is being captured, by interpolating between user defined views.

9 Computed Muscle Control

9.1 Why is Computed Muscle Control Necessary

Muscle controls computed from static optimization (Chapter 5) applied to a musculoskeletal model are likely to fail to reproduce the observed motion (the inputs to inverse dynamics and static optimization) when applied in a forward dynamics simulation. There are three principle causes for this discrepancy: 1) forward and inverse musculoskeletal models do not share identical dynamics, 2) experimental noise and sampling results in dynamically inconsistent kinematics and 3) musculoskeletal models are nonlinear dynamical systems and inherently chaotic. Cause 3) is often overlooked but it is important to realize that even if identical models where used in an inverse and then forward analysis with noiseless and error-free kinematics (i.e. synthetic data) a forward simulation will fail to reproduce the initial performance if the initial states of the simulation are not identical, since even the smallest of differences (too machine precision) can lead to diverging solutions. Cause 2) stems from the reality that data acquired (from a subject) does not match what could be generated by the model (satisfying modeled dynamics) and the estimates of joint kinematics (from IK) does not take into the continuity of system dynamics from one instant to the next given discrete samples of position data. The largest source of discrepancies is the fact that different models are used to perform inverse dynamics and static optimization versus that of a forward simulation. Even when static optimization includes force-length and force-velocity relationships, the estimate of muscle length and velocity are determined by the length of the whole muscle-tendon unit (inelastic tendon) and activations do not satisfy excitation-to-activation dynamics present in forward.

9.2 How it Works

Computed muscle control (CMC) attempts to bridge the gap between forward and inverse dynamics by combining: PD feedback control to track experimental kinematics, static optimization to estimate the feedforward controls (muscle excitations) in order to generate desired accelerations at a small time (T) in the future, and then forward integration to generate new states and step forward in time.

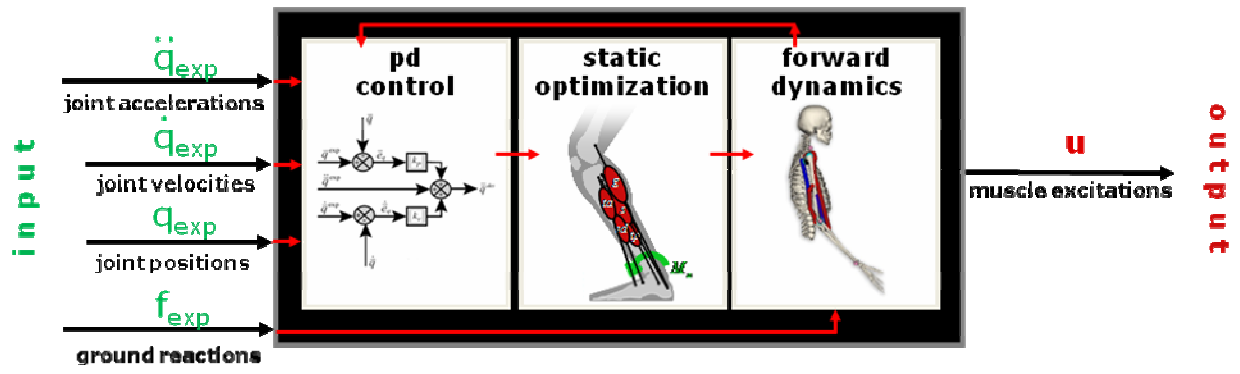


Figure 9-1: Overview of Computed Muscle Control

PD: $$\ddot{\vec{q}}^*(t+T) = \ddot{\vec{q}}_{exp}(t+T) + \vec{k}_v \cdot [\dot{\vec{q}}_{exp}(t) - \dot{\vec{q}}(t)] + \vec{k}_p \cdot [\vec{q}_{exp}(t) - \vec{q}(t)]$$

Two formulations of the static optimization problem are currently available in CMC. The first formulation, called the slow target, consists of a performance criterion () that is the sum of squared actuator controls (x_i) normalized by optimal force (F_o) plus the weighted sum of errors between desired acceleration ($\ddot{q}_j^*$) and actual acceleration ($\ddot{q}_j$):

Slow Target: $$J = \sum_{i=1}^{nx} \frac{x_i^2}{F_o} + \sum_{j=1}^{nq} w_j \left(\ddot{q}_j^* - \ddot{q}_j\right)^2$$

The second formulation, called the fast target, is the sum of squared controls augmented by a set of equality constraints (C_j) that requires the desired accelerations to be achieved within the tolerance set for the optimizer:

Fast Target: $$J = \sum_{i=1}^{nx} \frac{x_i^2}{F_o}\ ; \quad C_j = \ddot{q}_j^* - \ddot{q}_j = 0\ , \ \textit{for all } j$$

9.3 Inputs

The primary inputs to CMC consist of:

arm26_InverseKinematics.mot: Desired kinematics [.mot or .sto file] to be tracked

arm26_ComputedMuscleControl_Tasks.xml : Tracking tasks [.xml file] specifying which coordinates are to be tracked

(**arm26_CMC_Control_Constraints.xml**): Optional control constraints [.xml file] used to limit the allowed values of the actuator controls.

arm26.osim: The current model loaded in OpenSim.

(**arm26_Reserve_Actuators.xml**): Optional set of actuators (reserve are ideal torques) to append or replace the model's current set of actuators. Falls under the "Actuators and External Loads" tab of the **CMCTool**. In this case, ideal torques supplement muscles if muscles are unable to generate the required net joint moments.

9.4 Outputs

The following primary CMC outputs are placed in the specified output directory:

arm26_controls.xml: Actuator control [.xml file] (e.g., muscle excitations) computed by CMC that will drive a forward dynamic simulation.

arm26_controls.sto: Actuator controls [.sto file] computed by CMC in a format suitable for plotting.

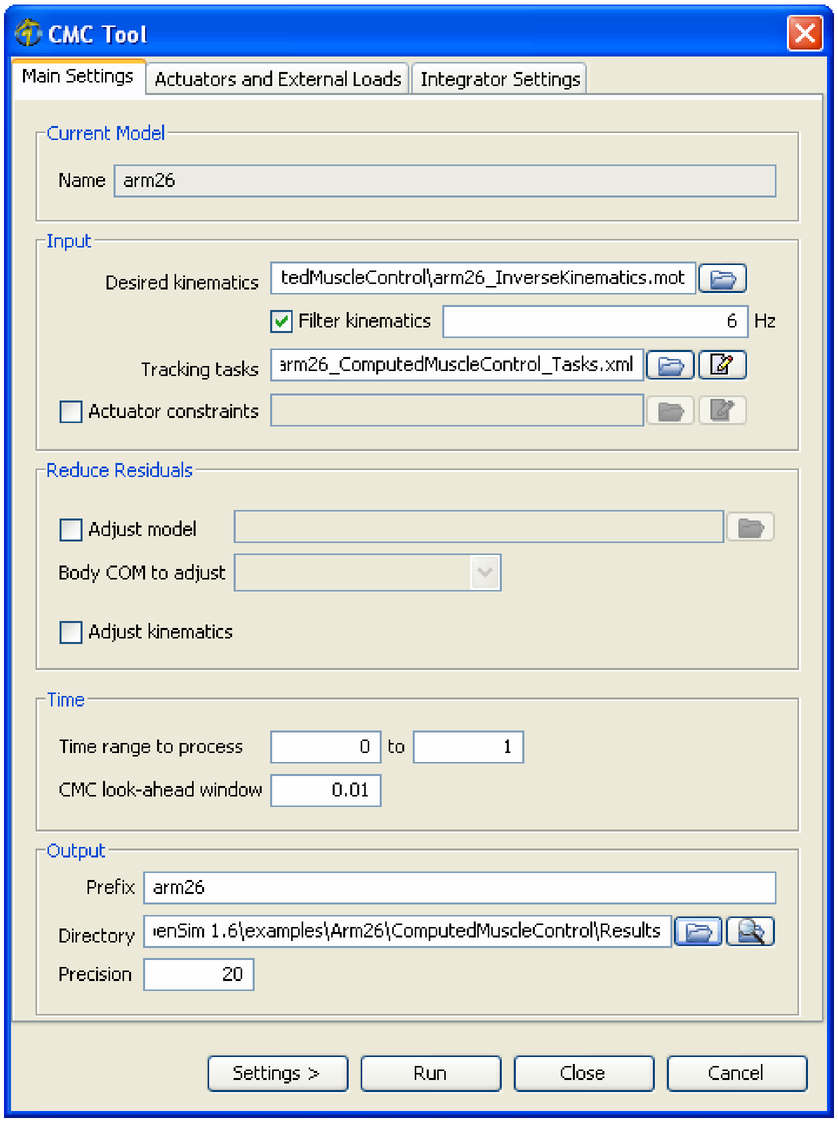
arm26_states.sto: Model states file [.sto file] containing the time histories of all model states that occurred during the CMC simulation.

arm26_Kinematics_q.mot: SIMM compatible joint motion file [.mot file] containing the time histories of the generalized coordinates resulting from CMC.

9.5 Computed Muscle Control (CMC) Tool

To launch the Computed Muscle Control Tool, select **Computed Muscle Control…** from the **Tools** menu. The Computed Muscle Control Tool is controlled by a dialog with three tabbed panes (Fig. 8-2). The **Main Settings** pane specifies parameters relating to the controls and states that will be input into the model, the time range for the simulation, and the output of the results. The **Actuators and External Loads** pane specifies the actuator set and the external loads applied to the model during the simulation. The **Integrator Settings** pane specifies integrator step sizes and tolerances used to solve the simulation. Limits on the range of controls can be defined by selecting the option **Actuator constraints** check box.

The Main Settings pane (Fig. 8-2) is organized into five main sections entitled **Current Model**, **Input**, **Reduce Residuals**, **Time**, and **Output**. The **Current Model** section displays uneditable information about the current model being used for analysis by the Computed Muscle Control Tool. The **Input** section displays editable information specifying the desred kinematics to be tracked by the CMC Tool. You may use the button to browse for the desired kinematics as either a storage (.sto) or motion (.mot) file. Filtering options are the next set of inputs, followed by the Tasks (.xml) file that specifies the kinematics to be tracked, their relative weightings and PD controller gains. The **Time** section displays editable information specifying the start and end time for the forward simulation during CMC as well at the look-ahead time window CMC uses to estimate accelerations in the future from current controls. The **Output** section displays editable information specifying the prefix appended to all of the resulting output files, the directory to which the files are saved, and the precision (number of decimal places) used when writing results. You may use the button to browse for a directory to save the output files, and the button to open an explorer to the specified directory.


CMC Tool

Main Settings | Actuators and External Loads | Integrator Settings

Current Model

Name: arm26

Input

Desired kinematics: tedMuscleControl\arm26_InverseKinematics.mot

[✓] Filter kinematics: 6 Hz

Tracking tasks: arm26_ComputedMuscleControl_Tasks.xml

[ ] Actuator constraints:

Reduce Residuals

[ ] Adjust model:

Body COM to adjust:

[ ] Adjust kinematics

Time

Time range to process: 0 to 1

CMC look-ahead window: 0.01

Output

Prefix: arm26

Directory: enSim 1.6\examples\Arm26\ComputedMuscleControl\Results

Precision: 20

Figure 9-2: Dialog for the Compute Muscle Control Tool. The main settings pane.

10 Example: Computed Muscle Control

10.1 Using Computed Muscle Control

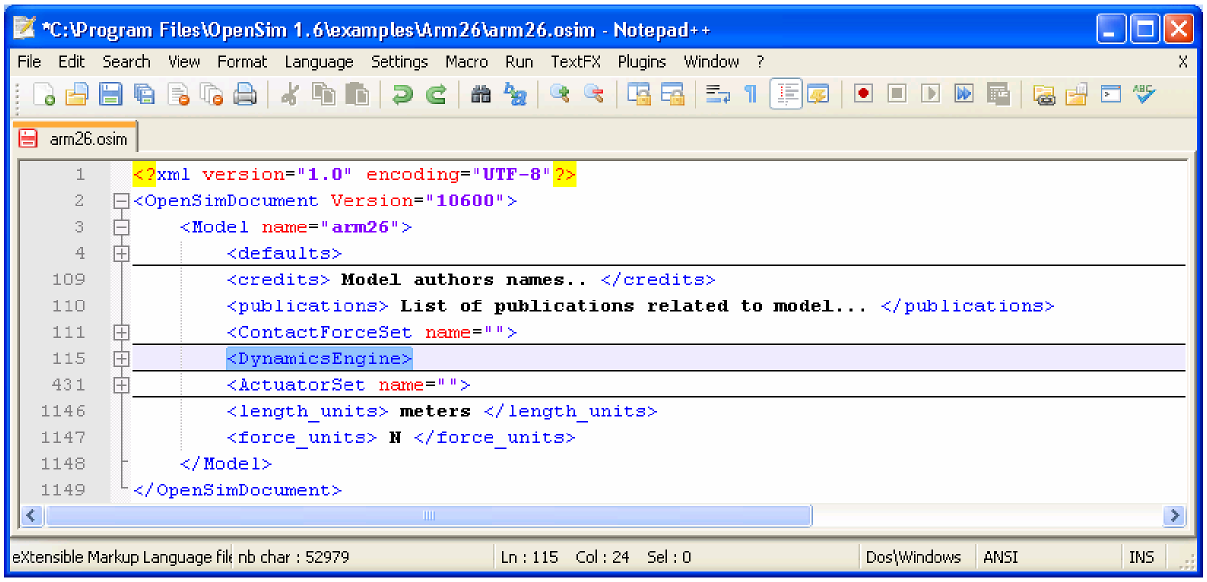
1. **Open Computed Muscle Control Tool**. To open the tool (Figure 7-1), select **Compute Muscle Control...** from the **Tools** menu.
2. **Specify Filtered Input Motion**. Browse to the inverse kinematics directory and select the **Desired kinematics** file (e.g., *..\InverseKinematics\arm26_InverseKinematics.mot*), check the **Filter kinematics** option, and enter a cutoff frequency of 6 Hz.
3. **Specify Tracking Tasks**. Browse to select the tasks file (e.g., *arm26_ComputedMuscleControl_Tasks*.xml) specifying the joint coordinates for CMC to track and their relative weightings as well as the Kp and Kv gains on the errors.
4. **Uncheck Adjust Model and Adjust Kinematics**. These options are used when performing residual reduction to obtain more dynamically consistent simulations. For our *Arm26* example, residual reduction is not necessary.
5. **Specify Time Range**. The **time range** for the forward simulation is specified and these should be set from 0 to 1 sec to correspond to the interval upon which the controls from static optimization were computed.
6. **Set CMC look-ahead window.** A time window of 0.01 is generally sufficient for muscle activations to change enough to produce the desired accelerations.
7. **Specify Output Directory**. Set the output **Directory** (e.g., *..\ComputedMuscleControl\Results*), so that you are able to compare the results of a CMC simulation with Forward Dynamics simulations using Nonphysiological and Physiological controls generated earlier by Static Optimization.
8. **Specify Additional Model Actuators**. Set the actuator settings to **Append** (rather than replace) and edit the **Additional actuator set files** field by adding an actuator set file (e.g., *arm26_Reserve_Actuators.xml*) containing reserve torques.

9. **Save Settings**. Use the **Settings >** button to save your settings to a setup file (e.g., *arm26_Setup_ComputedMuscleControl.xml*).
10. **Run and Close CMC Tool.** You will see the model begin to move as muscles contract and accelerate the model. When the simulation has completed by reaching the end of the specified time range, the specified output directory will be populated by states files (e.g., *arm26_states_degrees.mot*) and the corresponding motion file (e.g., *arm26_states*) will be associated with the model in the GUI.

11 Example: Model Editing

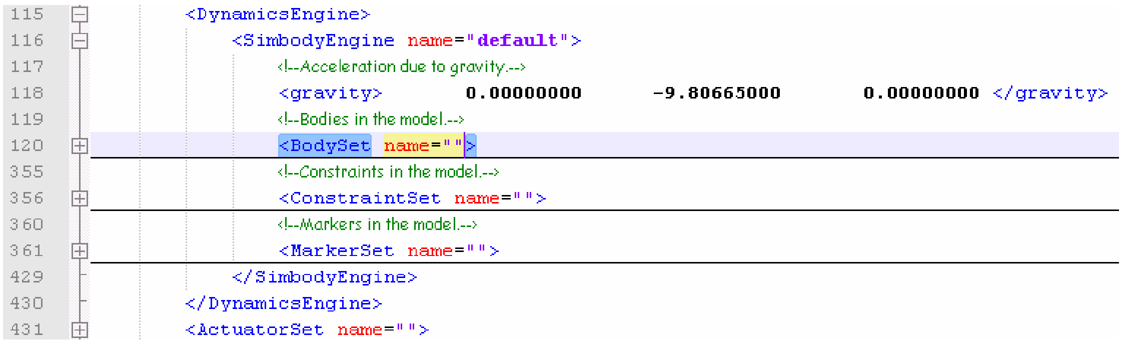
11.1 Connecting an Additional Segment to the Model

1. **Save Model File for Editing**. To save the model, select **Save Model As…** from the **File** menu. Specify a **File name** (e.g., *arm26_editable.osim*) and click the **Save** button.
2. **Open Model File for Editing**. Use an XML editor (e.g., Notepad++) to open the OpenSim model file (e.g., *arm26_editable.osim*). When collapsed to the 3rd level (e.g., **Alt+3** in Notepad++), you should see the following (*Note: the **DynamicsEngine** tag has been highlighted*):


*C:\Program Files\OpenSim 1.6\examples\Arm26\arm26.osim - Notepad++
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
arm26.osim
eXtensible Markup Language file nb char : 52979
Ln : 115 Col : 24 Sel : 0
Dos\Windows ANSI INS

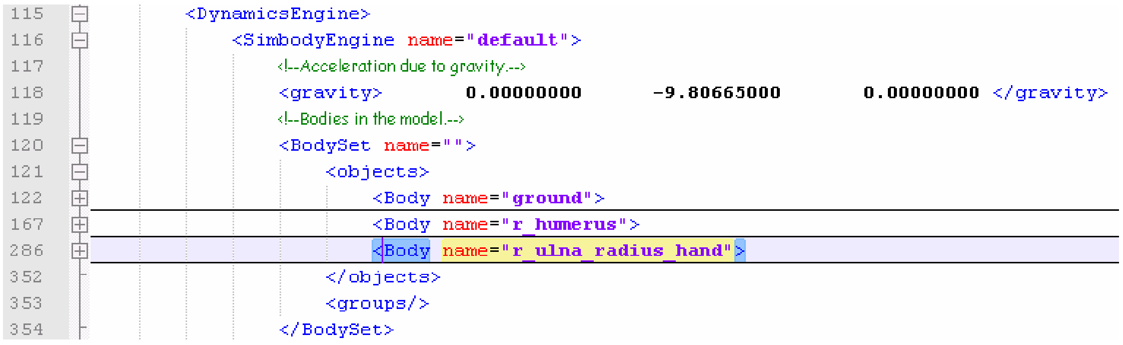
```
<?xml version="1.0" encoding="UTF-8"?>
<OpenSimDocument Version="10600">
    <Model name="arm26">
        <defaults>
        <credits> Model authors names.. </credits>
        <publications> List of publications related to model... </publications>
        <ContactForceSet name="">
        <DynamicsEngine>
        <ActuatorSet name="">
        <length_units> meters </length_units>
        <force_units> N </force_units>
    </Model>
</OpenSimDocument>
```

3. **Explore DynamicsEngine Children.** The **DynamicsEngine** tag has four children named **gravity**, **BodySet**, **ConstraintSet**, and **MarkerSet**. *Note: the **BodySet** tag has been highlighted.*



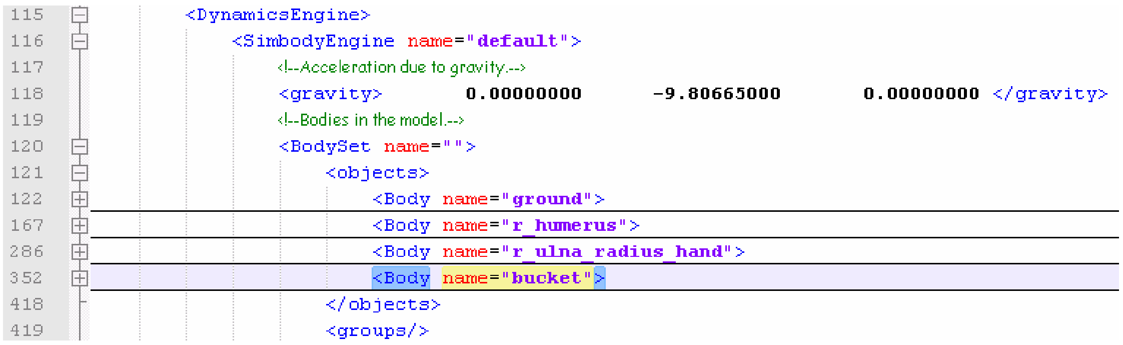
```
<DynamicsEngine>
    <SimbodyEngine name="default">
        <!--Acceleration due to gravity.-->
        <gravity>      0.00000000      -9.80665000       0.00000000 </gravity>
        <!--Bodies in the model.-->
        <BodySet name="">
        <!--Constraints in the model.-->
        <ConstraintSet name="">
        <!--Markers in the model.-->
        <MarkerSet name="">
    </SimbodyEngine>
</DynamicsEngine>
<ActuatorSet name="">
```

4. **Explore BodySet Children.** The **BodySet** tag has three grandchildren **Body objects** named **ground**, **r_humerus**, and **r_ulna_radius_hand**. *Note: the **Body** tag named **r_ulna_radius_hand** has been highlighted.*



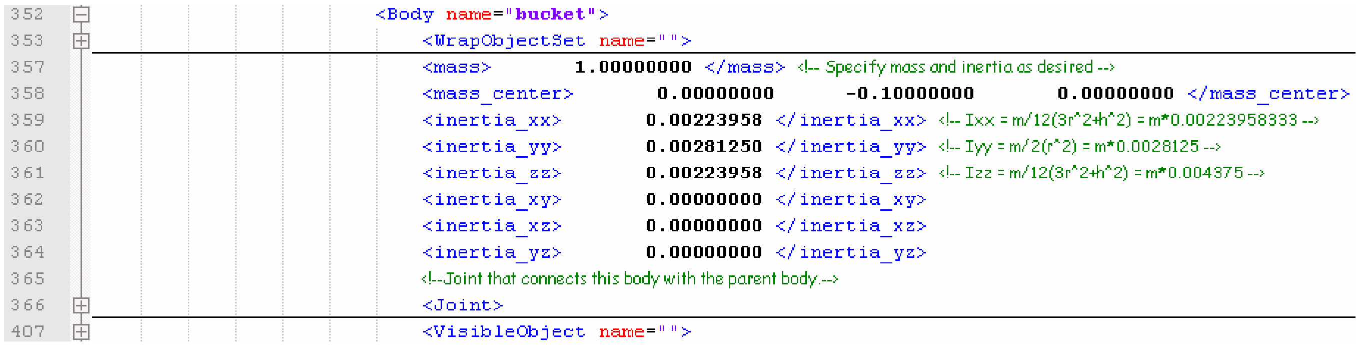
```
<DynamicsEngine>
    <SimbodyEngine name="default">
        <!--Acceleration due to gravity.-->
        <gravity>      0.00000000      -9.80665000       0.00000000 </gravity>
        <!--Bodies in the model.-->
        <BodySet name="">
            <objects>
                <Body name="ground">
                <Body name="r_humerus">
                <Body name="r_ulna_radius_hand">
            </objects>
            <groups/>
        </BodySet>
```

5. **Add New Body.** Highlight the **Body** named **r_ulna_radius_hand** along with all of its children and copy (**Ctrl+C**) to the Clipboard. Paste the Clipboard contents immediately below the **Body** named **r_ulna_radius_hand**. Rename the new **Body** to **bucket**. *Note: the new **Body** tag named **bucket** has been highlighted.*



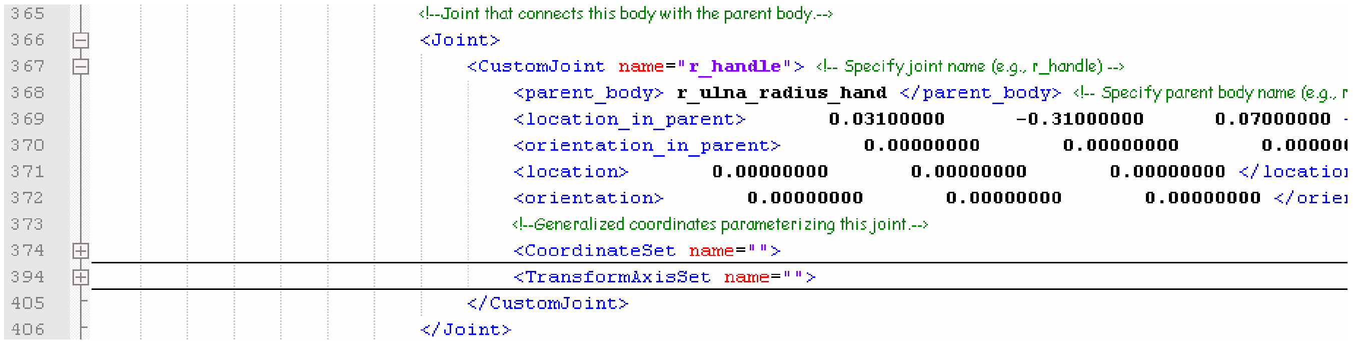
```
<DynamicsEngine>
    <SimbodyEngine name="default">
        <!--Acceleration due to gravity.-->
        <gravity>      0.00000000      -9.80665000       0.00000000 </gravity>
        <!--Bodies in the model.-->
        <BodySet name="">
            <objects>
                <Body name="ground">
                <Body name="r_humerus">
                <Body name="r_ulna_radius_hand">
                <Body name="bucket">
            </objects>
            <groups/>
```

6. **Specify Mass Properties.** Enter values for the **mass**, **mass_center**, **inertia_xx**, **inertia_yy**, and **inertia_zz** of the bucket as seen below:



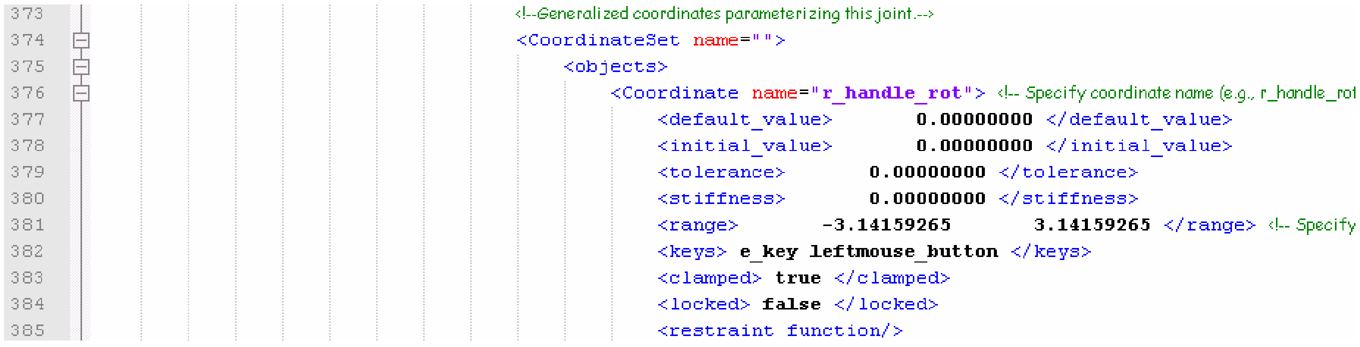
```
<Body name="bucket">
    <WrapObjectSet name="">
    <mass>      1.00000000 </mass> <!-- Specify mass and inertia as desired -->
    <mass_center>      0.00000000      -0.10000000      0.00000000 </mass_center>
    <inertia_xx>      0.00223958 </inertia_xx> <!-- Ixx = m/12(3r^2+h^2) = m*0.00223958333 -->
    <inertia_yy>      0.00281250 </inertia_yy> <!-- Iyy = m/2(r^2) = m*0.0028125 -->
    <inertia_zz>      0.00223958 </inertia_zz> <!-- Izz = m/12(3r^2+h^2) = m*0.004375 -->
    <inertia_xy>      0.00000000 </inertia_xy>
    <inertia_xz>      0.00000000 </inertia_xz>
    <inertia_yz>      0.00000000 </inertia_yz>
    <!--Joint that connects this body with the parent body.-->
    <Joint>
    <VisibleObject name="">
```

7. **Specify Joint.** Enter names for the **CustomJoint** and **parent_body**, and enter values for **location_in_parent** as seen below:



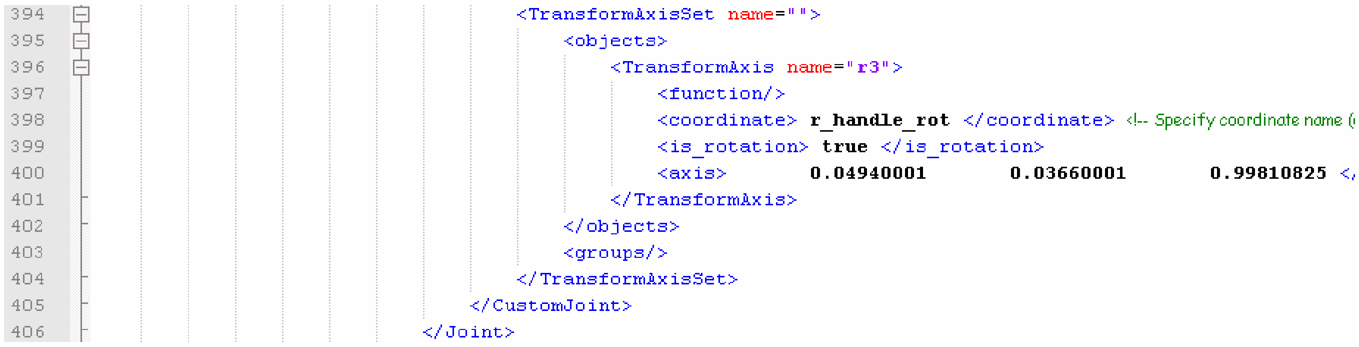
```
<!--Joint that connects this body with the parent body.-->
<Joint>
    <CustomJoint name="r_handle"> <!-- Specify joint name (e.g., r_handle) -->
        <parent_body> r_ulna_radius_hand </parent_body> <!-- Specify parent body name (e.g., r
        <location_in_parent>      0.03100000      -0.31000000      0.07000000 -
        <orientation_in_parent>      0.00000000      0.00000000      0.000000
        <location>      0.00000000      0.00000000      0.00000000 </locatio
        <orientation>      0.00000000      0.00000000      0.00000000 </orie
        <!--Generalized coordinates parameterizing this joint.-->
        <CoordinateSet name="">
        <TransformAxisSet name="">
    </CustomJoint>
</Joint>
```

8. **Specify Generalized Coordinate.** Enter name for the **Coordinate**, and enter values for **range** as seen below:



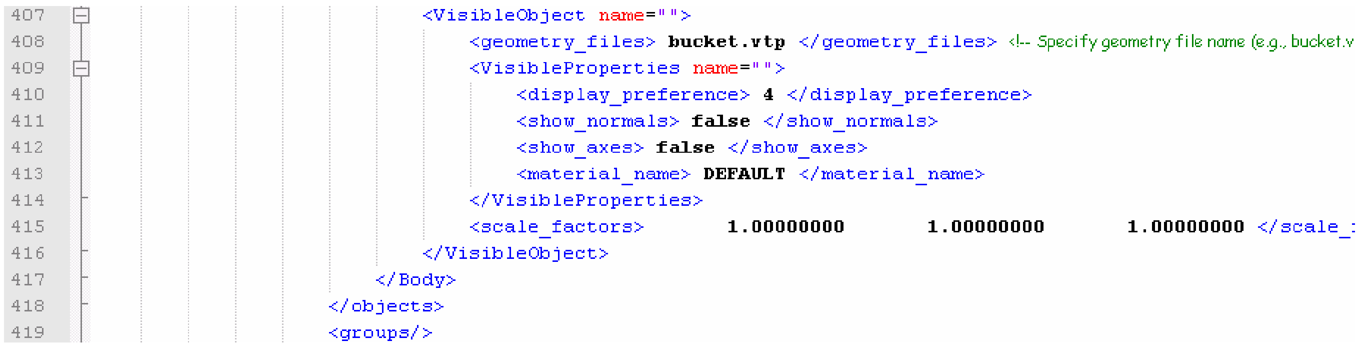
```
<!--Generalized coordinates parameterizing this joint.-->
<CoordinateSet name="">
    <objects>
        <Coordinate name="r_handle_rot"> <!-- Specify coordinate name (e.g., r_handle_rot
            <default_value>      0.00000000 </default_value>
            <initial_value>      0.00000000 </initial_value>
            <tolerance>      0.00000000 </tolerance>
            <stiffness>      0.00000000 </stiffness>
            <range>      -3.14159265      3.14159265 </range> <!-- Specify
            <keys> e_key leftmouse_button </keys>
            <clamped> true </clamped>
            <locked> false </locked>
            <restraint function/>
```

9. **Specify Joint Axis.** Enter name for **coordinate**, and enter values for **axis** as seen below:



```
          <TransformAxisSet name="">
            <objects>
              <TransformAxis name="r3">
                <function/>
                <coordinate> r_handle_rot </coordinate> <!-- Specify coordinate name (
                <is_rotation> true </is_rotation>
                <axis>       0.04940001       0.03660001       0.99810825 </
              </TransformAxis>
            </objects>
            <groups/>
          </TransformAxisSet>
        </CustomJoint>
      </Joint>
```

10. **Specify Geometry File.** Enter name for **geometry_files** as seen below:



```
    <VisibleObject name="">
      <geometry_files> bucket.vtp </geometry_files> <!-- Specify geometry file name (e.g., bucket.v
      <VisibleProperties name="">
        <display_preference> 4 </display_preference>
        <show_normals> false </show_normals>
        <show_axes> false </show_axes>
        <material_name> DEFAULT </material_name>
      </VisibleProperties>
      <scale_factors>       1.00000000       1.00000000       1.00000000 </scale_
    </VisibleObject>
  </Body>
</objects>
<groups/>
```

11. **Save Model File.** From the XML editor, save the OpenSim model file (e.g., *arm26_with_bucket.osim*).

12 Simulation Analysis

12.1 How it Works

The Analyze Tool uses the model together with states and/or controls to perform an analysis on a muscle-driven forward dynamics simulation (Chapter 8). An analysis consists of calculations based on the states and/or controls of the model to provide more in depth information about the performance of the model.

12.2 Inputs

Three files are required as input by the Forward Dynamics Tool:

arm26_controls.xml: Contains the time histories of muscle excitations (the controls in arm26) which can be used in conjunction with states to compute quantities of interest (such as muscle force per unit of excitation).

arm26_states.sto: Contains the model states including joint angles, joint velocities, muscle activations, and muscle fiber lengths from a forward simulation (including results of Computed Muscle Control). These states are used by the individual analyses setup with the Analyze tool.

Depending on the actuator type (e.g. muscle vs. an applied force) an analysis, such as Actuation, will report both controls (i.e. for an applied force, the force is the control) or quantities calculated from the states (i.e. muscle output force is calculated from tendon strain).

arm26.osim: The current model loaded in OpenSim.

12.3 Outputs

The **Analyze Tool** writes results of analyses associated with the model to files in the output directory.

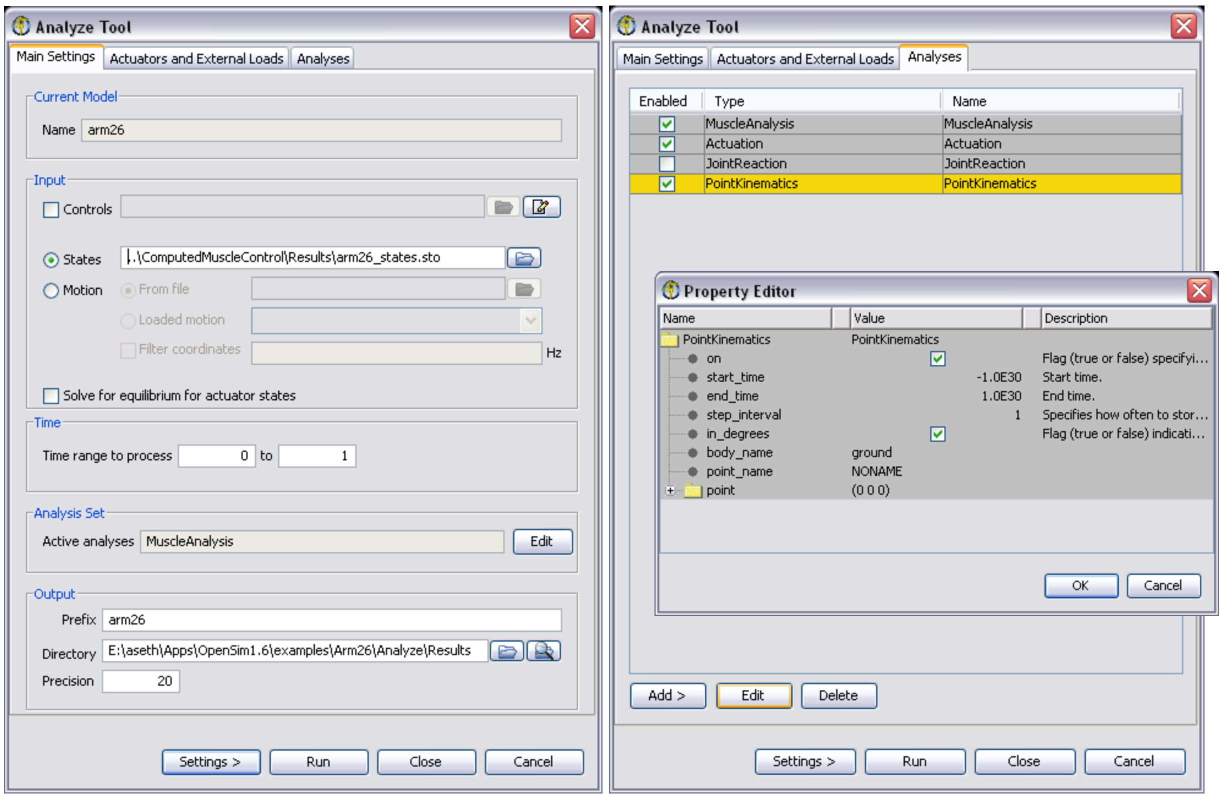
Every analysis in the specified by the Analyze Tool typically generates at least one file in the specified folder:

> **<modelPrefix>_<AnalysisName>_<outputType>.sto**: Contains the time histories of all the computed quantities. There are multiple file according to the number of analyses specified by the Analyze Tool and the number of output types (for example, there are three files corresponding to positions, velocities and accelerations in a Kinematics analysis).

12.4 Analyze Tool

To launch the Analyze Tool, select **Analyze…** from the **Tools** menu. The Analyze Tool is controlled by a dialog with three tabbed panes (Fig. 11-1.a). The **Main Settings** pane specifies files for the input controls and states of the model, the time range for the analysis(es), the set of analyses to be run and the output folder for the results. The **Actuators and External Loads** pane specifies the actuator set and the external loads applied to the model during the simulation. The **Analyses** pane (Fig. 11-1.b) enables the selection of analyses to be run by the AnalyzeTool from a list of analyses available in OpenSim including those contributed by user plugins.

The Main Settings pane (Fig. 11-1.a) is organized into four main sections entitled **Current Model**, **Input**, **Time**, **Analysis Set**, and **Output**. The **Current Model** displays information about the current model being used by the Analyze Tool. The **Input** section enables the entry of the input (optional) controls and states files to be used to run the analysis. The **Time** section enables the specification of the start and end time for the analysis. The **Output** section allows you to provide the prefix for all of the resulting output files, the directory to which the files are saved, and the precision (number of decimal places) used when writing results.



Analyze Tool — Main Settings

Current Model
Name: arm26

Input
Controls: [ ]
States: [✓] .\ComputedMuscleControl\Results\arm26_states.sto
Motion: [ ]
From file
Loaded motion
Filter coordinates Hz
Solve for equilibrium for actuator states: [ ]

Time
Time range to process 0 to 1

Analysis Set
Active analyses: MuscleAnalysis

Output
Prefix: arm26
Directory: E:\aseth\Apps\OpenSim1.6\examples\Arm26\Analyze\Results
Precision: 20

Analyze Tool — Analyses

Enabled	Type	Name
✓	MuscleAnalysis	MuscleAnalysis
✓	Actuation	Actuation
	JointReaction	JointReaction
✓	PointKinematics	PointKinematics

Property Editor

Name	Value	Description
PointKinematics	PointKinematics	
on	✓	Flag (true or false) specifyi...
start_time	-1.0E30	Start time.
end_time	1.0E30	End time.
step_interval	1	Specifies how often to stor...
in_degrees	✓	Flag (true or false) indicati...
body_name	ground	
point_name	NONAME	
point	(0 0 0)	

Figure 12-1: The Analyze Tool. The main settings (a) and analyses (b) panes.

The Analyses pane provides a list view of analyses associated with the tool, which is initially empty. By clicking **Add>** available analyses are added to the list. To edit the properties of an analysis, highlight it in the list and the click **Edit**, which opens a property editor. The property editor is a generic object editor that enables common property data types (bool, int, double, array of doubles, etc...) associated with an Analysis to be edited. For example, the body and point names (strings) and coordinates of the point (array of three numbers) for the PointKinematics analysis is shown in Figure 11-1.b.

13 Example: Analyzing Simulations

13.1 Analyzing Muscle Behavior from a Forward Dynamics Simulation

1. **Open Analyze Tool**. To open the tool (Figure 12-1), select **Analyze…** from the **Tools** menu.
2. **Specify States from a Forward Simulation**. Select a **States** file from an earlier forward dynamics simulations directory (e.g., *..\ForwardDynamics\PhysiologicalResulst\arm26_states.sto*) in the Analyze Tool.
3. **Specify Time Range**. The **time range** should be set from 0 to 1 sec to correspond to the interval for the forward dynamics simulation.
4. **Specify Output Directory**. Set the **Output Directory** (e.g., *..\Analyze\ForwardResults*), so that you are able to compare the results of a Forward Dynamics simulation using controls generated earlier by Static Optimization.
5. **Specify MuscleAnalysis**. From the Analyses pane (Figure 12-1) **Add >** MuscleAnalysis to the list of analyses.
6. **Specify Muscles to Analyze**. Select the MuscleAnalysis (only member of the Tool's list) and click the **Edit** button. In the PropertyEditor click the + icon to add a muscle to the muscle_list. Add *TRIlong* and *BIlong* be particularly careful of typos, since an exact match is required to report results for that muscle.
7. **Specify moment_arm_coordinates**: Since both muscles are biarticular their behavior at the shoulder and the elbow is of interest, so keep the default (ALL) setting.
8. **Finalize properties.** Click the **OK** button.
9. **Run Analyze Tool.** You will see the model begin to move as it analyzes the states. When the analysis has completed by reaching the end of the specified time range, the

specified output directory will be populated by muscle analysis storage files.

13.2 Analyzing Muscle Behavior from Computed Muscle Control

1. **Specify New States from Computed Muscle Control**. Select a new **States** file from an earlier computed muscle control directory (*..\ComputedMuscleControl\Results\arm26_states.sto*) in the Analyze Tool.
2. **Specify New Output Directory**. Set the new **Output Directory** (e.g., *..\Analyze\CMCResults*), so that you are able to compare the results of a Computed Muscle Control simulation with a Forward Dynamics simulation using controls generated earlier by Static Optimization.
3. **Run Analyze Tool.**

13.3 Comparing Muscle Behavior across Multiple Simulations

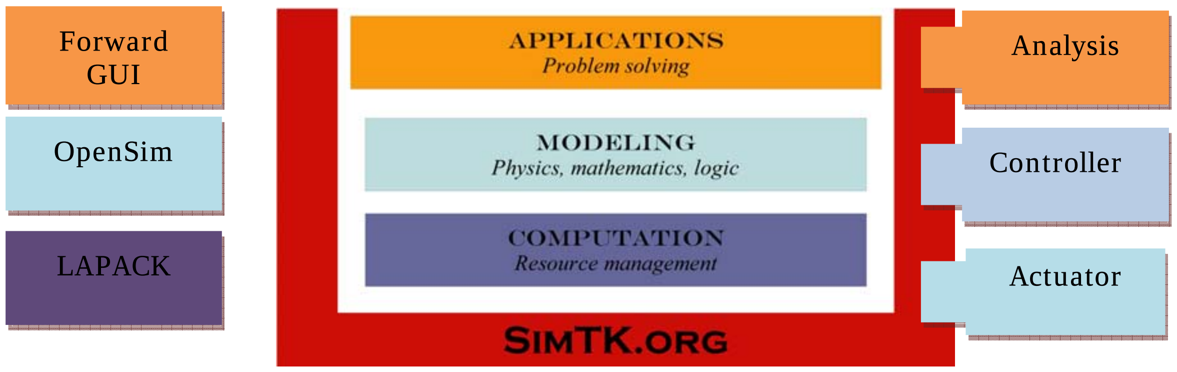
1. **Plot Muscle Moments from Forward Dynamics**. Browse to the output directory specified above (e.g., *..\Analyze\ForwardResults arm26_MuscleAnalysis_Moment_r_elbow_flex.sto*). Plot *TRIlong* and *BIClong* versus *time*. Leave **Plotter** dialog open to compare subsequent muscle moments.
2. **Plot Muscle Moments from Computed Muscle Control**. Browse to the output directory specified above (e.g., *..\Analyze\CMCResults arm26_MuscleAnalysis_Moment_r_elbow_flex.sto*). Plot *TRIlong* and *BIClong* versus *time*.

How do the muscle moments from the Forward Dynamics simulation using controls generated earlier by Static Optimization compare to the Computed Muscle Control simulation?

14 Extending OpenSim's Capabilities: User Plugins

14.1 Organization of OpenSim with SimTK

OpenSim is built on the computational and simulation core provided by SimTK. These include low level efficient math and matrix algebra libraries such as LAPACK. At the modeling layer, Simbody™ is a powerful multibody dynamics solver. OpenSim is yet a higher modeling layer which maps biomechanical structures (bones, muscles, tendons, etc...) into bodies and forces so that the motion of the structure can be resolved by Simbody.



14-1: Organization of Simulation Tools in SimTK.

OpenSim is essentially a set of modeling libraries for building complex actuator (muscle) force generators and capturing the motion (kinematics) of highly articulated bodies (bones) that can be controlled by model controllers (Computed Muscle Control) to estimate the neural control and muscle forces required to produce human movement. At the highest level these blocks are assembled into specialized applications (ik.exe, forward.exe, analyze.exe) to simulate and analyze model movement and internal dynamics.

14.2 OpenSim Architecture and Important Objects

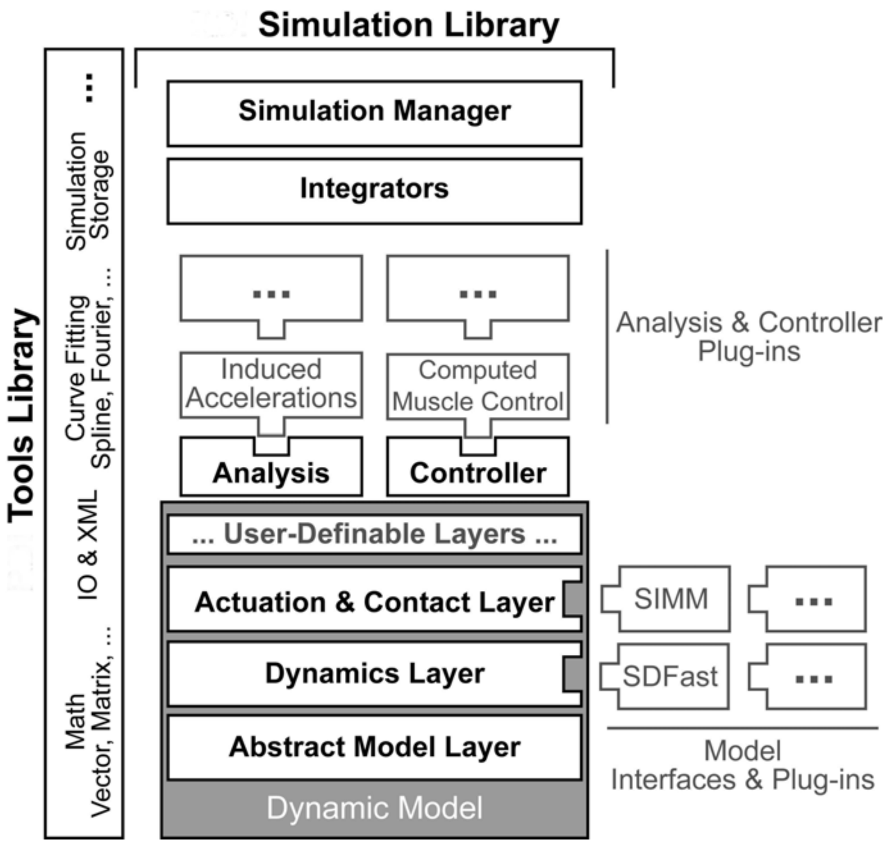


Figure 14-2: Schematic of the OpenSim Architecture

OpenSim specifically targets, actuators and contact as plugin components of a model used to define the dynamics of the model. At the system level, controllers that dictate how the model will move and analyses that quantify various measures of performance can also be developed as plugins, to enable a user or outside application to extend OpenSim’s capabilities.

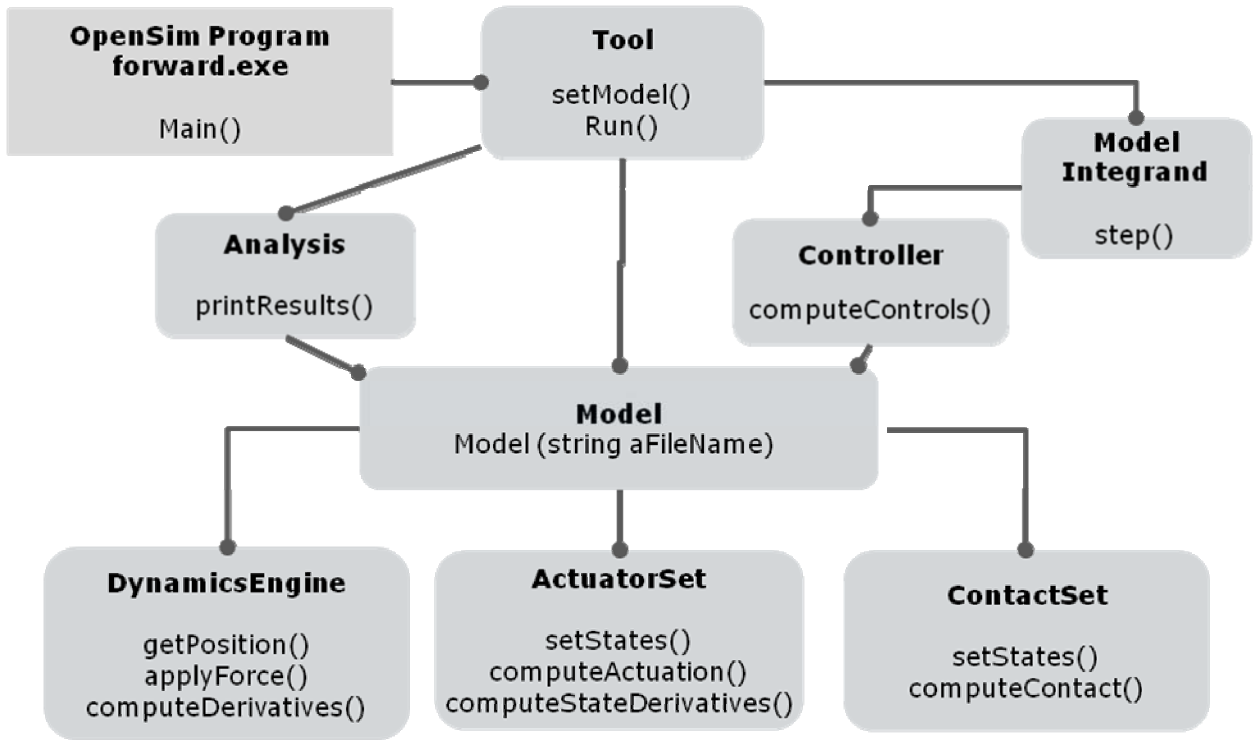


Figure 14-3: Hierarchy of Important OpenSim Objects

In order to build custom components it is necessary to have a general understanding which objects (classes) are responsible for what actions/behaviors. The functions (methods) that OpenSim's public classes provide (so outside applications/programs can call) define its Application Programming Interface or API. Figure 13-3 is an overview of relationships between the main objects that provide the API. The best way to obtain details about the methods each object provides, refer to the Doxygen generated documentation in <OpenSimInstallDir>/sdk/doc/html.

14.3 Steps to Build Plugins for OpenSim

1. **Install Visual C++** 2008 Express Edition freely available from http://www.microsoft.com/express/download/ if you do not already have Visual C++ 8.0 (2005) or 9.0 (2008) setup on your computer.
2. **Install CMake.** CMake is a cross-platform open-source build system that will setup the build environment for creating an OpenSim plug-in. CMake is freely available from http://www.cmake.org/files/v2.6/cmake-2.6.1-win32-x86.exe.

3. **Dowload OpenSim 1.6** at https://simtk.org/project/xml/downloads.xml?group_id=331 and make note where you have installed OpenSim (referred to as <OpenSimInstallDir>).
4. **Prepare your development folder.** Copy the <OpenSimInstallDir>/sdk/examples/plugin directory into a folder (work space) outside of the OpenSim installation so that future uninstalls and installs of OpenSim do not destroy your work. Any empty folder will do, for example, C:/OpenSimPlugin/ would be easy to recognize.
5. **Run CMake.** Select the plugin folder you just copied for "Where is the source code:" and for "Where to build the binaries:" select a build directory that is convenient, like C:/OpenSimPlugin/build. Click **Configure** which will setup CMake so it creates a build profile compatible with your version of Visual C++, which you must select from the list it presents. Additional settings that CMake requires to proceed will appear in red. For the CMAKE_INTSTALL_PREFIX field, select <OpenSimInstallDir>, which tells Visual C++ where to install your plug-in also set the <OpenSimInstallDir> for the OPENSIM_INTSTALL_DIR field, which identifies where OpenSim and its libraries live. Click **Configure** again and when there are no fields in red, the OK button will be enabled. Click **OK**. This will setup all the necessary build files in your build directory.
6. Open the **OsimPlugin.sln** from your build directory which will **launch Visual C++** with OsimPlugin as a project. Change the Solution Configuration from Debug (default) to **Release**.
7. **Build solution** (from the Build menu) which will compile the template analysis into a .dll (plugin). Follow that with a build **Install** which will install the osimplugin.dll into <OpenSimInstallDir>/plugins.
8. **Launch OpenSim** and load the plug-in from Tools->User Plugins, by clicking on the osimplugin.dll. This will confirm that the plug-in is available to use in OpenSim.

15 Example: Creating Your Own Analysis

15.1 Build a Body Position Analysis from the Template

1. **Rename Template.** In your plugin directory (e.g., *C:\OpenSimPlugin\plugin*), rename the *AnalysisPlugin_Template.h* and *.cpp* to *MyAnalysis.h* and *.cpp* (or any other name that is unique from built-in analyses). The template analysis simply reports the center-of-mass position of selected bodies.
2. **Run CMake.** Click **Configure** and then **OK**.
3. **Open the OsimPlugin.sln** which will **launch Visual Studio** from the solution file. Do a search and replace (in the entire solution) to replace *AnalysisPlugin_Template* with (or the name you gave your analysis).
4. **Build** solution (from build menu) which should compile your analysis into a dll (plugin) and follow that with a build Install (should install in <OpenSimInstallDir>/plugins).
5. **Launch OpenSim** and load the plugin from **Tools->User Plugins**, by clicking on the *osimplugin.dll.*
6. **Setup Analysis for Model.** Load arm26 and perform the analysis with the AnalyzeTool and choose your analysis from **Add>** list. Select states from any previous simulation and an output directory.
7. **Run it!** ... *arm26_MyAnalysis_pos.sto* should appear in the output directory containing the COM location and body rotation for each modeled body segment.
8. Celebrate, you just added and ran your own analysis plugin!

15.2 Build a Body Position, Velocity, and Acceleration Analysis

1. **Declaring additional output storage and internal working arrays**. The number of outputs have changed. Before we had one storage file with position data:

   ```
   /** Storage for recording body positions. */
   Storage _storePos;
   ```

   and an,

   ```
   /** Internal work array to hold the computed positions. */
   Array<double> _bodypos;
   ```

   Add additional storage for velocities and accelerations and provide a work array for accelerations in the .h file.

2. Update the description of the Analysis in `constructDescription()` in the .cpp file.

3. Setup the storage for the velocity and acceleration results:

   ```
   setupStorage()
   {
   // Positions
   _storePos.reset(0);
   _storePos.setName("Positions");
   _storePos.setDescription(getDescription());
   _storePos.setColumnLabels(getColumnLabels());
   …
   ```

4. Correctly size working arrays :

   ```
   setModel(Model *aModel)
   { …
         int numBodies = _model->getNumBodies();
         _kin.setSize(6*numBodies);
         …
   ```

5. An analysis' `record()` method is the heart of the analysis. It collects or computes the data necessary to perform and output the results of an analysis. It requires adding a calculation (call to the DynamicsEngine) to get the model accelerations

```
/*
 * Compute and record the results.
 *
```

```
* This method, for the purpose of example, records the position and
* orientation of each body in the model. You can customize it
* to perform your analysis.
*
* @param aT Current time in the simulation.
* @param aX Current values of the controls.
* @param aY Current values of the states.
*/record(double aT,double *aX,double *aY)
  {
        // GET THE MODEL READY ----------------------------------
        // Set the configuration of the model.
        _model->set(aT,aX,aY);
        …

        // Comput and apply all actuator forces.
        _model->getActuatorSet()->computeActuation();
        …

        // compute and apply all contact forces.
        _model->getContactSet()->computeContact();
        …

        // After setting the state of the model and applying forces
        // Compute the derivative of the multibody system (speeds and
        // accelerations)
        // NOTE: computeDerivatives() on the dynamicsEngine must be
        // called before getting acclerations and reaction forces.

        // API (Doxygen html files) for AbstractDynamicsEngine can be
        // found in <OpenSimInstallDir>/sdk/doc/html
        <add here>

        …

        // POSITION
        BodySet *bodySet = _model->getDynamicsEngine().getBodySet();
        int numBodies = bodySet->getSize();
        for(int i=0;i<numBodies;i++) {

              AbstractBody *body = bodySet->get(i);
              SimTK::Vec3 com;
              body->getMassCenter(com);

              // GET POSITIONS AND EULER ANGLES
              _model->getDynamicsEngine().getPosition(*body,com,vec);
              _model->getDynamicsEngine()
                          .getDirectionCosines(*body,dirCos);
              _model->getDynamicsEngine()
                    .convertDirectionCosinesToAngles(dirCos,
                    &angVec[0],&angVec[1],&angVec[2]);

              // CONVERT TO DEGREES?
              if(getInDegrees()) {
                    angVec[0] *= SimTK_RADIAN_TO_DEGREE;
                    angVec[1] *= SimTK_RADIAN_TO_DEGREE;
```

```
                angVec[2] *= SimTK_RADIAN_TO_DEGREE;
            }

            // FILL KINEMATICS ARRAY
            int I=6*i;
            memcpy(&_bodypos[I],&vec[0],3*sizeof(double));
            memcpy(&_bodypos[I+3],&angVec[0],3*sizeof(double));
        }
        _storePos.append(aT,_bodypos.getSize(),&_bodypos[0]);

        // VELOCITY
```

Repeat process for velocity and accelerations. Check Doxygen for calls to the DynamicsEngine to get velocities and accelerations.

```
        // API (Doxygen html files) for AbstractDynamicsEngine can be
        // found in <OpenSimInstallDir>/sdk/doc/html
```

6. In `begin()` reset the storage objects at the specified time.

```
    // RESET STORAGE
    _storePos.reset(aT);
```

7. An analysis is finalized by printing results out to file:

```
/*
 * Print results.
 *
 * The file names are constructed as
 * aDir + "/" + aBaseName + "_" + ComponentName + aExtension
 *
 * @param aDir Directory in which the results reside.
 * @param aBaseName Base file name.
 * @param aDT Desired time interval between adjacent storage vectors.
 *       Linear interpolation is used to print the data out at the
 *       desired interval.
 * @param aExtension File extension.
 *
 * @return 0 on success, -1 on error.
 */

printResults(const string &aBaseName,const string &aDir,double aDT,
                      const string &aExtension)
{
   // POSITIONS
   _storePos.scaleTime(_model->getTimeNormConstant());
   Storage::printResult(&_storePos,aBaseName+"_"+getName()+"_pos",aDir,
                        aDT,aExtension);

   // VELOCITIES
   …
```

8. Compile and debug in Visual Studio.

9. Build install when satisfied (will overwrite the previous osimplugin.dll)

10. Restart OpenSim, load plugin and run your analysis with arm26.