QSolver Quickstart Guide

# Introduction

The QSolver tools are a set of scripts for Maya, written in python, that allow for the reading, editing and writing of QTM solver XML files. The XML reader constructs a Maya scene from a solver definition file. You can then edit the scene elements to adjust the skeleton to be solved. The exporter writes out a solver XML file that you load into QTM for solving with your data. This guide provides multiple examples of variations on this process to get you familiar with the procedures for editing solver definitions. But first, some terms, definitions, and requirements:

* **QTM** – The Qualisys Track Manager. The main editing and analysis software at Qualisys. It also runs solver.
* **Solver** – The globally optimized solver tool for calculating skeleton motion from marker data. It is built in to QTM.
* **Maya** – Autodesk’s Maya 2020 is the version used for the initial development of these scripts. They have been tested through version 2023.
  + It is assumed in this guide that you have some familiarity with using Maya.
  + Installation of the scripts is easy:
    - Automatic:
      * Follow the instructions for installing qtm\_connect\_maya and these scripts and icons for them will be also installed.
    - Manual:
      * Copy the scripts to the Maya scripts folder
      * Make shortcut icons on a shelf in Maya for the QImportSolver and QExportSolver scripts.
* **Python –** All the scripts are written in Maya python which now uses (in Maya 2023) python version 3.9.7
* **Namespace** – A **namespace** is a simple grouping of objects under a given **name**. **Namespaces** are primarily used to resolve name-clash issues in **Maya**, where a new object has the same name as an existing object (from importing a file, for example). Using **namespaces**, you can have two objects with the same name, as long as they are contained in different **namespaces**. The name policy for markersets in QTM is that the namespace in Maya (and Motionbuiilder) is the same as the markerset name.
* **Markers** – These are locators in Maya whose positions come from a frame of marker data in QTM. These can come from the XML file or they can be created via other methods, such as importing an FBX file.
* **Skeleton** – The hierarchy of joints in Maya that define the skeleton portion of the model pose.
* **Character/Avatar** – These terms are used interchangeably. They refer to a user defined character/avatar rig that is independent of any description of the performer. They will almost always consist of a skeleton and a skin that is bound to the skeleton. Optionally they might have a reference node that is used to place/orient the character in the scene. This reference node is not driven by mocap data.
* **Poses**
  + **T-Pose** – A standard pose for many calculations. All retargeting tools (HIK, MotionBuilder) require a T-Pose for characterization as a way of relating different characters to each other. Some motion capture tools require this pose. QTM typically uses a T-Pose for calculating the animation (and sports) skeleton, although an A-Pose can be used as well.
  + **A-Pose** – A common alternative to the T-Pose for binding the skeleton to a skin. The shoulders are easier to skin in a relaxed position compared to the T-Pose.
  + **Zero Pose** – The pose of the skeleton when all joint rotation values are set to zero. Often this is equal to the T-Pose, but it doesn’t have to be.
  + **Bind Pose** – The Maya pose for binding a skeleton to a skin. Most often this is the T-Pose, but many users prefer an A-Pose here.
  + **Model Pose** – The combination of markers and skeleton that define the offsets from the skeleton segments to the markers. This is usually a T-Pose, but it doesn’t have to be.

# QuickStart Samples

A series of examples for helping the user understand the solver model pose setup process.

## Adjusting the Animation Skeleton

This is an example which shows how to load an animation XML file, make a simple change to the model pose in Maya, then export back to QTM for solving. This assumes you have downloaded the sample dataset *Skeleton Solver Demo – Animation Marker Set.* Be sure to have a version of QTM that supports roll bones in the animation markerset (2020.3 or later) and a dataset version that matches (a QAvatar with roll bones in the forearms).

How to adjust an existing animation skeleton in QTM:

1. Load the file “EliROM.qtm” in to QTM
   1. Start QTM
   2. Select “Load Project” and choose the folder with the animation data
   3. Select EliROM.qtm
2. Export the Eli XML file to “Eli.xml”
   1. Open the Project Settings Dialog
   2. Select “Skeleton Solver”
   3. From the Name list select Eli
   4. Select Save
   5. Choose the file “Eli.xml” to save, for convenience put it in the “Data” folder
3. Using the “QImportSolver.py” script in Maya, load the file “Eli.xml”
   1. Start Maya
   2. Set the current project folder to the “Data” folder of the sample dataset. Create “default project settings” for Maya.
   3. Note that QTM typically works in Z up world axis (the example data is Z up), Maya is typically Y up, but for this you should change your preference in Maya to Z up.
   4. Select the icon for QImportSolver. If you don’t have this, you can either just source the python script file or you can load the file and execute it from the script editor.
   5. Choose the file “Eli.xml”
4. Make a change to the skeleton
   1. Select the RightShoulder joint and rotate it a small amount backwards so the RightArm joint is closer to the RShoulderBack marker instead of being directly under the RShoulderTop marker.
   2. Counter rotate the UpperArm joint to put the RightHand joint back between the hand markers.
5. Load the change back into QTM
   1. Use the script QExportSolver to save out a new XML file.
      1. Select the root joint “Hips”
      2. Select QExportSolver
      3. Specify the file name to save, perhaps “Eli\_modified.xml”
   2. In QTM select “Reprocess”
      1. Turn off all options except for “Solve Skeletons”
      2. Select “Skeleton Solver”
      3. Select “Load”
      4. Choose “Eli\_modified.xml” (note that it comes back in as “Eli” in the list because that is the name saved in the XML file (which matches the namespace in Maya)
      5. Select “OK” and the skeleton will be recalculated.
6. Note that the solver results have been modified.

Some images:

|  |
| --- |
|  |
|  |

Above: Where to find the [sample data](https://qfl.qualisys.com/#!/project/skeleton-solver). Go to the Qualisys website and look for the “Demo Files” link in the upper left. Locate and download the ***Skeleton Solver Demo – Animation Marker Set***sample data.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Above: images for Step 1. Load the original EliROM.qtm file. Note the position of the clavicle bones relative to the shoulder markers

|  |  |
| --- | --- |
|  |  |
|  |  |

Above: Images for Step 2. Export the Eli skeleton solver to an XML file.

|  |
| --- |
|  |

Above: Image for step 3. The QImportSolver icon is on the QTM\_Connect shelf. Yours will be where ever you put it. Note the namespaces for the “Markers” and “Hips”

|  |  |
| --- | --- |
|  |  |
|  |  |

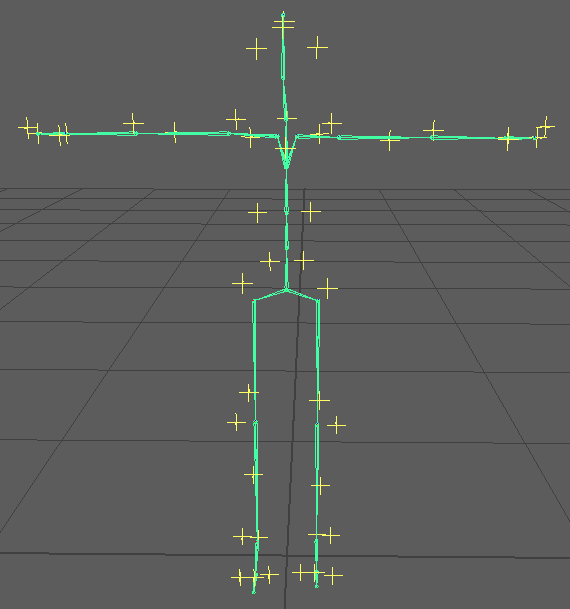
Above: Images for step4. Top image is the skeleton right after importing the XML. The bottom image is after rotational adjustments have been made.

|  |
| --- |
|  |
|  |
|  |

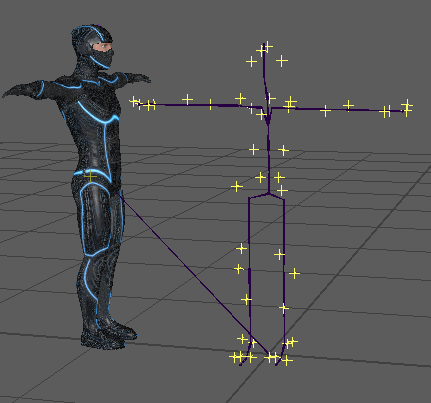
Above: Images for step 5. Note the change in the position of the clavicles.

## Streaming to the QAvatar

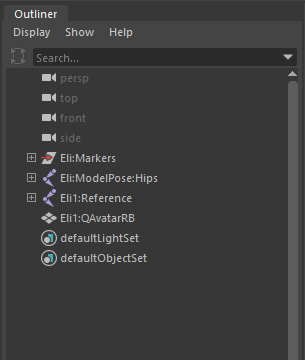
This shows how to take the above process and stream it onto the QAvatar with roll bones.



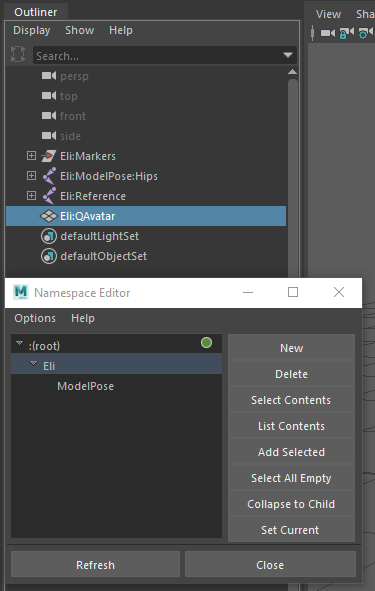
Above image, start with the modified skeleton from the first quickstart outline.



Import the QAvatar Maya file. Depending on your import settings you will get a modification to the names in the QAvatar file. Below is shown when the file is imported with namespaces.

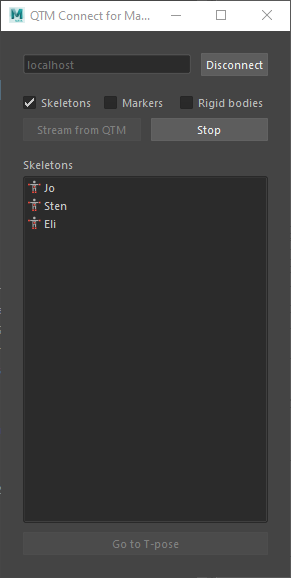
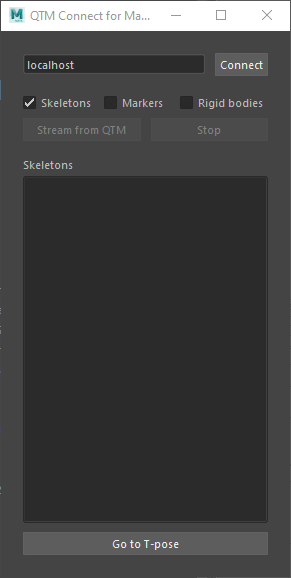


However the names were imported, the goal is to get the avatar into the “Eli” namespace without any prefixes. Use the Maya Namespace Editor to delete any invalid namespaces (such as “Eli1” in the above image) and put the QAvatar into the “Eli” namespace.



What your scene’s hierarchy should look like. The original Eli setup and the new Reference and QAvatar nodes from the QAvatar file. Note that there are two identical copies of the skeleton in the scene- the one in the “Eli” namespace and one under the “ModelPose” sub-namespace. This is how the name clashes are avoided.

Set QTM to “Play with Realtime Output”. This starts steaming data from QTM. In Maya, locate the QTM\_Connect shelf. Select the “Connect to QTM” icon to bring up the connection dialog.

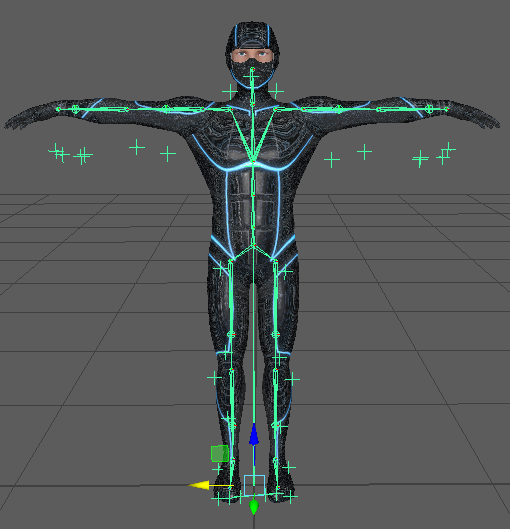


The QTM\_Connect dialog before and after selecting “Connect” and “Stream from QTM”



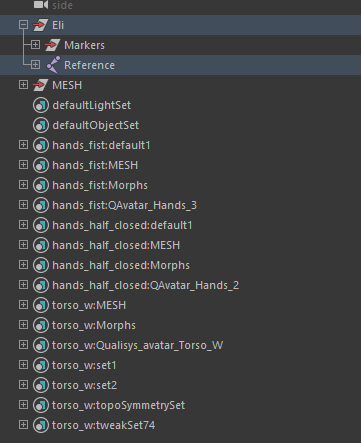
The result of streaming the modified skeleton from the first quickstart outlne. This is the animation skeleton with modifications.

Now on to creating a model pose directly from the QAvatar model.



To get started, recreate the scene as depicted in the above image. Do this with the following steps:

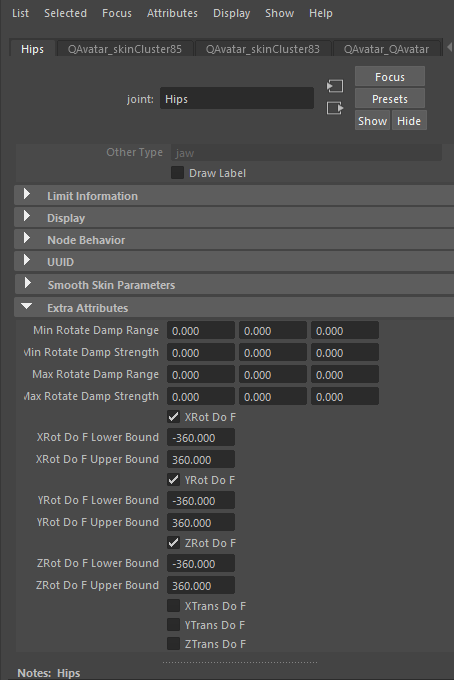
* Start with a new scene
* Do QImportSolver on the Eli.xml file to create the model pose.
* Import QAvatar to merge it with the model pose
* Remove any namespace or prefix from the imported QAvatar, do not put the “Eli” prefix on it.
* Rearrange the hierarchy of the scene to replace the old skeleton with the new one
  + Delete the old model pose skeleton
  + Position the QAvatar in its place by only modifying the Hips node. Do not modify the “Reference” node.
  + Place the QAvatar joint hierarchy under the “Eli” root node (make “Reference” a child of “Eli”)



After rearranging the scene to be as above. You need to add DOF attribute information to all the joints. This uses a python helper script called “AddDOFAttributes.py”.

* Select the node “Hips” in the joint hierarchy.
* Use the Maya menu item “Select->Hierarchy” to select all the joints in the hierarchy.
* Invoke AddDOFAttributes.py

Select any joint to see the attributes that have been added. Here the Hips joint is shown:



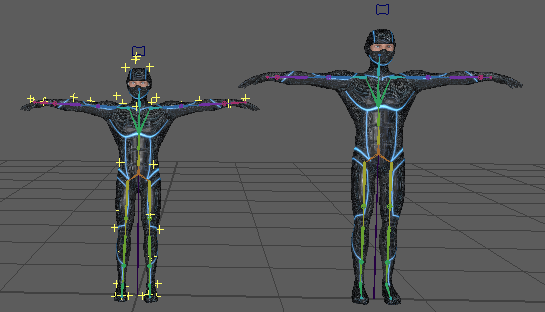
Now you must adjust the settings for a number of different joints.

* Turn on the translation dofs for the root.
* Turn off all dofs for the roll bones.
* Turn off “Segment Scale Compensate” for all the joints so the whole skeleton can be scaled from the hips joint.
* Scale the Hips joint by 0.86 in all axes. That is what works for this combination of character and performer. Other combinations will be different. But be sure to make all adjustments from the Hips node and not the parent “Reference” node.
* Add end effector joints for the hands, head and toes.
  + Select the end joint.
  + Duplicate it
  + Parent the new joint to the original
  + Rename the new joint the same as the original with “\_end” added.
  + This creates a “EndPoint” tag for end effectors to make the segment draw correctly in QTM

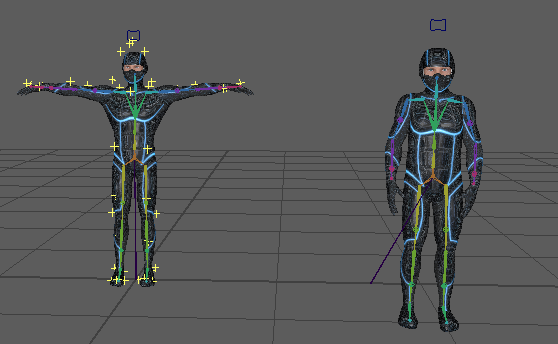
|  |
| --- |
|  |
|  |

Above: what the model pose setup should look like, top and front views.

Select the Hips joint and use QExportSolver to create the XML for QTM. Load it into QTM and solve it on the EliROM capture. To view the streaming result, import the QAvatar again and rename the prefixes to “Eli\_” for the streaming skeleton. You will now have two copies of the QAvatar in your Maya scene, one is for the model pose definition, the other is for viewing the streaming result.

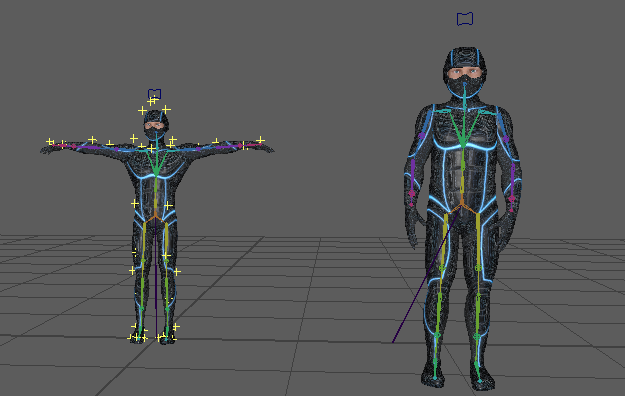


This is what it will look like after importing the new QAvatar and rotating it into place with a translation to separate them. Don’t worry about the scale difference, the new QAvatar will be scaled down by the data once it starts streaming. Now start streaming the data from QTM:



Another rotation adjustment of the reference node will be necessary to get the avatar upright. Note that the scales of the characters now match.

If you need for the QAvatar character to maintain its original scale, then enter a scale value of 1.163 (this inverse of 0.86) into the scale values of the Reference joint and turn off the “Segment Scale Compensate” flag for all the joints.



The final result on the QAvatar. The QAvatar is back at its original scale with its original skeleton structure. The data from the smaller performer is applied to it.

|  |  |
| --- | --- |
|  |  |

A comparison of the difference between using the built-in animation model pose (left) versus rebuilding the solver model pose from the original QAvatar (right). The difference in the shoulder construction is apparent, the structure differences of the hips and leg can also be seen.

## Solving to a Rigid Body Prop

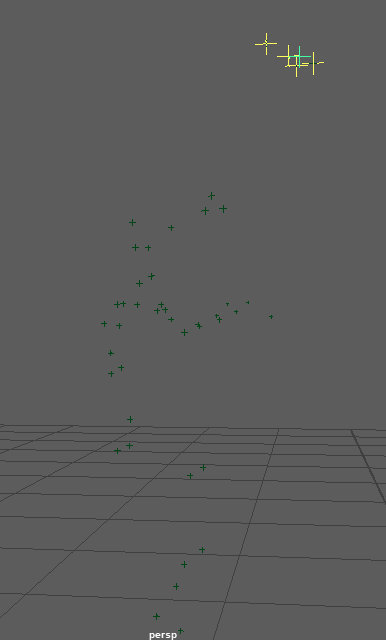
Will show how to use the solver for solving a simple prop.

Using the sample data “DomiTake1” from the SIGGRAPH2019 dataset. In this capture there is a “stick” prop object as part of the capture. This quickstart outline will show how to turn the processing of the stick into a single segment solver definition.

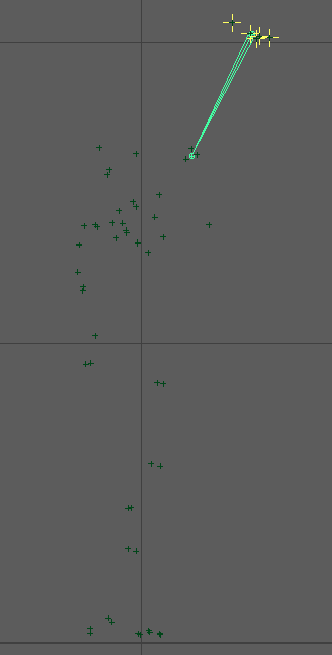
1. Change the names of the markers for use with Maya and a markerset. Rename the stick markers “STICK\_S1”, “STICK\_S2”, etc….
2. Delete all the other marker names
3. Generate a new AIM definition. Call it “STICK”.
4. Remove the 6DOF definition of the stick.
5. Reprocess the move with the new STICK AIM model.
6. Export the data to an FBX file, be sure to export only marker data.
7. Load the FBX into Maya. You will see the marker data from the Stick as well as from the performer and another prop.
8. The first two markers of the stick aren’t actually used for the stick. So we’ll only work with the last 5 markers. Find a frame that makes it easy to see how the stick is to be held, I choose frame 123.
9. Select the 5 markers and use the python script “WashLocators” to make duplicates of those 5 markers at that frame. The new locators will be larger and brighter so they’re easy to see.
10. Create a joint that extends from the middle of the newly created markers down to where the hand of the performer is (this will be two Maya joints).
    1. Name them “TorchHandle” and “TorchHandle\_end”
    2. Make sure the Joint Orient values are zero for both. You will have to rotate the TorchHandle joint back to the correct position after
    3. Make sure the translation offset for Torch Handle\_end is only on the X axis. I chose an offset of 50.
    4. You will have to rotate the TorchHandle joint back to the correct position after these changes.
11. Once the TorchHandle joint has been positioned correctly, delete all the old locators (select the “Opticals” node and delete it).
12. Select the new locators and group them under a “Markers” group node
13. Select the Markers group node and the TorchHandle joint, group them under a node called “STICK”.
14. Change the names of the locators to remove the “New\_STICK\_” prefix.
15. Add marker attachments by:
    1. Select the TorchHandle joint
    2. Add the individual locators to the selection.
    3. Invoke the AddAttachments script to ad the attributes to the TorchHandle joint.
16. Add the DOFs to the TorchHandle:
    1. Select the TorchHandle joint and invoke the script AddDOFAttributes.
    2. Turn on the translation DOFs for the TorchHandle joint.
17. Save your Maya scene.
18. Export the XML
    1. Select the TorchHandle joint
    2. Invoke ExportSolver.py
19. Load the XML into QTM and re-solve
    1. Open the Reprocess interface
    2. Choose to “Apply the current AIM models”
    3. Make sure the STICK AIM model is loaded
    4. Choose “Solve Skeleton”
    5. In the Skeleton Solver options
       1. Remove the “m” skeleton solver
       2. Load the “STICK” skeleton solver
    6. Optionally, remove the STICK 6DOF rigid body

|  |  |
| --- | --- |
|  |  |

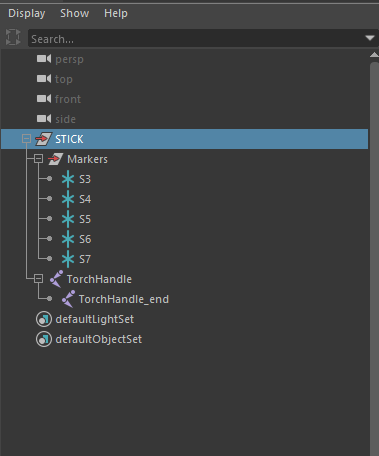
Before the label change (left) and after (right).



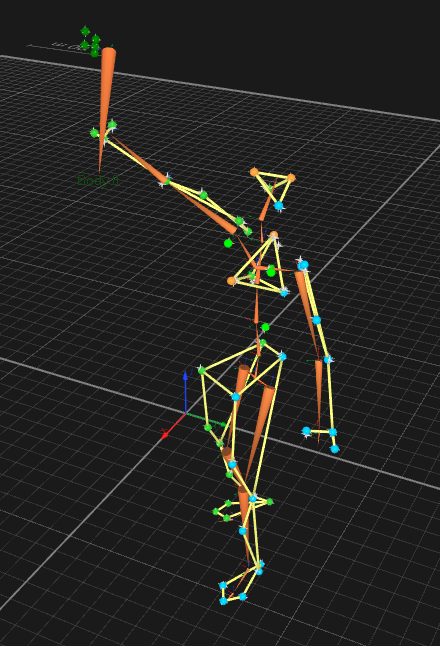
Above, after making the new locators by “Washing” the old ones.



Using the Front Y orthographic view in Maya, draw the joints as in this image.



The scene hierarchy after deleting old locators, grouping the new ones and grouping the markers with the joints.



After retracking and re-solving with the STICK (torch) solver model.

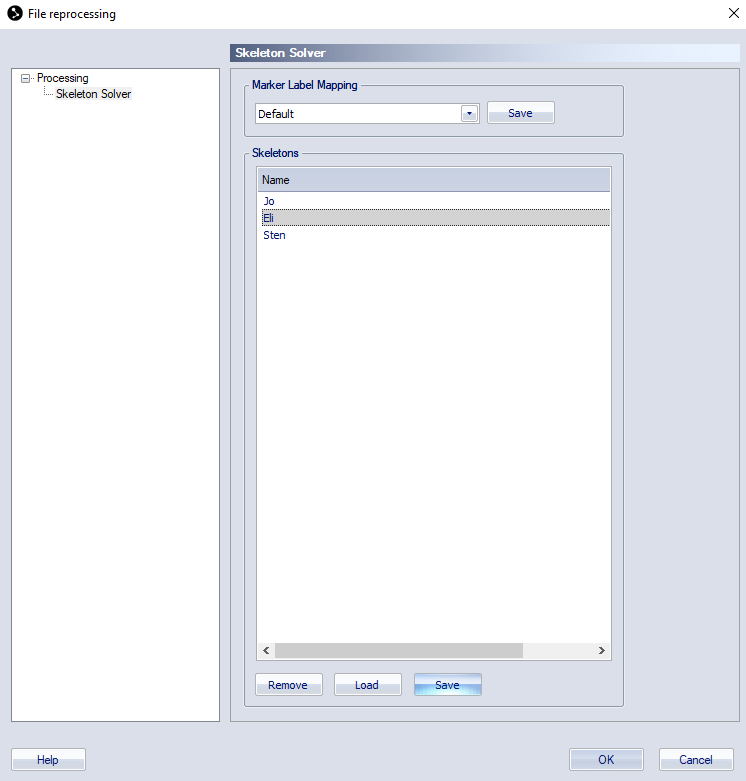
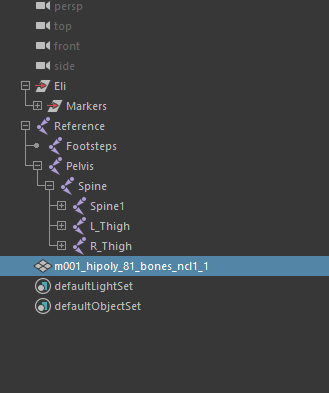
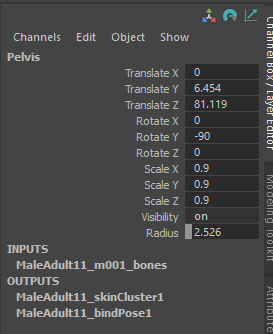
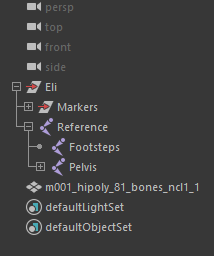
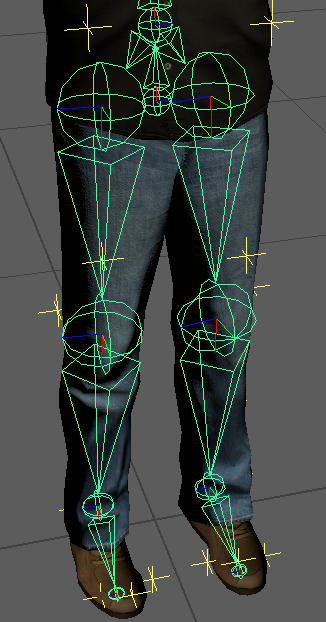
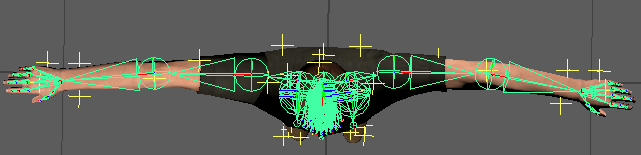
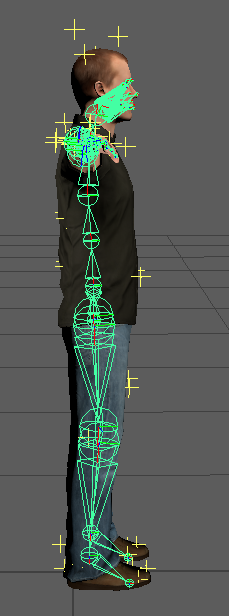
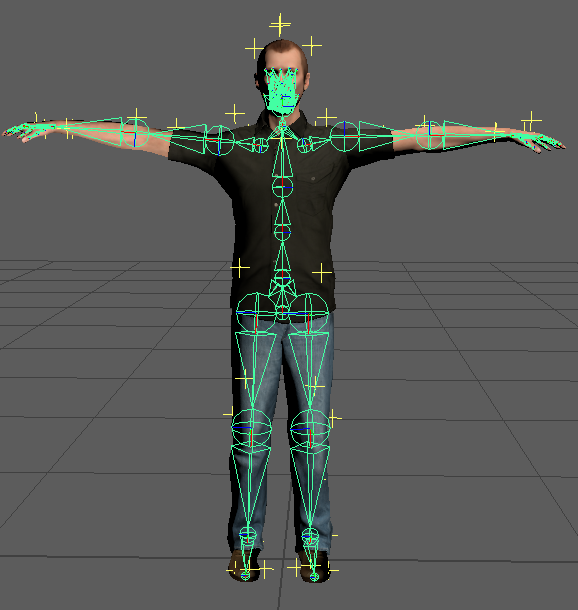
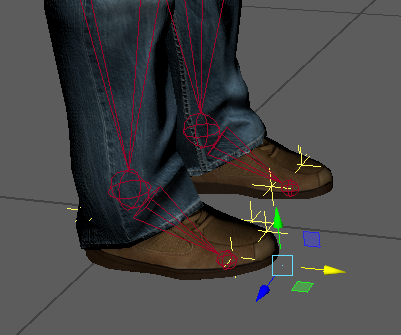
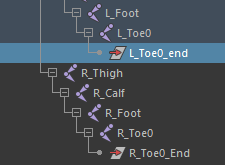
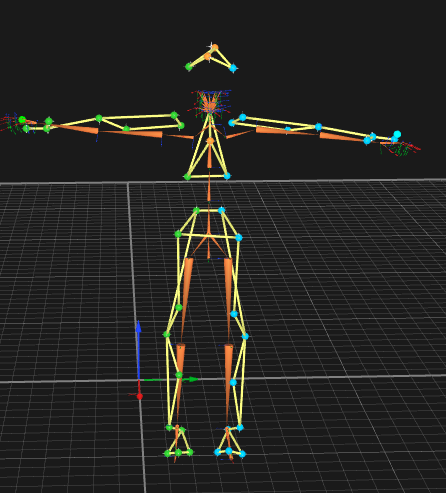
## Solving to a Complex Prop with Multiple Segments

In this case a bow shooting an arrow.

## Solving to a User Defined Avatar

Show how to make a model pose from scratch using an avatar created outside of Qualisys. As always, be sure to save your Maya file periodically while going through the guide.

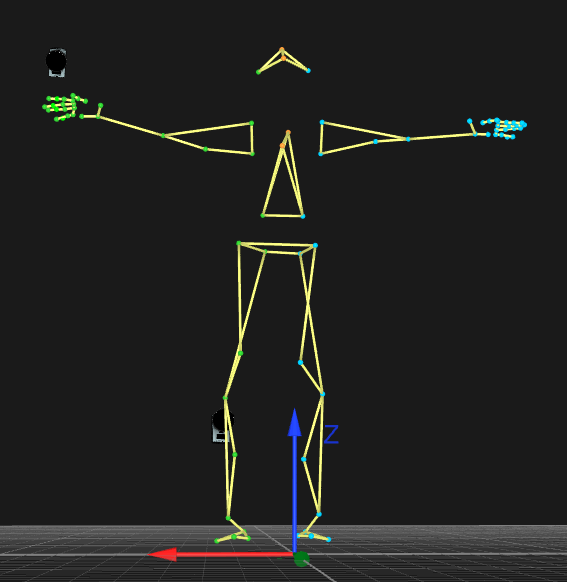
Using the QFL *Skeleton Solver Demo – Animation Marker Set.*  For obvious reasons the first step, Prepare the Model, must come first. But the next three 3 steps (Position the Avatar, Set the DOFS and Set the Marker Attachments) can go in any order and can be revisited as often as necessary to modify the settings.

1. Prepare the Model
   * Load the EliROM measurement
   * Export the Eli.XML file by using the “Save” option from the Reprocess->Skeleton Solver dialog:  
     
   * Load the XML file into Maya
   * Delete the skeleton by selecting “Hips” and hitting the delete button.
   * Import the Maya file of the Microsoft avatar “MaleAdult11.mb”  
     
   * Rename the elements to remove any namespace or prefix that has been added to the imported nodes:  
     
   * Remove any keys from the reference node and set all its values to zero so it is at the origin with no rotation.
   * Select the Pelvis and rotate, translate and scale the avatar to fit the markers. Here are the values for this character:  
     
   * Parent the joint hierarch to the “Eli” group node
     + Select the “Reference” node.
     + Control select the “Eli” node
     + Hit “P” (or use the menu command) to make the Reference node a child of Eli.  
       
2. Position the Model
   * Now rotate the joints into place so that the avatar matches the position of the markers:  
     
   * Select the Pelvis node and use Maya->Select->Select Hierarchy to select all of the joints.
   * Invoke the python scrip “AddDOFAttributes.py”
   * Edit the DOF attribute flags for all the joints:
     + Add all three translations DOFs to the Pelvis.
     + Turn off all DOFS for the fingers
   * Add an extra node to the toes (to give them length when viewed in QTM). The head and hands have many child nodes so there is no need to add an extra node, but if your avatar has no fingers then also had a child node).
     + Select the toe
     + Duplicate it
     + Make the new node a child of the toe
     + Rename the child node something like “Toe\_end”, the name doesn’t matter.
       - After reparenting it’s possible that a transform node was added. This is fine, just delete the new node and rename the transform node.
       - Offset the new node to give the parent some length. It’s best to zero out rotations and joint orient values.  
          
3. Set the Marker Attachments
   * For each segment with a DOF
     + Select the segment
     + Shift select in the 3D window (or control select in the outliner) the markers to be attached to the segment
     + Invoke the “AddAttachment.py” script to add the selected markers to the segment.
     + It’s good to practice doing this for the character so you know which attachments are going to which segments.
   * As an alternative to the above step, a MEL script called “AttachAnimToMSAvatar.mel” is provided to apply all the marker attachments. This can be used only with this combination of markers (the animation markerset) and skeleton (the MS Avatar). If you have a different set of markers or skeleton you will have to do the attachments manually the first time. It’s worth the time to copy the MEL commands for this into a script for the next time you set up a character with this skeleton/marker combination.
4. The model definition is complete. Export and solve the new definition.
   * Select the Pelvis node
   * Invoke the “QExportSolver.py” script. Select a file to export to.
   * Load the file in QTM
     + Select Reprocess
     + Turn off everything except “Solve Skeletons”
     + Select “Skeleton Solver” in the outliner
     + Select “Load”
     + Load the XML file
     + When you select “OK” the skeleton will be re-solved.  
       
5. Making Adjustments. The solve will be pretty good, but there are some trouble spots. The neck moves too much and the clavicles might not be right. Here are some adjustments to make in Maya to address these issues:
   * On the Neck joint, set the two head marker weights to a small value: 0.1.
   * On the Clavicles
     + turn off the X rotation DOF.
     + Set the marker weights to 5 each.
   * On the hands
     + Turn off the X rotation DOF.
   * On the Toes
     + Turn off the X and Y rotation DOFS (leaving only Z)
     + Set the weight of the toe marker to 5

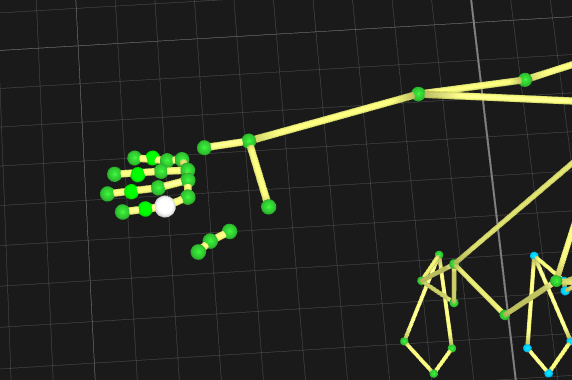
## Solving to a User Defined Avatar with Fingers

The construction of a solver model with fingers is no different than with a standard full body model without fingers: the joints of the fingers are driven by markers. The main differences in how the fingers are constructed is determined by the number of markers and where they are placed. It’s possible to use markersets with as many as 19 or 20 markers for the fingers and as few as three (the so called “Lobster Claw” has one marker for the thumb, one for the index finger and one for the pinky). The fewer the markers the more complicated the solver setup if the goal is to drive all the fingers with data. The easiest setup for finger solving is when there are the full compliment of 19 or 20 markers for the fingers. This is what is described here first. Later sections will describe how to leverage the solver to extract more motion from fewer markers.

The first step is to have a markerset with all the markers, like so:



Above: Front view.



Above: closeup of the hand markers.

This markerset is the animation markerset with 19 markers added for each hand: 4 on each finger, 3 on the thumb. As with previous examples, an AIM model with a markerset name prefix must be created for this markerset. In this case it is called “DTN”.

The setup process is just like in previous sections. Like with the Rigid Body Prop you will export the optical data from QTM into an FBX file to extract the T-Pose of the performer. Then the steps for setup are just like as with a User Defined Avatar.

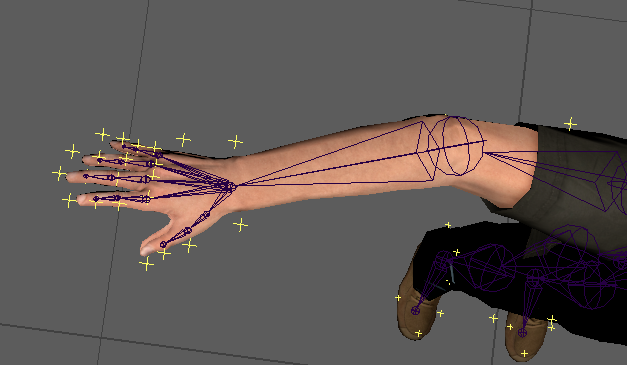
To recap:

1. Export the optical data from QTM into an FBX file.
2. Load your character into Maya.
3. Import the FBX optical data.
4. Find the frame of data for the T-Pose, select the markers and use the “WashLocators.mel” script.
5. Rename the markers, add it to the correct namespace (in this case it will be “DTN”)
6. Delete the other optical data.
7. Prepare the character skeleton as before:
   1. Add the “ModelPose” namespace
   2. Add the DOF attributes
   3. Fit the skeleton/character to the T-Pose
   4. Attach markers and set the weights.

At this point you have set up the body of the avatar for solving just like as in the User Defined Avatar section.



Model Pose front view.



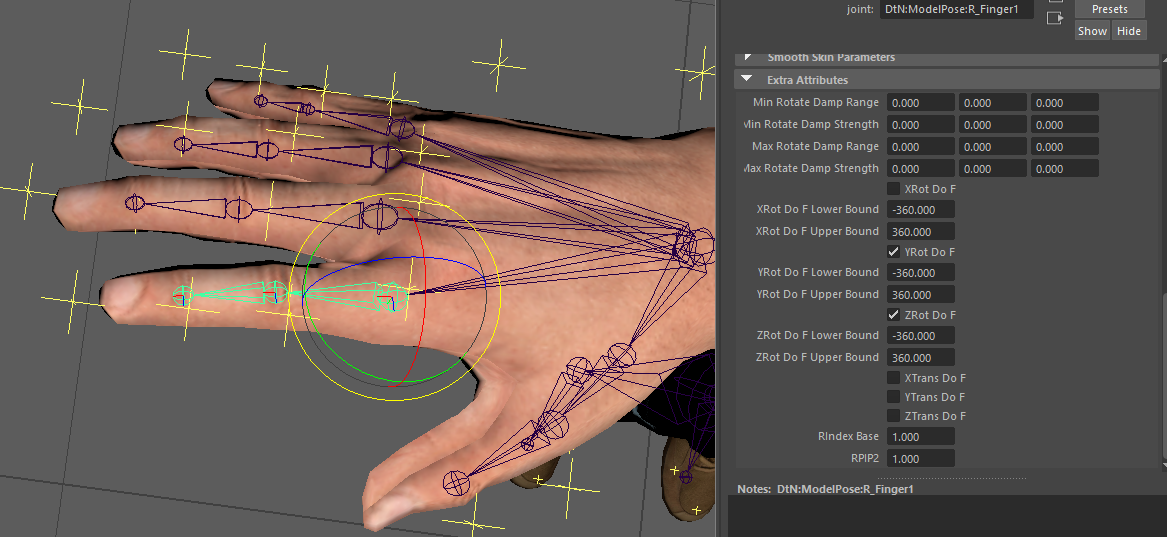
Closeup of the hand.

When aligning the skeleton for finger solving is very important that the fingers are well aligned with the finger markers. This is often hard because the overall height of the character may not match the overall length of the arms. In this example you can see that in the front view of the model pose the avatar is a bit short compared to the performer’s markers (there is a noticeable gap between the head and the topmost head marker) . The scale is left this way because if the avatar was scaled up then the arms would be too long. This is common in character design where virtual characters often have longer arms than is normal, even for characters that are humans.

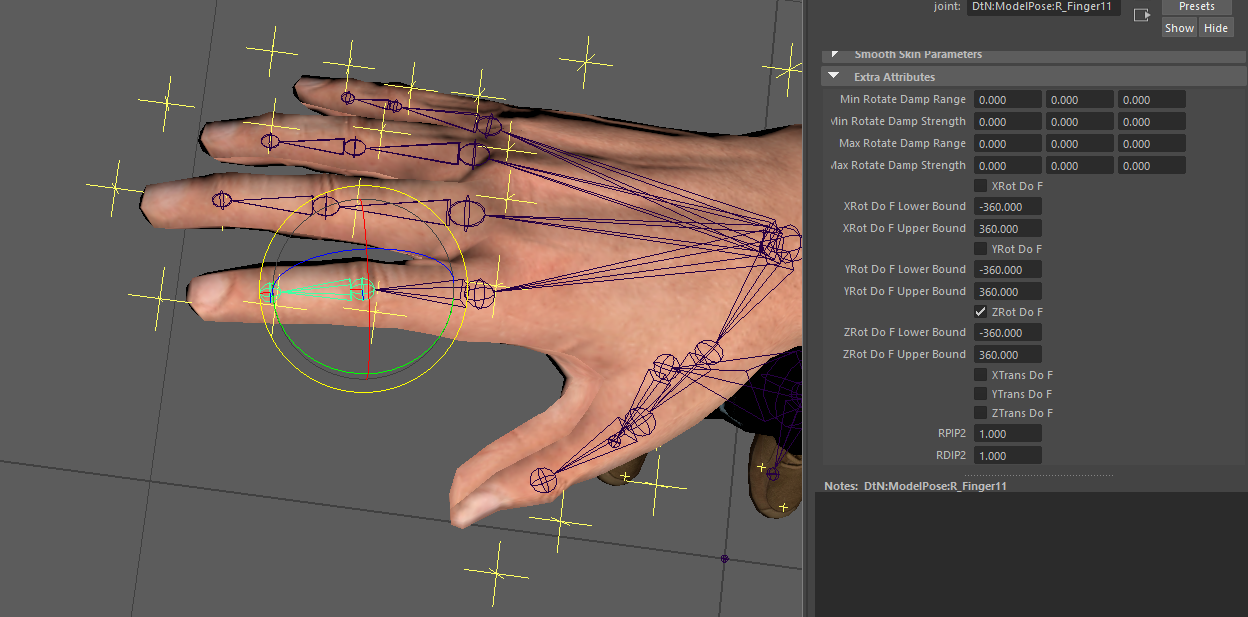
In the Closeup of the Hand image you can see that the elbow is flexed a bit more than normal to pull the hand back to a position where the finger markers match well.

The joints of the fingers are setup as follows:

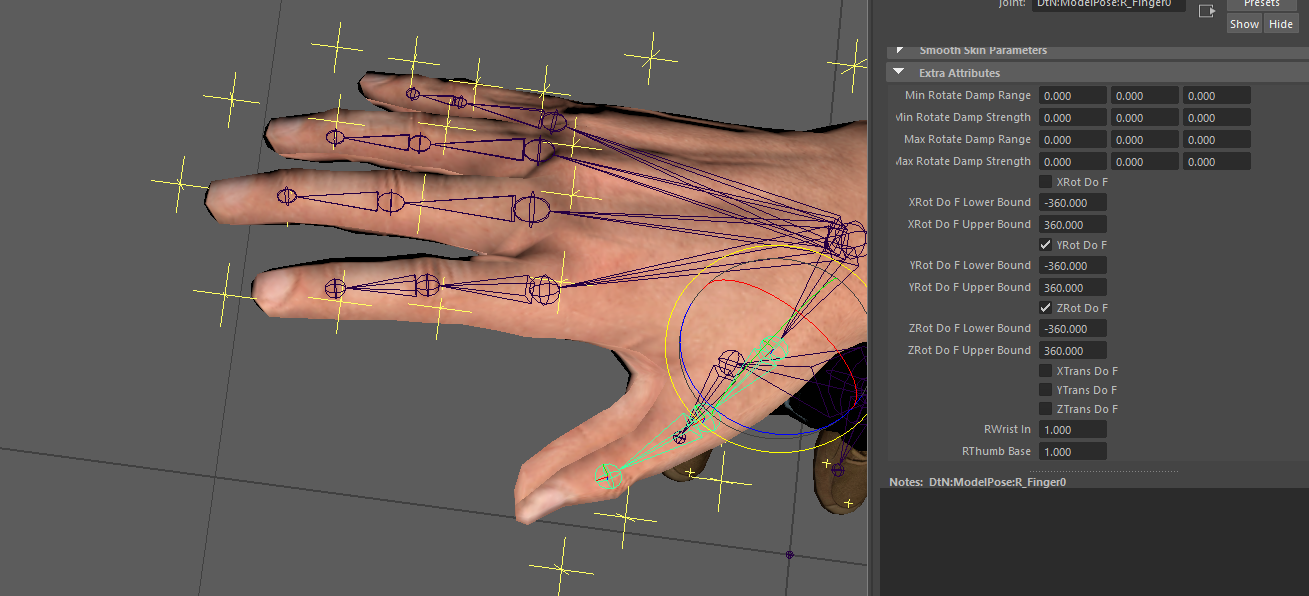
* Base joints have 2 degrees of freedom: up/down and left/right. No twist is allowed.
* Finger joints have 1 degree of freedom: up/down.
* The joints need to be aligned so that the hinge rotation is on one axis (X, Y, or Z)
* The thumb works the same way as the rest of the fingers



The base joint rotates on the Y and Z axes.



The finger joint rotates on only the Z axis.



Same goes for the thumb.

# Notes

## Maya Scene Construction

These scripts support a very specific Maya scene construction. The export script has been updated to allow for the presence of other scene elements, but there must be a model pose construction that obeys the following:

* A namespace must exist that matches the QTM markerset name. This must be a child of the root namespace
* A sub-namespace called “ModelPose” must be created.
* A group node called “Markers” must be in the namespace. This node contains all the markers as locators.
* All the locators are globally located. They’re still children of the Markers node but that node has no transforms. The global position is stored in each locator.
* The skeleton is in the ModelPose namespace. The name of the joints can be anything.
* Some characters have a reference node, some do not. The reference node is not driven by mocap data. It must not be used to scale/position the character for the model pose. All such adjustments must be made on the root of the character, usually the Hips or Pelvis joint. This is the joint that must be selected when exporting the solver information.
* There MUST be NO name clashes. With namespaces this shouldn’t generally be a problem.
* Information for the model pose is stored in the joint nodes in Maya:
  + Each marker attachment is defined as a floating point user attribute. The value is the marker weight. The existence of the attribute indicates an attachment.
  + Every joint is assumed to have all the DOF flags. These are Booleans that indicate the use of each DoF.
  + The Rotational DoFs have lower and upper bound attributes. These are floats expressed as degrees in Maya but get turned into radians for the XML file.
  + The Maya “Preferred Angle” attribute is used for the “Default Transform” in the XML file. In Maya the Preferred Angle is used by the IK system, but otherwise has no effect on the joint.
* Except for the root, the “Joint Orient” value MUST be set to zeros.
* Careful attention should be made to the orientation of joints.
* Careful attention should be made to the offsets of children, this is what determines the “length” or “direction” of segments.
* At leaf nodes an “\_end” joint must be added. This is required to specify the length of the segment (mostly for display purposes in QTM).
* Maya does not allow for names with periods or spaces in them. All marker names with periods or spaces need to be renamed. Using spaces for names is a bad policy in general.
* The root of the joints (such as the “Hips” or “Pelvis”) stores the global scale factor used to size the skeleton to the marker cloud. This value is stashed in the scale value in the XMl. Likewise, on import, the scale is stored in the joint root with all other joints having their offsets adjusted accordingly.

## The Scripts

The functionality is provided in a set of Python and MEL scripts for importing, editing and exporting the solver model pose information. Installing the scripts is easy:

* Copy the scripts to the default Maya scripts folder.
* Make shelf icons for the scripts QImportSolver, QExportSolver and AddAttachments (see below for more details)

### **QImportSolver.py**

Note! This script replaces the obsolete “ImportSolver.py”! Use QImportSolver.py instead. It updates the old file by making it a proper class and ready for use directly (as it is described here) but also for use by the qtm\_connect\_maya streaming plugins for Maya (for the “Pull” and “Push” solver definitions).

This script constructs a Maya scene from the given QTM XML solver file (or string). NOTE!! This script nukes the current Maya scene, so be sure to save your existing scene before using this script.

Put this in your Maya shelf icon:

import qtm\_connect\_maya.QImportSolver

reload(qtm\_connect\_maya.QImportSolver)

qtm\_connect\_maya.QImportSolver.ImportQTMSkeleton()

This code ensure that the most recent version of the script is loaded and executed. If you make changes to the script be sure to save it before executing through the icon.

The scene will be the solver model pose: a skeleton and a set of locators. The locators are colored yellow to make them more visible than the default dark green.

### **QExportSolver.py**

Note! This script replaces the obsolete “ExportSolver.py”! Use QExportSolver.py instead. It updates the old file by making it a proper class and ready for use directly (as it is described here) but also for use by the qtm\_connect\_maya streaming plugins for Maya (for the “Pull” and “Push” solver definitions).

This script exports the current scene to an XML file (or string). The scene construction must be very specifically made to support this script as defined in the Maya Scene Construction section. Put this in your Maya shelf icon:

import qtm\_connect\_maya.QExportSolver

reload(qtm\_connect\_maya.QIExportSolver)

qtm\_connect\_maya.QExportSolver.ExportQTMSkeleton()

This code ensure that the most recent version of the script is loaded and executed. If you make changes to the script be sure to save it before executing through the icon.

QExportSolver() requires that the user select the root joint to be exported. This allows for a parent joint to be used as a global locator for the character/skeleton. The parent joint should be left at the origin and the root of the character (such as Hips or Pelvis joints) be used to position the skeleton in the marker cloud (including using scale factors). A SanityCheck() function was added to make sure that the correct type of joint was selected and that its parent (or parents parent) matches the parent of the “Markers” node. Popup dialogs are displayed to guide the user through the correct setup for export.

### **AddAttachments.py**

This is a helper script that adds marker attachments to joint. Put this in your Maya shelf Icon:

import AddAttachments

reload(AddAttachments)

AddAttachments.AddAttachments()

To use:

1. Select the joint to have markers attached to. This MUST happen first.
2. Add the locators to your current selection (Ctrl select in the outliner, shift select in the 3D scene)
3. Run the script.
4. Double check the correct addition of the attributes by selecting just the joint and reviewing the user attributes for the joint (Ctrl-A to bring up the joint attribute interface).

Some MEL scripts are provided that show the scripted use of the AddAttachments() python function. Each of these scripts assume a specific skeleton and a specific markerset. They were made by doing the attachments by hand then copying (with some editing) the echoed MEL commands and encapsulating the commands in a script.

### **AddDoFAttributes.py**

This is a helper script for when you are creating your own joints from scratch. All joints must have these attributes for export. Simply select all the joints (note, you can’t just select a root node, you must explicitly select al the joints) and run this script.

### **WashLocators.py**

This helper script is for setting up the markers to be used in the model pose. It assumes that marker data has been imported into the Maya scene from an FBX file (likely one exported from QTM). When locators are created in this way they come in under a group node that likely has some rotation applied to it. Check that the rotation is correct. Also verify that the global scale is correct.

This process will bake out any of the rotation and scale values from the marker group node. It will also effectively get rid of all the keyframe data that isn’t used.

To use:

1. Move the time slider to the frame with the configuration of markers you want to be your model pose.
2. Select all the locators (not the group node, select all the locators)
3. Run the script. This creates a copy of all the markers with “New\_” prefixed to the names.

After this, the typical workflow is as follows:

1. Delete the original markers group node. This deletes all the marker animation data.
2. Select all the new markers and great a group node for them called “Markers”
3. Parent the Markers node to the scene root node (or make a scene root node if necessary)
4. Use Maya’s “Search and Replace Names” with the fields set to “New\_” and “\n”. This will remove the prefix from all the names.

## ColorLocators.mel

A helper Maya script for turning the locator color to bright yellow. This make it easier to see them. Select all the locators you wish to change and then run the script.

## AttachAnimToQAvatar.mel

A helper script for creating marker attachments to the skeleton. This specifically binds the Qualisys Animation markerset to the QAvatar skeleton.