

MT_RIGZ Rigging System Documentation

Author: Helge Mathee
Version: 3.0
Date: 16.06.2006

Installation

To install the add-on please refer to the XSI documentation. I recommend to install the add-on into a work-group, as you will have to create additional files later which you might want to share with other people in your environment.

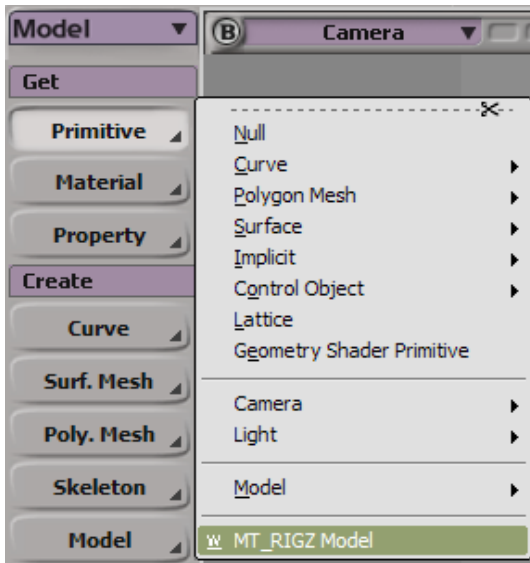
After the add-on has been installed I recommend to map the custom command "MT_RIGZ_Inspect" to a key in your key-map. Personally I use F4, but you can use any key you like.

Contents of the add-on

The MT_RIGZ add-on contains the following items:

- \Application\Views\RV_MT_RIGZ.xsivw
A relational view to store the PPG of the MT_RIGZ model.
- \Application\plugins\MT_RIGZ_plugin.js
The plugin file containing the implementation for the commands and the property.

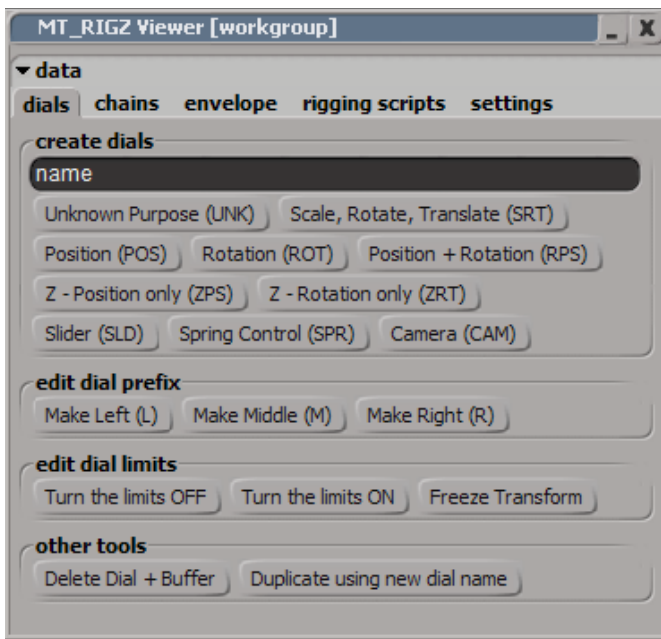
Creating a MT_RIGZ model.



A MT_RIGZ model is a normal model with some additional objects, such as groups and a global SRT dial. When I talk about dials I mean 3D objects which are used to control a rig. Most importantly though, a MT_RIGZ model contains a property called "data" which stores access to the functions of this rigging system.

To create a MT_RIGZ model choose *Get* -> *Primitive* -> *MT_RIGZ Model*. This will create a MT_RIGZ Model below the *Scene_Root*. You can open up the MT_RIGZ Viewer by selecting either the model or the controller shown in the center of the world and hit F4 (or what you mapped).

The MT_RIGZ Property



The MT_RIGZ Property contains functionality to create and edit the elements of a rig. There are so many different text fields, buttons and things you can do that I will go through all of them one by one. But before we do that, I would like to explain a little bit about how the system works internally so you get an idea why we are talking about a rigging system here.

How the system works from a programmer's point of view

The MT_RIGZ rigging system is a template based system. There is a pretty big list of templates for everything the system does. Moreover it is using a simple naming convention which is very important for all tools the system comes with.

The **naming convention** looks like this: *prefix_name_suffix*

So one example would be *m_global_SRT*. While you are free to choose the name (the middle part), the plugin contains a template for what prefixes are valid and what suffixes are valid, and, more importantly, what they stand for.

The base list of allowed prefixes is *l*, *m* and *r* for *left*, *middle* and *right*.
The list of allowed suffixes is a little bit longer, but here it comes:

UNK unknown / non-defined object
PAR parent object (just used to parent objects together)
BUF buffer object (used to neutralize a dial's transform)
CAM camera
LGT light
CNS constraint target
SLD slider dial (can only move between 0 and 1)
SRT scale, rotation and translation dial
SCL scale dial
ROT rotation dial
POS position dial
RPS rotation and position dial (no scaling)
ZPS z-position only dial
ZRT z-rotation only dial
ROO root of a chain
JNT joint of a chain
EFF effector of a chain
ENV envelope deformer
LBL label (such as text or frames)
PRX proxy object (non-render-geometry)
GEO geometry (to render)
CRV curves

The plugin defines a template with attributes for each of those suffixes, and the template includes wire-color, icon to use, icon-size, icon-offset, if the object has a buffer, the object's position- and rotation-limits as well as the object's group memberships.

Now, every time an object is created through the MT_RIGZ tools its attributes are applied to the object. The list is not complete, but can easily be extended. So you can add new suffixes or new attributes by editing the plugin file.

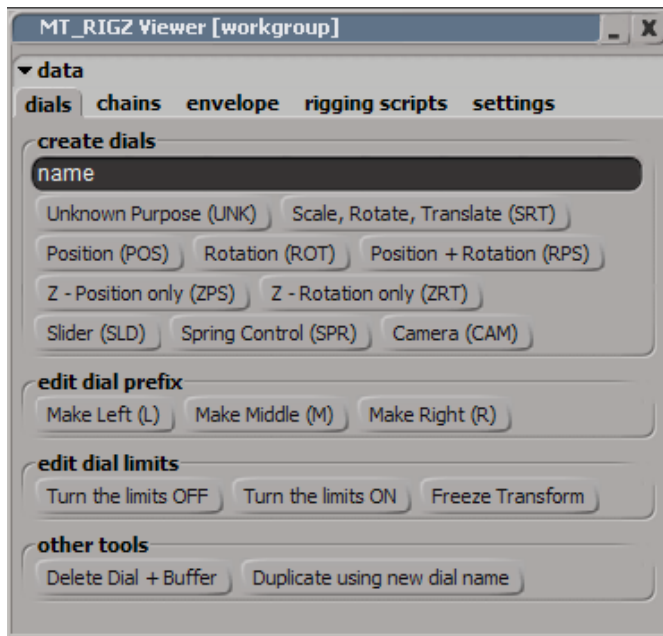
There is a template for what groups should be created when a MT_RIGZ model is created, and there are templates for other types of objects, but the most important thing of all is this:

Hierarchies created through the MT_RIGZ system are not only obeying the template rules, they also contain all of the information needed to recreate them only by their **names**. This means that it is simple to recreate a part of a rig once you created it through this rigging system. One example:

You create a facial dial setup, where you have objects for the eyes, eyebrows, mouth, nose, ears whatever. Even though this setup might be slightly different for the characters you create, the base is the same, as is the functionality in it. So this hierarchy, defined by all its objects, can become **a new template for the system**. And the next time you need to create a facial setup, you can just load up the template for the face. As this contains only template-based information for each dial and object in the hierarchy, if your default-wire-color for a rotation-dial changed since you last used the face-template, if you recreate the face-dials they will get the newly defined wire-colors, group memberships etc while maintaining per-instance attributes (such as size, color, limits etc). Those hierarchy templates (which are Jscript files really) store additional information such as Constraints and Expressions. Animation, Scripted Operators etc are not persisted, but the plugin is open-source so go ahead ;-)

So this is where we start using it now....

Creating dials



To create objects using the MT_RIGZ system just open up the property by hitting F4 (with an element or model selected) and switch to the **dials** tab.

In the first section, called **create dials**, there are buttons for all types of objects you can create.

To create one the workflow is as follows:

1. Select the parent of the new dial
2. Type in a name for the dial
3. Push one of the buttons to create the dial of your choice

In the second section on the **dials** tab, called **edit dial prefix**, you can edit the

prefix of the dial by just hitting a button. You need to select the dial(s) you want to change the prefix of, and then hit *Make Left*, *Make Middle* or *Make Right*. If you want to introduce more prefixes (such as top and bottom etc) you need to define them in the plugin and add buttons here.

The third section, called **edit dial limits** contains three buttons for the following functions:

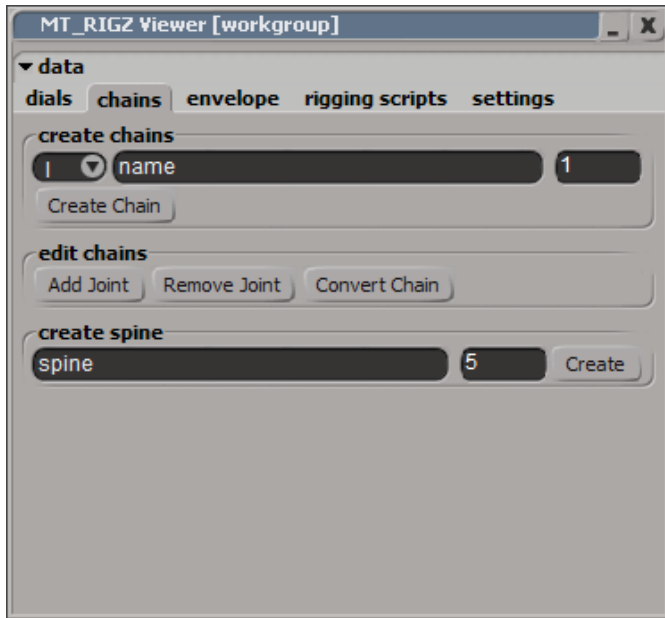
1. *Turn the limits OFF*: This button turns off position- as well as rotation-limits off of all selected dials.
2. *Turn the limits ON*: This button turns the limits back on, based on the template of the dial. So position-dials get rotation limits, etc...
3. *Freeze Transform*: This button does not use a neutral pose to freeze the transform of the object, it rather moves the buffer object below the selected dials to zero out the dial's transform.

The fourth and last section, called **other tools** contains two additional tools, of which the first one is a shortcut to delete a dial and its buffer, and the other one is discussed later.

Example: Open up a new scene, create a MT_RIGZ model and select the m_global_SRT dial. Hit F4 to open up the MT_RIGZ property. Switch to the **dials** tab, type "hip" into the name field and hit *SRT* to create a SRT dial. Move it up 5 units in Y and hit the *Freeze Transform* button. Now type "chest" into the name field and hit *SRT* again, move the chest up and zero out the transform again by hitting *Freeze Transform*. Now type in "head" into the name field, but this time hit *ROT* to create a rotation dial. As this dial is not movable, you need to turn off its limits to be able to move it around. So hit *Limits OFF* to do so, move the dial up to match the head position, hit *Freeze Transform* to zero out its transform, and finally hit *Limits ON* to get back the rotation-dial's limits. Keep an eye on the schematic view while you are doing all this!! Note that you can Reset the transform of all dials, and they will jump back correctly!!

Chains in the MT_RIGZ system

MT_RIGZ's chains are no real chains. They are something comparable to the guide chains coming with the biped guide in XSI. This is interesting because it enables you to design guides for parts of rigs. Using the MT_RIGZ property, you can convert chains from real chains to guides and vice versa, but only if they have been created through the system as well (as the tools rely on naming conventions). So let's have a look at the *chains* tab on the property.



The *chains* tab contains tools for chains and spines, but let's look at chains first.

In the first section, called **create chains**, you can create chains by specifying a prefix for the chain, the name for the chain and the number of joints you will need for the chain. The workflow there is as follows:

1. Choose the prefix
2. Type in the name for the chain
3. Type in the number of joints
4. Select the parent for the chain
5. Hit the *Create Chain* button

In the second section, called **edit chains** you can edit existing chains by adding joints or converting between guides and

skeleton-chains. To add a joint to an existing guide chain:

1. Select an object of the chain (root, effector or one joint)
2. Hit the *Add Joint* button.

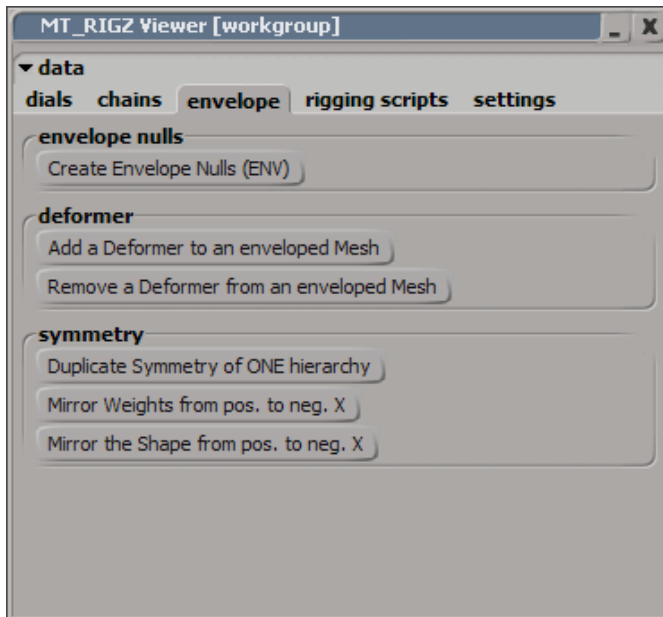
To convert the chain from a guide to a skeleton chain or vice versa:

1. Select the root of the chain
2. Hit the *Convert Chain* button.

In the third section of the **chains** tab you can create spines. The spines created through this section are the same as XSI's build-in ones, so nothing really new here. What the command does though, is it renames all parts and uses the suffixes accordingly. To create a spine:

1. Select the hip dial (the bottom controller for the spine)
2. Select the chest dial (the top controller for the spine)
3. With these two controllers selected:
4. Enter a name for the spine
5. Enter the number of vertebrae to create and
6. Hit the *Create* button.

Envelopes in the MT_RIGZ system



There is no replacement for the envelope here, just a couple of additional tools to simplify some of the standard tasks related to enveloping. Alright then...

In the first section of the **envelope** tab, called **envelope nulls**, there is just one button, to create nulls below every selected dial resp. chain. To use it:

1. Select all objects you want to create envelope nulls for
2. Hit *Create Envelope Nulls*

In the second section, called **deformer**, you can add and remove deformers to / from an existing enveloped object. To add a deformer to an enveloped object:

1. Select the mesh
2. Select the new deformer
3. Hit the *Add a Deformer to an enveloped Mesh* Button

To remove a deformer from an envelope:

1. Select the mesh
2. Select the deformer
3. Hit the *Remove a Deformer from an enveloped Mesh* Button.

The third section, called **symmetry** contains three functions to deal with hierarchy, envelope and shape symmetry. I agree that hierarchy and shape symmetry doesn't essentially belong on the envelope tab, but it is there now, go ahead and move the buttons around if you don't like them ;-). The *Duplicate Symmetry of ONE hierarchy* button uses XSI's Duplicate Symmetry command and then afterwards fixes the names of all created objects and flips their prefixes, so left becomes right etc.... To use it:

1. Select the hierarchy to duplicate in branch (make sure it is not prefixed with **m**)
2. Hit the *Duplicate Symmetry of ONE hierarchy* button.

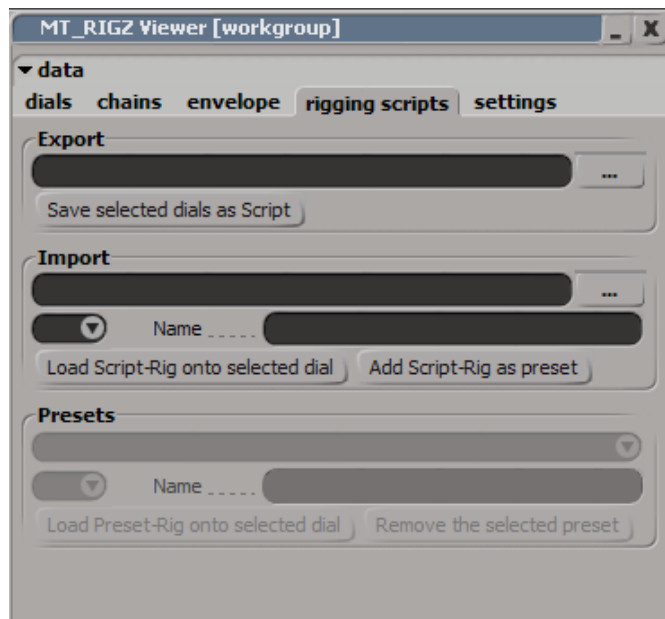
To mirror the weights of an envelope from the positive X side to the negative X side, the object has to have a symmetry map with the same direction. To mirror the envelope:

1. Select the mesh to mirror the envelope on
2. Hit the *Mirror Weights from pos. to neg. X* button

When dealing with Shapes as actual geometry (especially when using Select Shape Key and the shapes have been modeled in a different application), the last button is extremely useful. It mirrors the shape from one side to the other one, from positive X to negative X while maintaining the point-order correctly. To use this command you have to have a symmetry map and:

1. Select the mesh to mirror
2. Hit the *Mirror the Shape from pos. to neg. X* button

Rigging scripts and templates in MT_RIGZ



If you created a hierarchy using this rigging system, you can save it as a rigging script. To do so, switch to the **rigging scripts** tab and do the following:

1. Select the dials which make up the roots of your hierarchy
2. Either: Browse for filename in the *Export* section or leave the filename empty. Please use JS as the extension for the file.
3. Hit the *Save selected dials as script* button.

After you saved out such a script, you can load it back, by using the functions on the **rigging scripts** tab. When you load and execute these rigging scripts, you can

change the prefix the resulting elements will use and you can also append another name to the original name of all elements in the rigging scripts. This allows to change a prefix from m to l, so for example when you rig an eye you can rig it using the m prefix, export the rig as a new rigging script, and when loading it back change it to l resp. r. Adding another name allows to specialize the imported rig, so for example when you created a rigging script for a camera rig, you can add a "hero" or "top" to every name of every object in the camera rig when importing it. These scripts have a name remapping functionality build in, so expressions and constraints are re-targeted automatically if a name changes. To import a rigging script:

1. Browse for the file or leave the filename empty
2. choose a new prefix or leave the prefix field empty
3. choose a name to append or leave the name field empty
4. Select the parent of the new rig to be imported
5. Hit the *Load Script-Rig onto selected dial* button

When you use those rigging scripts a lot, it is quite handy to have the preset functionality in the bottom of the **rigging scripts** tab. To add a rigging script as a preset:

1. Browse for the file in the *Import* section of the **rigging scripts** tab
2. Hit the *Add Script-Rig as preset* button

You can now execute those rigging scripts in the *presets* section. You have the same prefix and name options, and you can also remove presets again. To use a preset:

1. Choose the preset to use
2. Choose a new prefix or leave it empty
3. Choose a name to append or leave the name field empty
4. Select the new parent for the rig to be imported
5. Hit the *Load Preset-Rig onto selected dial* button

The Spring in MT_RIGZ

There is a spring controller hidden in the system. To create it, do the following:

1. Select the parent for the spring
2. Hit the *Spring (SPR)* button on the **dials** tab.

The spring is a radial spring, not a positional one. Moreover, the simulation of the spring is bound to start at the frame 1, so you will need to change the frame at least to 2 to see thing move. Moreover, the spring "looks" along X to figure out how to move, so I think the easiest way to understand how the spring works is to show an example:

Open up a new scene, create a MT_RIGZ model and select the m_global_SRT dial. Hit F4 to open up the MT_RIGZ viewer. Switch to the **dials** tab, enter "Lilly" in the name field and hit the *Spring (SPR)* button to create the first spring. Open up a schematic to check what has been created. Note that the name of the spring is not Lilly but LillyA. This is done to simplify the creation of multiple springs in a row. To be able to see the spring move, create another SRT dial called "feedbackA" below the spring and move m_feedbackA_SRT +5 units along local X. It is very important to understand that the spring will only work along X, if you want to use a different direction you need to rotate the PAR of the spring, called m_LillyA_PAR in this example.

The operator driving the spring is stored on the buffer, called m_LillyA_BUF in this example. You can inspect it by looking at the local kinematics of this buffer. But for now, let's keep the settings as they are.

To test out the spring, select the m_global_SRT and switch to the second frame. You can now move the dial around to see the spring react.

You can use multiple springs in a row to create tails and similar "spring-chains". Furthermore you can use the rotation controllers, such as m_LillyA_ROT in this example, to pose the spring. The rotation controllers provide an extra layer of animation.

That's it. Note that this plugin / add-on is an example how to use template-based rigging in XSI. This plugin is neither finished nor perfect, and that's the reason it is open-source. You can modify the plugin, add or remove functionality to make it fit your rigging needs.

Regards,

Helge Mathee