Chrono::R3D

Reference manual

Contents

1 How to use the reference manual

In this manual you find the detailed description of all the properties of Chrono::R3D objects , as you can access them by selecting Chrono::R3D specific objects in Realsoft3D 'select window' and using the pop-up menu 'Properties..' to open the 'property window'.

Each Chrono::R3D object (rigid bodies, links, markers, etc.) have specific properties: these can be accessed in the 'Spec' (specific) tab of Realsoft3D **property window**. Of course, also basic properties like name, color, wireframe invisible, etc. can be still accessed in other tabs.

By means of the property window, you can control the power of the most advanced features of Chrono::R3D , so be prepared to use the property window a lot. (However, we suggest to close or minimize the property window when playing animation preview: in fact an open property window may slow down the playback rate).

As for all objects available in select window (splines, meshes, etc.) you can open the property window for multiple-selected objects: properties are shown with red fonts if they are not the same for all objects, and typing new properties will affect all selected objects at once.

Most Chrono::R3D objects are created by using ad-hoc tools, for example the Link Tool, whose buttons (and icons) can be added to the default toolbar of your environment. In the chapter about installation, you have already learned how to add or arrange these tools on the toolbar. In the tools chapter you find the detailed description of the behavior of these tools.

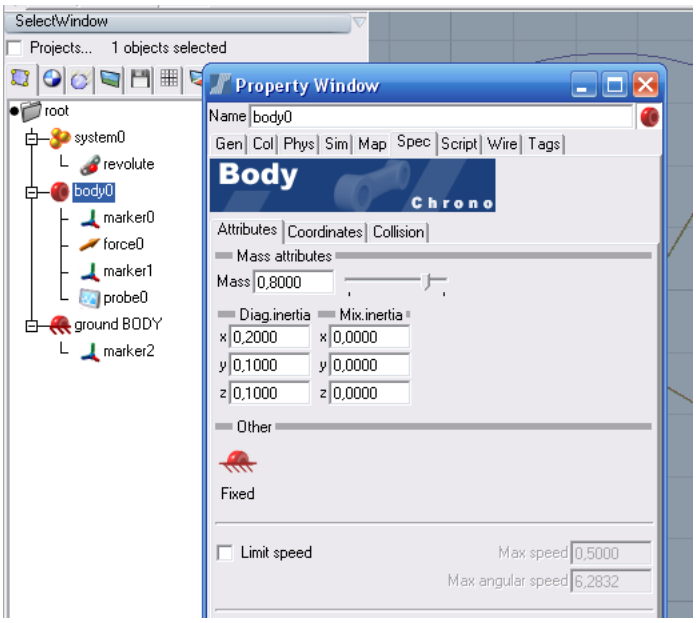


Figure 1.1 Property window for a selected object.

2 Objects

2.1 Rigid body

Rigid bodies are the most important elements of Chrono::R3D , since they represent the building blocks' of all mechanisms. In other words, they define the parts of the mechanical system which can move in space, each as rigid block.

Each rigid body can be constrained to other rigid body by means of link objects: these are described in a separate chapter.

Each rigid body can contain marker objects: these are coordinate systems which can be used as references for building links (constraints).

Also, rigid bodies can contain force objects (these are used to impose forces to the moving body) or probe objects (used to record variables).

Each rigid body object is, indeed, a level' object, and can contain whatever kind of 3D object (like spheres, nurbs surfaces, etc.), so you can model a part of the mechanism with many details -think about bolts, surfaces, etc.- which are contained in the body level, given that all these details will represent a rigid part of the complete mechanism. For example, a simple car' model can be made of five rigid bodies, four wheels and chassis, where the chassis level contains many details like glasses, lights, seats, doors, etc.

In fact, you may put 3D objects (surfaces, shapes, etc.) in a rigid body level for many reasons, for example:

- **snapping**: you may find useful to put axes, planes, input planes and similar references in rigid bodies, so these can be used for snapping when building other features (ex. links) in 3D modeler.
- **aesthetics**: shapes will be rendered in animation previews, renderings, illustrations, etc. and will give a good idea of mechanism motion.
- **mass, COG and inertia values** can be computed from a geometric representation of the part shape, using a special feature.
- **collision detection**: shapes contained in rigid body level will be used for collision detection and contacts, if multibody collision detection is turned on.

Note: you may think of building the car chassis with many separate rigid bodies which are then glued together with many constraints of lock type, but this would unnecessarily increase bodies in the system, so the simulation becomes slower to compute. In general, the rule is: use the least amount possible of rigid bodies, for a fast simulation! If two bodies must move together because glued, it is better that you make a single body -unless you need to recover reaction force in the gluing constraint-.

Rigid bodies can be **saved on disk, copied and pasted** just like all other 3D objects of Realsoft3D. Just remember, however, that duplicating groups of bodies at once may not have the effect you expect because the links between them can be copied only in certain

circumstances. Also, if you copy to disk a mechanism, remember to save also the link objects, not only the body objects.

Rigid bodies can be **moved and rotated** in 3D simply using the move and rotate tools of Realsoft3D, or using the coordinate handles and mouse. Note that you cannot extend' or scale a rigid body object: if you need this, you must select its inner parts and perform the scale or stretch operation on the contained geometric objects. After you moved an object, you may have broken some link. Anyway, as soon as the system is assembled for kinematics or dynamical simulation, the body will be moved in the correct position. You can move the body also by using precise numerical fields in the property window.

The **coordinate system** of the rigid body object, shown as colored handles in 3D view, represent also the center of mass and bodys XYZ reference system. If you want to move the center of mass respect to the body after you added many details in its level, you just use the tool 'move' of Realsoft3D with the 'handle' option (this will just move the handles, not the rest of the object). The same applies for coordinate axis rotation (use the 'rotate' tool and 'handle' option).

A rigid body object **should not contain** another rigid body in object tree hierarchy. If this happens, the mass of the contained body is NOT added to the container body, nor the contained will be constrained to the container. They will behave as separate, unrelated bodies.

Multiple rigid bodies **can be contained in levels**. This helps you to organize a complex mechanical system into sub-mechanisms (each contained in separate levels), where sub mechanisms are made of bodies. For example a train with three vehicles can be a level containing three sublevels: the first for the leader vehicle, the second for the middle one, etc.

Bodies can represent **'grounded' parts** (parts which do not move respect to the world reference, like trusses, basements, walls, roads, etc.) . To do so, use the 'Object locked to ground' flag in property window. Note: you can have multiple ground' objects in your system.

The properties of this object can be accessed and modified through the Javascript: see the documentation on body javascript object .

2.1.1 Body property window

Mass
Total mass of rigid body $[kg]$. Default is 1, but there is an option in Body creation tool which computes the mass of the body from its geometry and density. Later, you can use the button 'Recompute mass, inertia, COG' below, to recompute mass from given geometry and density. Note: this mass value has nothing to do with the 'mass' value of default Realsoft3D simulation system (see property window/Phys).

Diag inertia
Values of XX inertia $[kgm^2]$ for body rotation. Computed about bodys x,y,z axes. They represent the diagonal values $I_{xx}I_{yy}I_{zz}$ of the typical I_{ij} 3x3 inertia tensor. These values can be automatically computed by using the 'Recompute mass..' button below, given body density and geometry (otherwise a default value is used).

$$I_{ij} = \begin{Bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{Bmatrix} \tag{2.1}$$

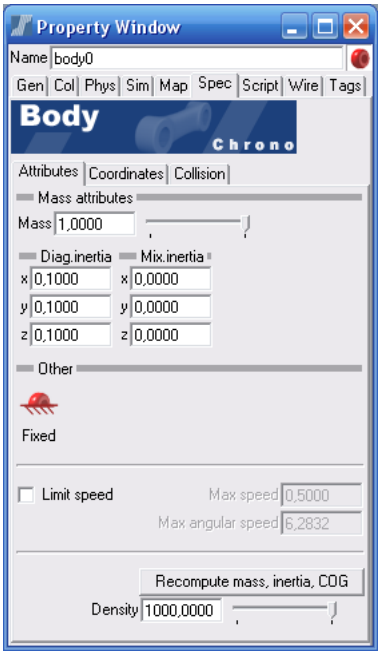


Figure 2.1 Body specific properties.

Mix inertia

Values of extra-diagonal inertia $[kgm2]$, about bodys x,y,z axes. They represent the extra-diagonal values $I_{xy}I_{xz}I_{yz}$ of the typical I_{ij} 3x3 inertia tensor (note that other tree values are $I_{xy} = I_{yx}$, $I_{zx} = I_{xz}$, $I_{zy} = I_{yz}$) These values can be automatically computed by using the 'Recompute mass...' button below, given body density and geometry. Note that these values are null if bodys axes are aligned exactly as main inertia axes (this depends on the shape of body).

Fixed

By default, this is not checked and body must be kept in position by forces, springs, links, etc (of course, otherwise it falls down!). Otherwise, if checked, this body is not affected by forces and constraints: it is grounded to the world reference. In fact if you use a Link tool to constrain a free body to the absolute reference, a ground body (with Locked on) is created as a reference. Note that you can have multiple grounded objects in a single system.

Limit speed

You can set this option ON if you want that this body never surpasses a maximum linear speed or a maximum angular speed. This feature can be used to improve the stability of the solver without the need of using smaller time steps, when handling a large amount of objects which might move too fast (hence causing, for example, missed collisions etc.) This approach is often used in animations for realtime applications like videogames; however remember that this affects the precision of the simulation and should be avoided in case of engineering simulations, where you should simply use a smaller timestep to improve the stability of the solver.

Recompute mass, inertia, COG

Use this button to compute automatically the mass of the body, depending on its shape (and density, see 'Density' field below). Also, computes the entire inertia tensor -the 'Diag

inertia' and 'Mix inertia' fields- and moves the COG (Center of Gravity) in the correct position.

In fact it often happens that you do not know the mass of a mechanical part, and it is even more difficult to know the values of inertia tensor. Therefore, you can use this 'Recompute...' button to let Chrono::R3D compute mass, COG position and inertia, simply using the geometric shape of the rigid body and its density.

Some notes about the 'Recompute mass, inertia, COG' button:

- *the rigid body level should include some geometric objects (ex. cubes, cylinders), otherwise an error will occur (zero mass is not admitted!).*
- *the geometry of the rigid body can be made of compound shapes, and freeform surfaces or SDS shapes can be used as well, but all surfaces must be closed otherwise resulting mass and inertia will be wrong.*
- *this function uses a low-order quadrature method (the space is volume-sampled to compute mass, inertia, etc.). This sampling has a finite precision, so maybe that the results are not 100% exact, you can expect some approximation if body shape has many thin details.*
- *the volume sampling of this feature can take some time for computing results, even some seconds with slow computers and complex shapes.*
- *center of mass (COG) is placed where needed, but axis are not aligned to main inertia axes (that is, body handles are translated to COG, but not rotated).*
- *rather than adding many details (ex. screw, holes) to bodys shape, it is more straightforward and time-saving to create a rough approximation of the full shape with few 3d objects (cubes, spheres): the inaccuracy can be negligible.*

Density

In order to recompute mass etc., the 'Recompute...' feature above needs the density of the rigid body: hence in this field you can set the density. CHtip Note that the density is expressed in $[kg/m^3]$ (assuming that you are using the default measuring system, 1 Realsoft length unit = 1 meter), so this value can be quite high. Some examples: density of water: $1000[kg/m^3]$, density of wood: $500-1000[kg/m^3]$, density of steel: $7000[kg/m^3]$, density of light alloys: $3000-5000[kg/m^3]$.

COG position

Position of body reference respect to the absolute (world) reference. The reference for the body is the coordinate system shown as colored handles in 3D view, also used to express the center of mass (COG).

Alignment

Alignment (rotation) of body reference respect to the absolute (world) reference. Rotation can be expressed as quaternions, Cardano angles, Eulero angles, Rodriguez parameters: just use the alignment angle-set selector.

Speed -linear-

Speed of body reference respect to the absolute (world) reference. You can also enter new x,y,z values in this dialog, in case you want to impose specific speeds for the starting condition of a dynamical simulation (if the speed you enter is not compatible with constraints, it will be corrected as soon as the mechanism starts simulation or gets assembled).

Speed -angular-

Angular speed of body reference respect to the absolute (world) reference, expressed in body coordinate system. You can also enter new x,y,z values in this dialog, in case you want to impose specific angular speeds for the starting condition of a dynamical simulation

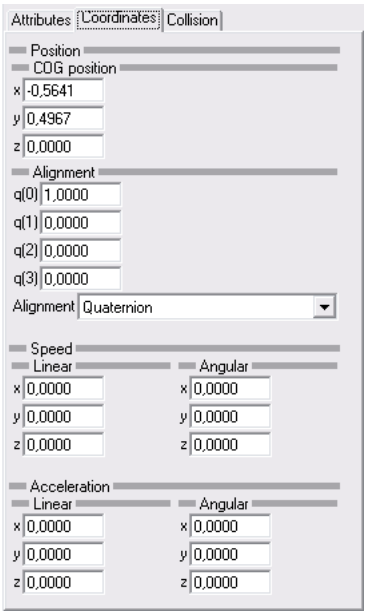


Figure 2.2 Body specific coordinates

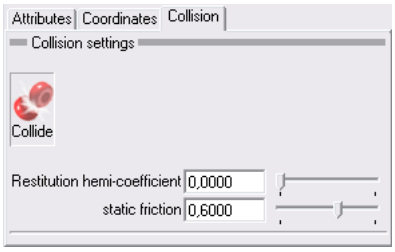


Figure 2.3 Collision settings for rigid bodies.

(if the speed you enter is not compatible with constraints, it will be corrected as soon as the mechanism starts simulation or gets assembled).

Acceleration -linear-
Acceleration of body reference respect to the absolute (world) reference. This is a read-only value.

Acceleration -angular-
Angular acceleration of body reference respect to the absolute (world) reference, expressed in body coordinate system. This is a read-only value.

Collide
If not checked (default) the body does not collide with other parts. If checked, the body collides with other parts which have this flag turned on.

Notes:

- *Colliding objects should not intersect when simulation starts.*
- *Ill-posed situations like 'shaft-in-hole with little clearance' can be troublesome to simulate with collision detection (the mechanism may explode' while simulating.*

- *Some typical situations which may cause problems in collisions: forcing a shape into a hole with little tolerance, contact between trimmed Nurbs, contact between shapes with very small details like grillages, collision between shapes which travel too fast for the simulation step.*
- *This Chrono::R3D collision has nothing to do with the default collision detection of Realsoft3D: do not use the default Realsoft collision detection while performing Realsoft collisions. The collision engine of Chrono::R3D is faster and more precise, and supports precise sliding contacts, static contacts, etc.*
- *The collision engine is optimized for the following types of primitive objects: spheres, ellipsoids, cubes, cylinders. Other types of shapes will be invisibly converted to polygonal meshes, but the precision and speed of the collision algorithm will be lower, unless these meshes are 'fixed' (not moving).*
- *If you want that some object contained in the rigid body level wont be considered for collision/contacts, select it and add an integer tag called 'noc' (as 'NO Collision') with value 1. Use the 'Tags' tab in property window.*
- *Performance trick: approximate the shape of the rigid body with few simplified collision shapes (cubes/spheres/ellipsoids) and set them RT-invisible and WF-invisible. On the other side, set the tag 'noc=1' for the detailed shape of the body, to be rendered.*

Restitution hemi-coefficient

Restitution coefficient, representing how much the body bounces' if affected by impacts orthogonal to the surface. The lower this value, the less elastic the collision (zero means totally dissipative plastic impact). Upper value is 1, for totally elastic impacts. Note: for bodies with different coefficients, the average of the two is used, hence the term 'hemi-...'.

Static friction

When this body collides with another body and a continuous motionless contact happens, a Coloumb tangential static friction is enforced. Note that this means that a tangential constraint will be introduced, and will be removed as soon as the tangential reaction force surpasses the Coloumb limit: $F < f_s \cdot N$ Note: for bodies with different coefficients, the average of the two is used. In the current release of Chrono::R3D , this coefficient is used also for the dynamic (grazing, kinematic) friction.

2.2 Marker

Markers are **coordinate systems** which are used by Chrono::R3D as references for defining links (constraints between rigid bodies).

Markers are **contained in rigid bodies**.

A marker **can be contained** in a sub-level of a CHlinkbodyrigid body.

A marker is a level: **it can contain other objects**. However you should not put a marker inside another marker.

Markers **can be saved on disk, copied and pasted** just like all other 3D objects of Realsoft3D .

Deleting a marker will **invalidate the links** which were using it as a reference (they will get the state of 'not active').

Markers can be **moved and rotated** in 3D simply using the move and rotate tools of Realsoft3D, or using the coordinate handles and mouse. Note that you cannot extend' or scale a marker object: the length of their axes is always kept constant.

Markers are fixed to their rigid bodies by default, but -if needed- **you can impose the body-relative motion** of markers by using ChFunctions (see the motion' panel).

Alternatively, you can use the standard **keyframer** of Realsoft3D in order to impose marker motion respect to body (use the red button record' in animation bar, move the marker at different times, and a Realsoft3D choreography with interpolation between keyframe positions will be created). However, in this case speed and accelerations will be computed by approximated numerical differentiation. Also, ChFunctions of motion' panel wont be used anymore.

The properties of this object can be accessed and modified through the Javascript: see the documentation on marker javascript object .

2.2.1 Marker property window

Absolute position
Position of marker respect to the absolute (world) reference.

Absolute alignment
Rotation of marker respect to the absolute (world) reference. The rotation can be expressed as quaternions, Cardano angles, Eulero angles, Rodriguez parameters.

Absolute speed -linear-
Speed of marker reference respect to the absolute (world) reference. This is a read-only value: if you want to impose markers speed, you must use the ChFunctions in motion tab!

Absolute speed -angular-
Angular speed of marker reference respect to the absolute (world) reference, expressed in marker coordinate system. This is a read-only value: if you want to impose markers speed, you must use the ChFunctions in motion tab.

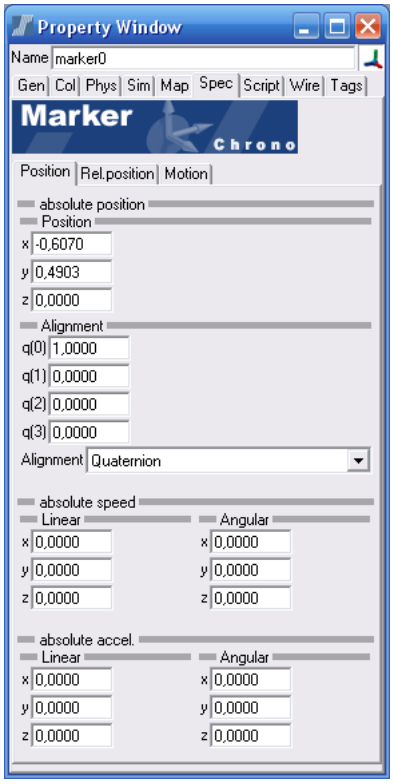


Figure 2.4 Marker specific properties: absolute coordinates

Absolute acceleration -linear-
Acceleration of marker reference respect to the absolute (world) reference. This is a read-only value: if you want to impose markers acceleration, you must use the ChFunctions in motion tab.

Absolute acceleration -angular-
Angular acceleration of marker reference respect to the absolute (world) reference, expressed in marker coordinate system. This is a read-only value: if you want to impose markers speed, you must use the ChFunctions in motion tab.

Relative position
Position of marker respect to the body reference.

Relative alignment
Rotation of marker respect to the body reference. The rotation can be expressed as quaternions, Cardano angles, Eulero angles, Rodriguez parameters.

Relative speed -linear-
Speed of marker reference respect to the body reference. This is a read-only value: if you want to impose markers speed, you must use the ChFunctions in motion tab.

Relative speed -angular-
Angular speed of marker reference respect to the body reference, expressed in marker coordinate system. This is a read-only value: if you want to impose markers speed, you must use the ChFunctions in motion tab.

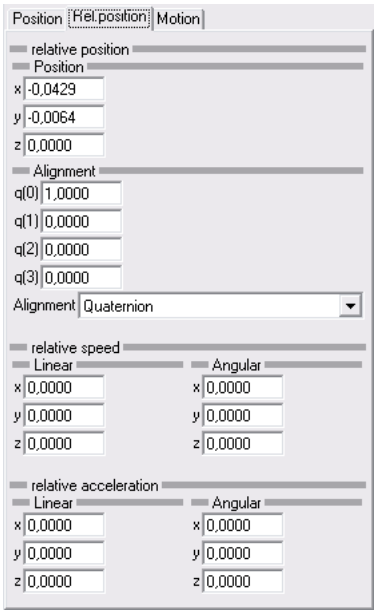


Figure 2.5 Marker body-relative coordinates

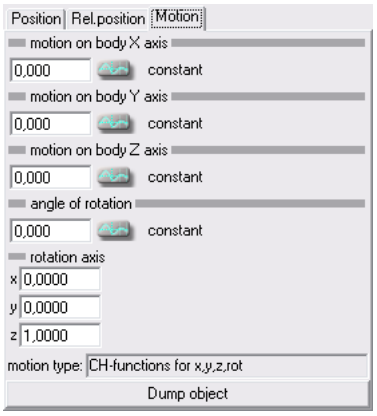


Figure 2.6 Marker imposed motion

Relative acceleration -linear-
Acceleration of marker reference respect to the body reference. This is a read-only value: if you want to impose markers acceleration, you must use the ChFunctions in motion tab.

Relative acceleration -angular-
Angular acceleration of marker reference respect to the body reference, expressed in marker coordinate system. This is a read-only value: if you want to impose markers speed, you must use the ChFunctions in motion tab.

Motion on body X axis
Use this ChFunction to impose the motion of marker respect to the owners body. The function will represent the motion law X=f(t) which defines the motion of marker along X axis of body. (Default is constant X=0, so the marker remains in its original position).

Motion on body Y axis
Use this ChFunction to impose the motion of marker respect to the owners body. The

function will represent the motion law Y=f(t) which defines the motion of marker along Y axis of body. (Default is constant Y=0, so the marker remains in its original position).

Motion on body Z axis
Use this ChFunction to impose the motion of marker respect to the owners body. The function will represent the motion law Z=f(t) which defines the motion of marker along Z axis of body. (Default is constant Z=0, so the marker remains in its original position).

Angle of rotation
Use this function to impose a rotation of marker respect to the owners body. The function will represent the motion law a=f(t) which defines the angle of rotation of marker about axis of rotation V. (Default is constant a=0, so the marker remains in its original position).

Rotation axis
Defines the rotation axis V (in body coordinate system) about which the marker will rotate, if the marker has non-default ‘angle motion’.

Motion type
Tells which method is being used to impose the position and motion of the marker (default is ChFunctions for x,y,z rot , but it can be also keyframed if the user has keyed the position of marker, etc.)

2.3 Force

Forces can be used to **apply forces** to rigid bodies . Force strength can be modulated in time and space with ChFunctions (non linearity in time and space)

Forces must be **contained in rigid bodies** .

A force can be **contained in a sub-level** of a rigid body .

Forces can be **saved on disk, copied and pasted** just like all other 3D objects of Realsoft3D .

Forces can be **moved and rotated** in 3D simply using the move and rotate tools of Realsoft3D , or using the coordinate handles and mouse. Note that you cannot extend' or scale a force object: the length of its axes is always kept constant.

Forces are **fixed** to their rigid bodies by default, and rotate as their bodies rotate. However you can set some flags and **prevent** their rotation or their translation with body frame.

This object can be used to apply **torques** as well.

The **point of application** of force (or torque) is represented by the **origin** of its coordinate system (objects handles in 3D space).

The properties of this object can be accessed and modified through the Javascript: see the documentation on force javascript object .

2.3.1 Force property window

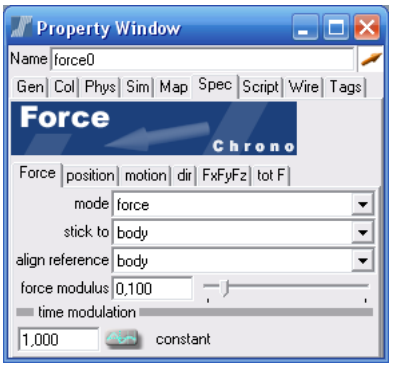


Figure 2.7 Properties of force object.

Mode
It can behave as force or torque. Force mode is the default. If in torque mode, the torque is applied on the axis used to define force direction in force mode.

Stick to
Application point of force can move together with body frame (default, stick to body) or can be fixed in space (stick to world).

Align reference
Direction of force (or torque) can rotate together with body frame rotation (default, align reference: body) or can remain fixed in space (align reference: world).

Force modulus
Modulus $\|F\|$ of imposed force, or imposed torque. Measuring unit: Newton $[N]$, the force which causes a unit acceleration ($1m/s^2$) for a unit mass ($1kg$).

Time modulation
A CH-function which can be used to modulate the force modulus (see above) as a function of time, that is $\|F\| := \|F\| \cdot f(t)$. Default is constant unit function $f(t) = 1$, that is no time modulation takes place.

Position
Application point of force respect to absolute (world) reference. This will change during body motion if stick to' is set to body, otherwise wont change if stick to' is set to world. (See stick to).

Rel.position
Application point of force respect to body reference. This wont change during body motion if stick to' is set to body, otherwise will change if stick to' is set to world. (See stick to).

X motion
Use this CH-function to impose the motion of application point of force. respect to the owners body. The function will represent the motion law $X = f(t)$ which defines the motion of force along X axis of body. (Default is constant $X = 0$, so the force remains in its original position).

Y motion
Use this CH-function to impose the motion of application point of force. respect to the owners body. The function will represent the motion law $Y = f(t)$ which defines the motion of force along Y axis of body. (Default is constant $Y = 0$, so the force remains in its original position).

Z motion
Use this CH-function to impose the motion of application point of force. respect to the owners body. The function will represent the motion law $Z = f(t)$ which defines the motion of force along Z axis of body. (Default is constant $Z = 0$, so the force remains in its original position).

Direction
Direction of force (or torque) respect to absolute reference. This will change during body motion if align reference' is body, or wont change if align reference' is world. (See align reference)

Rel.direction
Direction of force (or torque) respect to body reference. This wont change during body motion if align reference' is body, or will change if align reference' is world. (See align reference)

X force
Y force
Z force
If setting the force vector with modulus and direction isnt enough comfortable, you can use

Figure 2.8 Force components as separate X,Y,Z functions.

these three CH-functions to define separately the three components of the force along the X,Y,Z axis of body, as three separate functions of time. Note that the force obtained in this way is added to the default force (defined with modulus and direction).

tot.F -Force-
tot.F -Rel.force-
In the 'tot.F' tab you can see the total force applied to the body, as a three-dimensional vector. it is the sum of the force defined with modulus and direction and (if any) the three X,Y,Z forces defined with the three time-functions. These are read-only fields. In Force you get the total force vector expressed in the absolute reference, in Rel.Force you get the total force vector expressed in body-relative reference.

2.4 System

The System object represents a place to put the settings for the simulation engine of Chrono::R3D . All settings of the multibody solver can be accessed via this object (integration step, integration method, etc.)

Also, the System object behaves like a level, so other objects can be put inside.
In particular, this level is a nice place to put all the link objects which represent the constraints of your mechanical system (indeed this is the default behavior of the link creation tool).

Using the property window of the system object, you can access interesting functions like **Assembly analysis, Remove redundant links**, etc.

Also, global data like total number of degrees of freedom or total number of bodies can be found here.

Note: if you have multiple System objects in your system, only the settings of the last in the hierarchy will be used.
To avoid confusion, it is better to have only a single System object per each simulation, if possible.

You cannot find a tool in Realsoft3D interface which allow you to create a System object (Chrono::R3D does not add tool buttons for this), simply because the System object is automatically created when it is needed, if not yet present. Anyway, if you really need to create a System by hand, there is an item “Chrono System” in the popup menu of the select window, which creates a System.

The properties of this object can be accessed and modified through the Javascript: see the documentation on system javascript object .

2.4.1 System property window

Simulation type: Mode
Choose the simulation mode.

- **Dynamic simulation**, performs the full dynamical analysis of the mechanism.
- **Kinematic analysis**, perform the kinematic analysis (the system should have zero degrees of freedom, but if it has more than zero, a minimum square solution is provided).
- **OFF, no analysis**, nothing is done (Chrono::R3D won’t perform calculations during the animation playback).

Tools
Press these buttons for operations on the whole system.

- **Assembly**: tries to mount the mechanism if some constraints have been opened.
- **Statics**: finds the static equilibrium for the system, if possible. Note that for systems with high nonlinearities, you may need to press this more than once.
- **Delete keys**: if some Chrono::R3D bodies had been previously keyed, this button removes their keyframes at once.

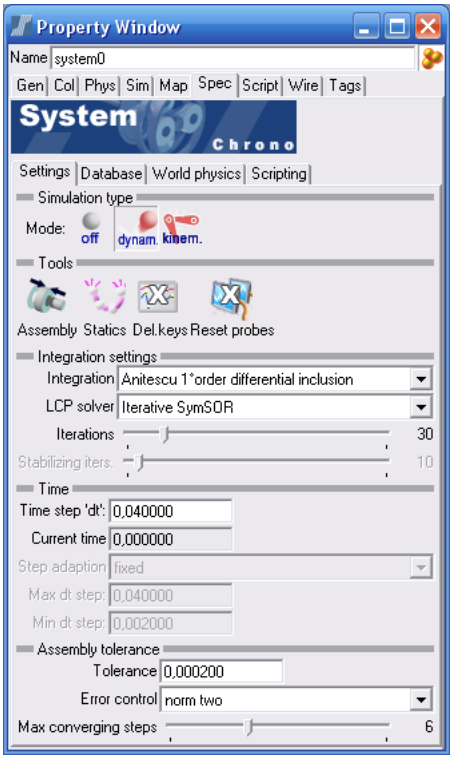


Figure 2.9 System specific properties.

- **Reset probes**: if Chrono::R3D Probe objects are present in the hierarchy, this button deletes all their recorded plottings at once.

Integration

Choose the integration scheme, that is the algorithm used to advance the time step by solving the differential inclusion problem at each instant. Currently, available methods are:

- **Anitescu**, a fast integration scheme which operates at the speed-impulse level. Accumulation of position error in constraints is avoided by using a stabilizing term. Though very fast, this method may cause a 'spongy' effect where you'd expect a rigid behavior of constraints (if so, either use lower timestep, or use the Tasora method). Also, simulating ill-posed collision scenarios affected by large interpenetrations may cause interpenetrated objects to move apart too quickly.
- **Tasora**, an integration scheme similar to the Anitescu, which operates at the speed-impulse level. However, accumulation of position error in constraints is avoided by solving an additional problem at the positions level. Usually this avoids the 'spongy' effect in constraints, but it is a bit slower. Also, this method is suggested when dealing with colliding systems affected by some large interpenetrations.

Currently all the integration methods above are explicit. As for all explicit methods, they can diverge if the integration step is too high in stiff systems.

LCP solver

All the integration methods above need to use a LCP-NCP (linear and non-linear complementarity problem) solver for finding unknown impulses at each timestep. In Chrono::R3D , you can currently choose among various iterative solvers. In most cases, the faster and more reliable are the SSOR and the SOR projected solvers.

Iterations

The LCP-NCP iterative solvers above require an upper limit on the iteration number. The higher this number, the more precise the simulation. The lower this number, the faster is the simulation. A default tradeoff is about 20-30, but it can be increased when dealing with a large number of interconnected objects. A sympthom of too low iteration number is the fact that constraints do not 'stay together' during the animation.

An higher amount of iterations is required when using high stacks of colliding objects.

An higher amount of iterations is required when using kinematic chains made of many parts (ex. long pendulums).

An higher amount of iterations is suggested when bodies with uneven mass ratio are connected (es. a 1kg wheel attached to a 1000kg body).

Time step 'dt'

This is the time step dt_{int} of the integrator.
(If the integrator is in variable-step mode, this is the current value of time step, ranging automatically between the two limiting extremals 'Max dt step' and 'Min dt step').

Note that the 'integration' time step dt_{int} can be smaller than the time step dt_{anim} of the animation system of Realsoft3D , for example you can play an animation with PAL framerate (each frame =1/25th of second, that is $dt_{anim} = 0.04[s]$), but in the meantime you can set a time step of $dt_{int} = 0.001[s]$ because you may need an accurate simulation (in such a case, each animation frame will contain 40 integration steps).
However, if this time step is higher than the animation step, it will be considered as small as the animation step.

Current time

Current time, in seconds. Read-only.

Step adaption

This option can be used to turn on the variable-step integrator, for more precise integration (but computation time may be unpredictable, and slower).

- **Fixed**, time step remains constant.
- **Variable**, time step changes (and becomes lower) if the integration error estimation grows too much, or if constraints tend to separate, etc. Also, the time step can grow if the integration precision is going on without problems.

Max dt step
Min dt step

Limiting extremals for the automatic variation of the time step, when using variable-step integrators. The 'Min dt step' allow you to limit the refinement of the integrator, in ill-cases where it would tend to use almost zero steps (and infinite computation time).

Tolerance

Tolerance ε_{tol}. Maximum value allowed for kinematic errors.
Used mostly for the Assembly tool.

Error control

Method used to compute the tolerance (type of norm computation).

Max converging steps
Maximum number of steps to be performed during the Newton-Raphson procedure in the Assembly procedure, which tries to keep the constraints closed. Usually, 3-4 steps are enough to reach average tolerances, when using mechanisms without bad or ill-posed constraints. Very rarely you will need higher limits, like 7 or 8 (usually, if the routine does not converge in less than 9-10 steps, there are something wrong in the way constraints are placed).

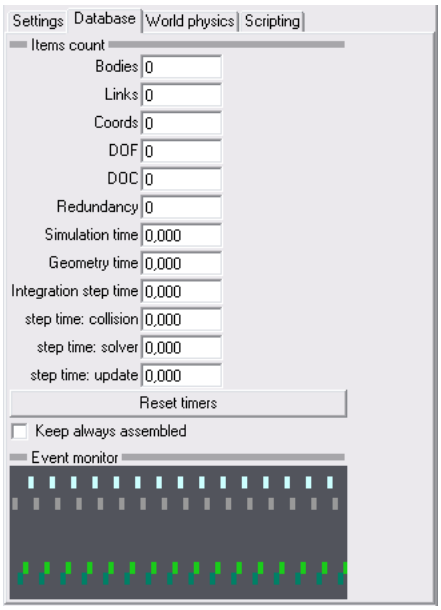


Figure 2.10 Other system-specific properties: database

Bodies
Number of free rigid bodies in the physical system.

Links
Number of links in the physical system.

Coords
Number of total coordinates (the size of the state vector). Note that each body introduces 7 coordinates since its rotation is represented by a quaternion, which is made of 4 scalars, not simply 3 angles.

DOF
Degrees Of Freedom, that is unconstrained motion coordinates. For over-constrained or hyper-static systems, it can be negative.

DOC
Degrees Of Constraint. Total number of scalar constraints added by link objects.

Redundancy
Total number of redundant constraints detected (***currently not activated***)

If a system contains redundant constraints, maybe you added unnecessary links , for example you are constraining a door to the wall by using two revolute joints.
This may be dangerous (in the case of the door with two revolute joints, if you misplace few millimeters the axis of one revolute, the door cannot be mounted). Though Chrono::R3D tolerates redundant constraints, they should not deny an admissible configuration, so try anyway to design mechanisms with the smallest amount of redundant constraints.

Simulation time
Computation time spent in pure Chrono::R3D simulation (time of PC clock). **Geometry time**
Computation time spent to update Realsoft3D geometry according to Chrono::R3D data (time of PC clock).

Reset timers
Sets to zero the counters above, that is *Geometry time* and *Simulation time*. You can use these values as 'chronographs' to discover how much computation time has been spent in Chrono::R3D simulation during an animation playback (the rest of the time being spent for usual Realsoft3D operations such as screen refresh, OpenGL drawing, etc.)

Event monitor
This large panel shows the sequence of the main events during Chrono::R3D operations. For example, during a dynamical simulation, it shows many coloured boxes, telling that for each animation frame you have update operations, integration time step advances, repetitions of the step, etc. All events are shown as colored boxes (the last enter on the right side, and the window scrolls to the left). To understand the meaning of the box, simply go with the mouse on the box and read the label which is automatically shown.

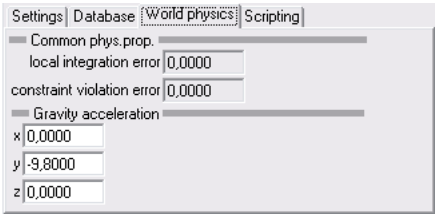


Figure 2.11 Other system-specific properties

Local integration error
This is the value of the local integration error ε_{int}, which can be computed by some integrators (for example Runge-Kutta 4/5). If the integration scheme does not provide local error estimation (ex: Anitescu, Tasora), this value will be zero. This is a read-only value.

Constraint violation error
This is the value of the norm of violations in geometrical constraints, `C`. Read-only.
You can think at it as an average measure of the instant 'clearances' in your links.

Gravity acceleration
Gravity acceleration g.
Note that Y axis is vertical, and acceleration points downward, so it takes a minus sign. On our planet, the Earth, $g_y = -9.8[m]/[s^2]$.

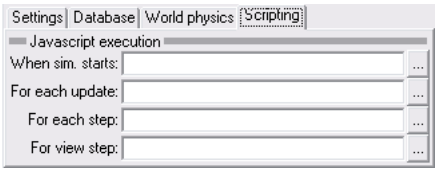


Figure 2.12 Other system-specific properties.

Scripting

In these four fields you can set the filename of Javascript programs which will be executed at the beginning of the animation, or at each time step, or at each update of system state, or at each animation step.
(Otherwise we suggest you to use a more flexible option, introducing the new Control object and setting the Javascript programs there).

2.5 Link

Each link object **defines a constraint** between two rigid bodies (two markers are used, one in each body, in order to create references for the link). In general, a link constraints the movement of a marker respect to the other.

Many types of links are implemented in Chrono::R3D : most are used to create typical constraints like revolute joints or spherical joints, but more advanced are provided as well, for example motors (imposing rotation between parts), linear actuators etc.

One of the two referenced markers (in general, it is the second) is called **master** because it is the **main reference** of the link. The other is called **slave**. For example, create a prismatic joint, and see how the prismatic guide' direction is defined by the master marker: if you move the position of that marker respect to the parent body, also the 3D representation of the prismatic guide will be moved. (In this example, the slave marker' will be always constrained to move on the prismatic guide defined by axis Z of master marker).

Link objects can be placed wherever in 3D object hierarchy, but we suggest to put them in the sub level of a System object (as happens by default, when they are created automatically by the Link creation tool).

A link is a level: it can contain other objects. For example, all links can contain, if wanted, probe which record interesting variables like reaction forces etc. Another example: the linear actuator' link can contain two objects which are automatically aligned to the piston in order to represent the two pieces which build a piston, etc.(see linear actuator specific chapter).

Deleting a marker which is used by a link **will invalidate the link** (it will get the state 'not active', and its icon in select window will become grey).

Selecting a link and moving or rotating it will have no effect, however you can select both the referenced markers and move them, or move each one separately: the link position will be updated.

The more links in the system, the slower the simulation. Therefore avoid using unnecessary or redundant or duplicated links if you want to **optimize simulation speed**.

The multibody engine of Chrono::R3D , exploiting advanced LDL^T decomposition, **can tolerate redundant links**, that is, links which define unnecessary constraints because already defined by other links, or ill-conditioned constraints. A typical example: if you have a door, you just need a single revolute joint link, but someone may be tempted to use two revolute joints as in real life doors (but doing so, you add unnecessary constraints). In similar situations, Chrono::R3D wont take in consideration redundant links -but remember that just one of the links will get the reaction forces!-

Despite Chrono::R3D can handle redundant or badly conditioned constraints, there are situations where **constraint cannot be satisfied** anyway. A simple example: a three-bar mechanism which should build a triangle, but where one of the three bars is much longer than the sum of the other two... this physically impossible! Other examples, are mechanisms where you add an engine (impose-rotation mode) which rotates a crank up to a configuration where the linkages will break. Only experiments and users knowledge can avoid these situations: Chrono::R3D can just issue warnings in the status bar ('Warning! Some constraints cannot be assembled [etc.]')

Most constraints offer the ‘limit’ feature, which can be used to impose **upper/lower limits** to the relative translation or rotation of the markers. This can be used to build links like the human knee, for example, which do not have full 360 rotation.

During the dynamic simulation, **constraint reactions** are computed for each link.

Special constraints, like gears, screws, motors, etc. are discussed in following specific chapters. Note that each special link has its own **custom properties** exposed in the Custom tab of the link property window.

The properties of this object can be accessed and modified through the Javascript: see the documentation on link javascript object .

2.5.1 Link property window

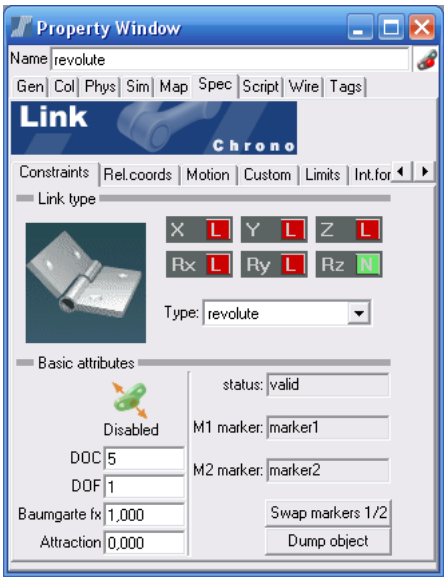


Figure 2.13 Properties of link object

.

Link mask
Each link type, in general, constraints some relative degrees of freedom of the two markers, and leave some other free. The 6 colored boxes of the link mask are a quick visualization of which relative motions (of marker M1 respect to M2) are constrained. Boxes X,Y,Z tell if degrees of translation of M1 on axes x,y,z of M2 are free. Boxes Rx, Ry, Rz tell if degrees of rotation of M1 about axes x,y,z of M2 are free. Red box means constrained, green box means unconstrained.

Type
Type of link: for most links, this field allow you to switch between different types, while for other links this is a read-only field, which cannot be changed since link creation. See later for a list of link types with an explanation of their behavior.

Disabled
Check this, if you want to temporarily turn off a link. This can be useful to simulate devices which break, constraint failures, and so on.

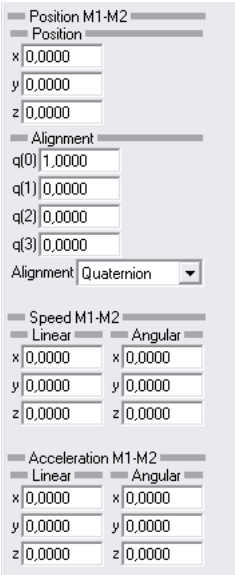


Figure 2.14 Link relative coordinates

DOC

Number of constraints enforced by this link. (DOC = Degrees Of Constraint). This is the number of scalar constraint equations added to the mechanical system because of the link (i.e. a single link may correspond to multiple scalar equations: for example a spherical joint adds 3 constraint equations, one for each x,y,z relative translation, and so on).

DOF

Number of degrees of freedom left free, for motion of marker M1 about M2 (it is, in general, DOF = 6-DOC)

Baumgarte fx

Proportional gain for the Baumgarte stabilizer, if using the link-stabilization technique for keeping links closed (this is unnecessary most times, since a more efficient technique is used by default: see System property window).

Status

Reports if the link is currently active, or inactive (for example if one of the two markers have been deleted).

M1 marker

Reports the name of the M1 marker (the slave' marker, whose motion is constrained to M2).

M2 marker

Reports the name of the M2 marker (the master' marker, which defines the main reference for link coordinates).

Swap markers 1/2

Press this button to swap the role of the two markers (M1 will become master' marker M2, and viceversa).

.

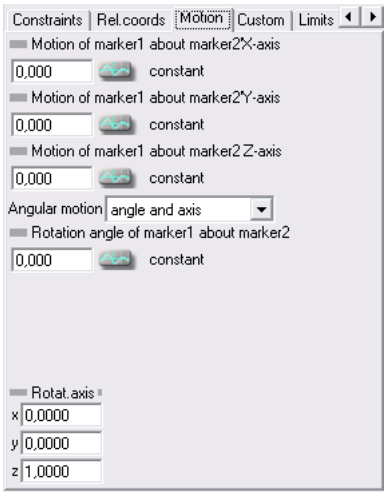


Figure 2.15 Imposed motion (case of 'lock' linnk type).

Position M1-M2
Position of marker M1 respect to master marker M2, in M2 coordinate system. This is a read-only value: it depends on mechanism motion.

Alignment M1-M2
Alignment (rotation) of marker M1 respect to master marker M2, in M2 coordinate system. Rotation can be expressed as quaternions, Cardano angles, Eulero angles, Rodriguez parameters. This is a read-only value: it depends on mechanism motion.

Speed M1-M2 -linear-
Speed of marker M1 respect to master marker M2, in M2 coordinate system. This is a read-only value: it depends on mechanism motion.

Speed M1-M2 -angular-
Angular speed of marker M1 respect to master marker M2, in M2 coordinate system. This is a read-only value: it depends on mechanism motion.

Acceleration M1-M2 -linear-
Acceleration of marker M1 respect to master marker M2, in M2 coordinate system. This is a read-only value: it depends on mechanism motion.

Acceleration M1-M2 -angular-
Angular acceleration of marker M1 respect to master marker M2, in M2 coordinate system. This is a read-only value: it depends on mechanism motion..

Motion of M1 about M2
In the 'Motion' tab you may find some ChFunction editors. These can be used to emphimpose motion for the degrees of freedom which are 'locked' in the selected link, that is, you can create time-dependent displacements for the degrees which have the red label 'L' (locked) in the link mask. Since each link has a different mask of locked motions, these ChFunctions can be selectively available (for example, the 'free' link does not constrain any degree of freedom, so there will be no 'Motion' editors available. On the other hand, the 'Lock' link constrains all 6 DOF, hence you will be able to impose translation and rotations in these 'Motion' editors, and so on.)

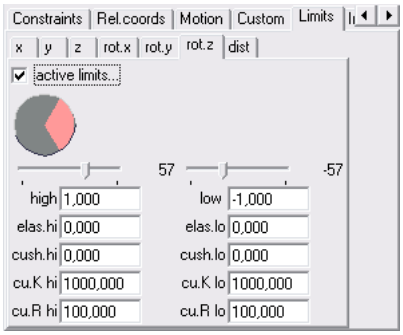


Figure 2.16 Limits for free motions.

These ChFunctions impose the motion of marker M1 about marker M2, in the coordinate system of M2. You can use this functionality to create basic rheonomic (i.e. time-dependent) constraints such as motors, linear actuators, etc. (However we suggest you to use more advanced link types like 'Motor' , 'Linear actuator' , etc, whenever possible).

Limits

In the 'Limits' tab you find some sub-tabs for editing limits. These can be used to impose MIN-MAX limits for the degrees of freedom which are 'free' in the selected link, that is, you can make upper and lower stops for rotations or translations which have the green label 'F' (free) in the link mask. Of course each link has its internal DOFs, so limit editors arent always available (for example, the 'revolute' type can define just one upper-lower limit: the one acting on relative rotation, like in human elbow). These limits act on the motion of marker M1 about marker M2, in the coordinate system of M2. Each limit has some sub-settings:

- active limits.. can be used to turn ON/OFF the upper/lower limits for that degree of freedom,
- High / Low are the upper and lower limits. These are in meters for translations, or radiants for rotations (but rotations have also a slider with degrees),
- elas.hi / elas.lo are the restitution coefficient for the impact at the upper-lower ends of the limit (0..1 range, where 0 means fully anaelastic impact, like sticking', and 1 means fully elastic bounce),
- cush.hi / cush.lo are the thickness of two cushions' at the upper-lower ends, with user defined stiffness and damping. These act with spring-damper force, before the true geometric ends are reached, so they can create less impulsive' limits,
- cu.K hi / cu.K lo / cu.R hi / cu.R lo are the stiffness and damping of monolateral spring-damper systems -with length as cushion thickness- at the upper-lower ends. (These values have no meaning if cush.hi and cush.lo thickness are null.)

Int. forces

In the 'Int. forces' tab you find some sub-tabs for editing user-imposed forces. These can be used to impose forces for the degrees of freedom which are 'free' in the selected link, that is, rotations or translations which have the green label 'F' (free) in the link mask. Of course each link has its internal DOFs, so force editors arent always available. Forces (or torques, in case of rotations) are applied to marker M1 -as well as the opposite reactions are applied on marker M2- in order to simulate, for example, springs which are mounted inside the joint. If a link has free translations (like in a prismatic joint), the correspondent x or y or z editors are available, and are used to add forces, directed as x or y or z axes of marker M2. If a link has free rotations (like in revolute joints), the correspondent rot.x or rot.y or rot.z

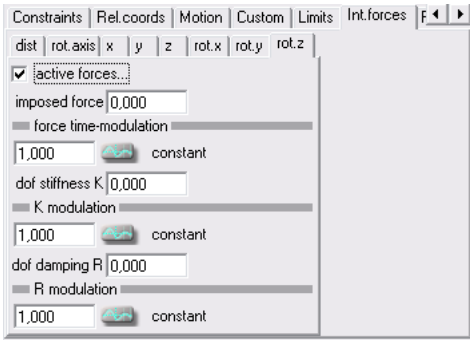


Figure 2.17 User-imposed forces for free motions.

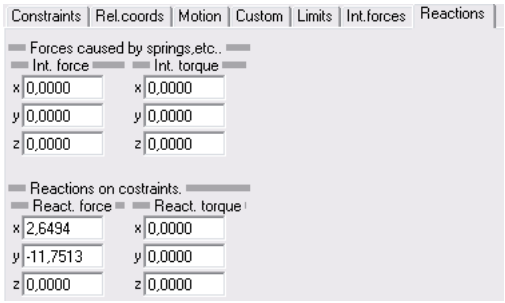


Figure 2.18 Link reactions

editors are available, and are used to add torques, applied about x or y or z axes of marker M2. In some link types, you can also add forces acting along the polar distance ('dist' force editor) or torques applied about the instant rotation axis ('rot.axis' force editor). Each force editor has the following settings:

- active forces... can be used to turn ON/OFF the force feature for that degree of freedom
- imposed force is the modulus m of the force,
- force time-modulation is a ChFunction $f_F(t)$ which can be used, if different from default unit-constant, to make a force whose modulus changes with time,
- dof.stiffness K is the stiffness $k[N/m]$ of a spring mounted along that degree of freedom, if degree is a rotation, this is a torsional spring $[N/rad]$.
- K modulation is a ChFunction $f_K(x)$ which can be used to make stiffness K dependant on coordinate x of degree of freedom (x is in meters, or radiants for rotations)
- dof.damping R is the damping $r[Ns/m]$ of a damper mounted along that degree of freedom, if degree is a rotation, it is a rotational damper $[Ns/rad]$
- R modulation is a ChFunction $f_R(x)$ which can be used to make damping R dependant on coordinate x of degree of freedom (x is in meters, or radiants for rotations)

Final force is, in the most general case:
$F = m \cdot f_F(t) + d \cdot k \cdot f_K(x) + dx/dt \cdot r \cdot f_R(x)$
If this level of customization isnt enough, remember that you can still use Chrono::R3D Javascript scripting (see link variables int_force and int_torque to create whatever force field with user formulas and programs.

Int. force

Internal forces acting on M1 (for example caused by springs, cushions' in limits, user-imposed forces, etc.), excluding constraint reactions (which are shown in the separate fields

react force and react torque). Application point is considered M1, coordinate system is master marker M2. (On marker M2 theres an equal force, but in opposite direction).

Int. torque
Internal torques acting on M1 (for example caused by torsional springs, cushions' in limits, user-imposed forces, etc.), excluding constraint reactions (which are shown in the separate fields react force and react torque). Coordinate system is master marker M2. (On marker M2 theres an equal reaction, but in opposite direction).

React. force
This is the vector representing the force (acting on M1) caused by link reactions. Application point is considered M1, coordinate system is master marker M2. Of course on marker M2 theres an equal reaction, but in opposite direction.

React. torque
This is the vector representing the torque (acting on M1) caused by link reactions. Coordinate system is master marker M2. Of course on marker M2 theres an equal reaction, but in opposite direction.

2.5.2 Link types

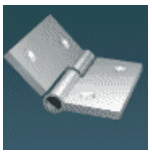
Here is a list of most common link types in Chrono::R3D . You can choose between them when you use the link tool to create a Link object.

The following list shows the link icon (the little picture which is shown in the link property window) and a short description of link behavior.

For the more advanced links, such as linear actuator , motor , spring-damper , gear , screw , brake , pneumatic cylinder , there are separate descriptions in their own chapters.

Free
No constraints are imposed. Anyway this link can be used for adding spring-damper systems, probes, etc.
For example it can be added when you need to measure the relative motion of the two markers M1 and M2.

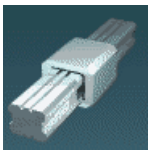
Lock
The two bodies are completely locked together in both rotation and translation, as if they were 'glued' or welded together.
By the way you can use this link if you are interested in the internal reactions of a rigid structure (say, a bar): you can divide the structure in two parts and study the reaction torques and forces in this link.
Note: use the Lock link only if you really need it (for instance, you are interested in reactions), otherwise, if some component of your mechanism is composed of many 'welded' parts, it may be wise to use a single body instead of using many bodies joined together with many 'Lock' links, because simulation speed would be faster.



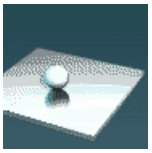
Revolute
This link constraints the translation of marker M1 respect to marker M2, and constraints the rotations about axis X and Y of marker M2. Hence, marker M1 can rotate on the Z axis of marker M2.
This is one of the most used constraints, since it is used to build articulated mechanisms, leverages, slider-cranks, etc. It is similar to the cylindrical link, except that translation along Z is not allowed.

Cylinder
This link constraints the translation of marker M1 respect to marker M2, except for translation on Z axis of marker M2, which is allowed. Also it constraints the rotations about axis X and Y of marker M2. Hence, marker M1 can rotate and translate on the Z axis of marker M2.
This is one of the most used constraints, since it is used to build articulated mechanisms, leverages, slider-cranks, etc. It is similar to the revolute link, except that translation along Z is allowed.

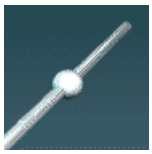
Spherical
This is a spherical joint which allows Rx Ry Rz rotations between the bodies. Only translation of marker M1 respect to M2 is constrained. This is one of the most used constraints. It can be used to represent biological constraints such as the shoulder joint or the hip joint.



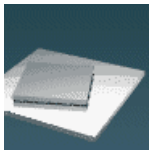
Prismatic
Only relative translation along prismatic joint direction is allowed, no rotations are allowed. In detail, marker M1 can translate along axis Z of marker M2, while their alignment remain the same.
This constraint can be used to represent rectilinear glyphs, linear guides, etc.



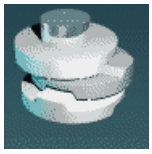
Point on plane
The M1 marker can translate on the plane defined by the M2 marker, rotations are allowed.
In detail, the plane is described by the X-Y axes of marker M2 (translation of marker M1 along axis Z of marker M2 is forbidden).



Point on line
The M1 marker can translate on the X axis of the M2 marker, rotations are allowed. This link can be used to make rectilinear glyphs, etc. If you need to constrain a point on a curved line, you should use the Point on spline link.

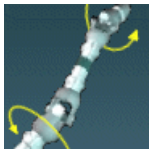


Plane on plane
The two planes defined by M1 and M2 markers must be coplanar while shifting, or rotating on perpendicular direction.



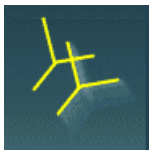
Oldham joint
The 'Oldham joint' transmits rotation between two parallel (maybe non coaxial) shafts. This kind of joint is often used in mechatronic applications, where electrical motors must be connected to devices which may be mounted with some displacement between the two axes (but no errors are admitted in angular misalignment: axes are kept parallel anyway).

Universal joint
The rotation about a shaft is transmitted homo-kinetically to another shaft, even not parallel. Position of M1 and M2 is the same. Note that this could be considered equivalent to the 'Rzeppa' or 'Birfield' joints, often used to transmit motion in automotive mechanisms.

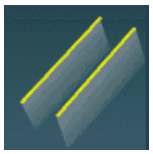


Cardano shaft
The rotation of a shaft is transmitted homo-kinetically to another shaft, even not parallel. Position of M1 and M2 can be free, as if you are using an extensible shaft. Note that this could be considered equivalent to an extensible shaft with two 'Rzeppa' or 'Birfield' joints at the ends, often often used in automotive mechanisms. Also note that, despite the name, this kind of devices does **not** use the 'Cardano' joints at the ends, because these would be homo-kinetic only in some specific conditions.

Hook
This link constraints the three relative motions of the two markers, and one rotation. It works like a homo-kinetic joint (see the universal link), but along axis Y of marker M2.



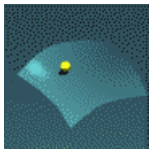
Align
The rotation of the two markers in space must be the same (coordinate systems'axes must have the same alignment, but their relative translation is free)



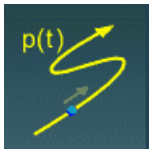
Parallel
Two axes of M1 and M2 markers are kept parallel. However, their relative translation is free.

Perpendicular
An axis of the M1 marker is kept perpendicular to a plane defined by the M2 marker. However, their relative translation is free.

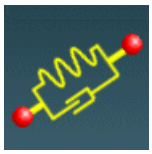
Point on spline
The marker M1 can shift along the spline referenced in sub level of marker M2.
In detail, marker M2 must contain the "*Realsoft3D link*" (the one which you create with the Realsoft3D tool set) which references a spline. Such spline is, most often, contained in a sub-level of body 2. (If you create this link with the link tool , this referencing will be set up automatically and you won't worry about these details).



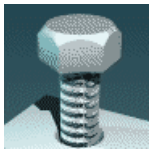
Point on surface
The marker M1 can shift over the Nurbs surface referenced in sub level of marker M2.
In detail, marker M2 must contain the "*Realsoft3D link*" (the one which you create with the Realsoft3D tool set) which references a Nurbs surface. Such surface is, most often, contained in a sub-level of body 2. (If you create this link with the link tool , this referencing will be set up automatically and you won't worry about these details).



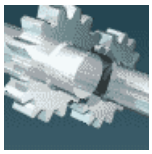
Trajectory
The marker M1 follows the imposed trajectory, defined with spline contained in sub level of marker M2.
Rotation is not constrained.
Note that the speed along the trajectory depends on the parametrization of the spline: if you need a constant-speed motion along a smooth curve, make sure that you set the proper uniform parametrization for that spline.



Spring-damper
Simulates a spring-damper between two markers.
Note: this link type is described in more detail in its own chapter .



Screw
This link constraints the mutual roto-translation of two bodies, given a thread lead.
Note: this link type is described in more detail in its own chapter .



Gear
The rotations of two bodies (two gears) are constrained, given a transmission ratio.
Note: this link type is described in more detail in its own chapter .

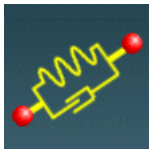
Linear actuator
Simulates a linear actuator between the two markers.
Note: this link type is described in more detail in its own chapter .

Engine
Simulates the effect of a motor which constraints the rotation of marker M1 respect to marker M2. Shaft is represented by Z axis of marker M2.
Note: this link type is described in more detail in its own chapter .

Brake
Simulates the effect of a brake which affects the rotation of marker M1 respect to marker M2. Shaft is represented by Z axis of marker M2.
Note: this link type is described in more detail in its own chapter .

Pneumatic actuator
Simulates the effect of a pneumatic actuator applied between the two markers.
Note: this link type is described in more detail in its own chapter .

2.6 Spring



.
This is a special link which implements a force-spring-damper system acting along the line which joins two markers .

You can create springs with **non-linear** behavior, either for the stiffness and for the damping, and you can set forces which change as functions of time.

The properties of this object can be accessed and modified also through Javascript: see the documentation on spring javascript object .

2.6.1 Spring property window

The property window of the Spring-damper link is the same of basic links , but also adds the following properties in the 'Custom' tab:

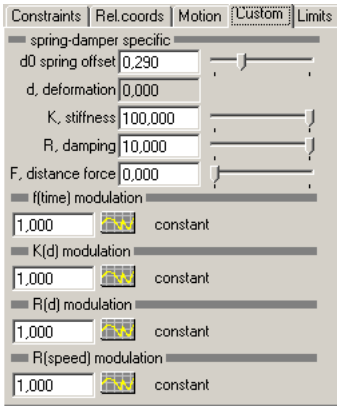


Figure 2.19 Custom properties of spring-damper

d0 spring offset
This is d_0, the 'rest length' that gives no strain force in spring (undeformed spring length).

d, deformation
Read-only value, telling how much is deformed the spring.

K, stiffness
The K_d stiffness of the spring, in [N/m].

R, damping
The R_d damper coefficient, in [N/(m/s)].

F, distance force
The F_d force, in Newtons [N], which acts along the line which joins the two markers.

F(time) modulation
A $f_F(t)$ CHfunction which can be used to modulate the force F_d during time. Default constant =1, for no time-modulation effect.

K(d) modulation
A $f_K(d)$ CHfunction which can be used to modulate K_s stiffness as a function of distance, for a non-linear stiffness effect. Default constant =1, for no effect.

R(d) modulation
A $f_R(d)$ CHfunction which can be used to modulate R_d damping as a function of distance, for a non-linear damping effect. Default constant =1, for no effect.

R(speed) modulation
A $f_V(v)$ CHfunction which can be used to modulate R_d damping as a function of polar speed v, for a non-linear damping effect. Default constant =1, for no effect.

In general, given polar distance d and polar speed v, the resulting force applied along the line which joins the two markers will have total modulus:
$F = d \cdot K_d f_K(d) + d \cdot R_d f_R(d) f_V(v) + F_d \cdot f_F(t)$
(If this is not enough for your needs, you can create even more complex non linear formulas by using the Javascript programming).

2.7 Motor

.
This is a special link which represents a motor (or a motor/reducer) between two parts.
Simulates the effect of a motor which constraints the rotation of marker M1 respect to marker M2. Shaft is represented by Z axis of marker M2.

This can act in many different ways, for example imposing **rotation**, or **rotation speed**, or **torque**.

The motor acts **on the Z axis** of the master marker (i.e. is the axis of rotation of the slave marker about the master marker).

In case the **'reducer' option** is used, the user can set reducer efficiency and transmission ratio.

Also, this kind of link can work in **learn mode** for robotic applications.

The properties of this object can be accessed and modified also through Javascript: see the documentation on motor javascript object .

2.7.1 Motor property window

The property window of the Motor link is the same of basic links , but also adds the following properties in the 'Custom' tab:

'Engine specific' scheme
The schematic picture of a motor and a reducer, visible in this 'Custom' panel, is a visual aid for the user. In fact it shows with a yellow label the variable which is imposed by the user (maybe the shaft speed ω, or the shaft angle α, etc. depending on the 'Mode'). The remaining labels are shown in blue, to tell that these will be computed as dependant variables.

Mode
The motor object has three main operational modes:

- Impose rotation
  The user can introduce the rotation of shaft as a function of time, $\alpha = \alpha(t)$ (angle measured in radiants $[rad]$), using the a(t) ChFunction editor above.
- Impose speed
  The user can introduce the speed of rotation of the shaft as a function of time, $\omega = \omega(t)$ (angular speed measured in radiants/second $[rad/s]$), using the w(t) ChFunction editor above.
- Impose torque(w)
  The user can introduce the torque applied on the shaft as a function of time and angular speed, that is $M = M(\omega, t)$. In detail, two ChFunction editors will be shown, to set M(t) (dependency $M_t(t)$ on time) and M(w) (dependency $M_w(\omega)$ on angular speed). In detail, final torque is $M = M_w(\omega) \cdot M_t(t)$.

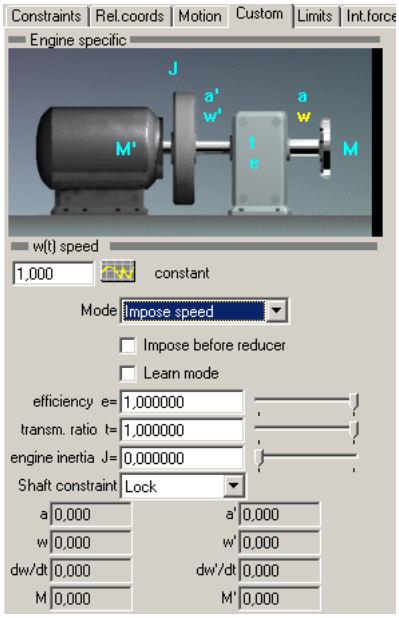


Figure 2.20 Custom properties of motor

- Impose keyframed rot.
  The rotation of the shaft can be keyframed. A choreography can be used to control the rotation (see the animateable property ”EngineRz” in the choreography window). An easy way to set keyframes for this property is to use the Chrono::R3D IK tool when the red ”Animation Recording” button is turned on: if the user rotates the parts with the mouse, the rotation value is stored automatically into a keyframe.
- Impose keyframed polar rot.
  As above, but also allows rotation on other two axes (as a spherical joint), then keyframes can be used for the properties ”EngineRz”, ”EngineRx”, ”EngineRy”. Note: for this mode, torque and rotation values shown in this tab of the interface do not have a physical meaning.

Note: when using the ‘Impose rotation’ or ’Impose speed’, this link applies a geometric constraint on the rotation, as for a motor controlled with infinite-stiffness and infinite precision. However, the ’Impose speed’ mode does not check that angle is exactly the integral of angular speed (after a lot of simulation time, numerical errors may introduce drifting in angle, even if instant speed is always correct).

Note: the ‘Impose torque(w)’ mode does not provide any control on either rotation and angular speed of the motor: there is no geometrical constraint but only an applied torque. Therefore, if the M(w) curve has a weird shape and/or there are no braking torques in your system, the shaft may accelerate up to very high speeds (until the integration step isn’t short enough to assure precision, and the integration may diverge).

Impose before reducer

If this is checked:

- in ‘Impose rotation’ mode:
  the imposed rotation angle is considered on the motor shaft (on the left of the reducer, in figure 2.20) instead that on the output shaft.

- in ‘Impose speed’ mode:
  the imposed angular speed is considered on the motor shaft (on the left of the reducer, in figure 2.20) instead that on the output shaft.
- in ‘Impose torque’ mode:
  the applied torque is considered on the motor shaft (on the left of the reducer, in figure 2.20) instead that on the output shaft.

This can be very useful because most industrial motors have high rotating speeds, and may be useful to simulate the reducer effect (with efficiency η and transmission ratio τ, see below).

Learn mode
If checked, the engine will be ‘free’ on the rotational degree of freedom, but its motion will be recorded into a ChFunction (rotation angle will be recorded if it was in ‘Impose rotation’ mode, or angular speed will be recorded if it was in ‘Impose speed’ mode).

This can be useful when studying robots: you can put all the motors of the robot in ‘learn’ mode, then you can move the end-effector with (for example) a ‘trajectory’ link. When the simulation is finished, all the motors will have the inverse-kinematics motion recorded into their ChFunctions , so the trajectory link can be deleted and the ‘learn’ flag of the motors can be switched off: the robot now will be moved by the motors (in ‘Impose rotation’ mode) and torque requirements can be studied.

Efficiency
Efficiency η of the reducer. Lost power is $W_l = W_{in} \cdot \eta$. In general, most industrial reducers have efficiency near 1.

Transm.ratio
Transmission ratio of the reducer, τ. Given input speed ω_{in} and output speed ω_{out}, it is $\omega_{out} = \tau \cdot \omega_{in}$.

Engine inertia
The engine shaft can have an extra inertia (if not already modelled in the rest of your 3D system). Usually, here you can put the sum of the inertias of the electrical engine shaft, the flywheel (if any), and a magnetic brake (if any). Note that this inertia, measured in $[kgm^2]$, is usually a very low value, when using small engines $(< 10kW)$.

Shaft constraint
Here you can choose what happen to the other five degrees of freedom of the motor link, apart from the Z constraint which is used for rotation. See what happen to the link mask when changing this setting.
(In Lock mode, for example, the link constraints also the rotation and translation of the two linked markers, so that there is no need to add a ‘revolute’ link to keep them with Z axes aligned).

a
Instant rotation angle α, on the output axis of the reducer (the one which acts on the linked parts).

w
Instant angular speed ω of the output axis of the reducer.

dw/dt
Instant angular acceleration $d\alpha/dt$ of the output axis of the reducer.

M
Instant torque M on the output axis of the reducer.

a'
Instant rotation angle α' of the motor shaft, before the reducer (the shaft which enters the reducer, on the left of figure 2.20).

w'
Instant angular speed ω' of the motor shaft, before the reducer.

dw'/dt
Instant angular acceleration $d\alpha'/dt$ of the motor shaft, before the reducer.

M
Instant torque M on the motor shaft, before the reducer.

2.8 Linear actuator

.
This is a special link which implements a linear actuator acting along the distance between two points (two markers in 3D space).

This link, when used with a constant length, acts like a rod with two ball joints at the ends. In fact, it **constraint the distance** between the two markers.

The Linear Actuator, if needed, supports a quick way to show telescopic struts in 3D views. In fact you can put two cylinder-like objects in the link level (see figure 2.21), and these shapes will will be automatically aligned with their Y axes along the joining line between the two markers. Also, origin of the two shapes will be moved to positions of the two ends, that is the positions of the two markers. The Linear Actuator will automatically keep these two cylindric shapes aligned as in telescopic actuators. Remember: this is only an 'aesthetical' approach to show 3D linear actuators, so the telescopic parts will be considered without mass! (If you need more advanced effects, you must build the telescopic strut with two real rigid bodies, two spherical joints and a prismatic or cylindrical guide.)

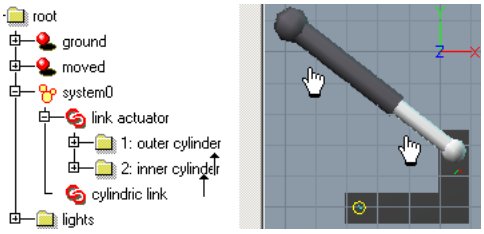


Figure 2.21 A way to show telescopic struts.

The properties of this object can be accessed and modified also through Javascript: see the documentation on linear actuator Javascript object .

2.8.1 Linear actuator property window

The property window of the Linear Actuator link is the same of basic links , but also adds the following properties in the 'Custom' tab:

d(t) extension
This is the ChFunction which can be used to set the extension of the actuator $d(t)$ as a function of time.

d0 offset
Initial length d_0 of the strut, for zero extension. In fact the total length will be $d = d_0 + d(t)$.

Learn mode
If checked, the distance constraint will be inactive, and the actuator can be extended freely, while the $d(t)$ ChFunction will record the extension over time. Later, you can turn off this option, after 'learning', and the actuator will enforce the distance constraint, reproducing the motion $d(t)$ which it learned in the last simulation.

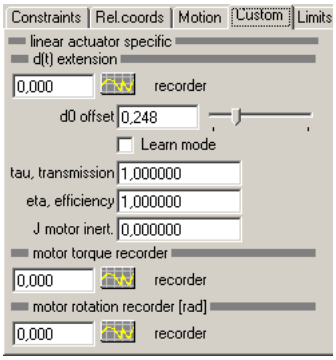


Figure 2.22 Custom properties of Linear Actuator link

tau, transmission
The transmission ratio τ, in case you consider the telescopic strut being moved by an electrical motor and a reducer with transmission ratio $\tau[m/rad]$.

eta, efficiency
The efficiency η, in case you consider the telescopic strut being moved by an electrical motor and a reducer with efficiency η.

J, motor inertia
The motor inertia J, in case you consider the telescopic strut being moved by an electrical motor with polar inertia J. You may add here also the inertia of the reducer (in motor coordinates)

Motor torque recorder
In this ChFunction you will find the recording of the torque which the motor required to enforce the specified motion, given motor η, τ, J. You can also use the probe object to record this variable, and others.

Motor rotation recorder
In this ChFunction you will find the recording of motor rotation for the specified motion, given reducer τ. You can also use the probe object to record this variable, and others.

2.9 Brake

.
This is a special link which implements a brake (or a clutch).

In detail, it creates a **frictional force** between two parts, such as in the case of two mated discs, which are pressed together in order to synchronize two shafts (so it is called 'clutch'), or to stop the motion of a shaft respect to a truss (so it is called 'brake'). Note that, from the rather abstract point of view of applied mechanics, brakes and clutches are the same things (a device which applies frictional torque between two parts).

This constraint can simulate the effect of the **stick-slip** adherence, caused by the fact that static friction coefficient is always higher that kinetic friction coefficient, for most materials. In this way, when the relative angular speed of the two mated discs becomes zero, a 'sticking' effect comes into place.

The properties of this object can be accessed and modified also through Javascript: see the documentation on brake javascript object .

2.9.1 Brake property window

The property window of the Brake link is the same of basic links , but also adds the following properties in the 'Custom' tab:

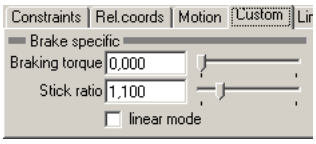


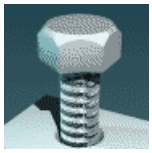
Figure 2.23 Custom properties of Brake link

Braking torque
Instantaneous applied braking torque $[Nm]$. Note that you can move this slider during the simulation, to modulate the braking effect. If zero, no braking at all. Also, try to control this value with ChControls.

Stick ratio
Ratio r_s between static friction coefficient f_s and kinetic friction coefficient f_k, that is $r_s = f_s/f_k$. Usually, this value is near 1.1 for many materials.

Linear mode
If this is checked, the link turns into a 'linear' brake (that is, the slave marker motion about Y axis of master marker is braked). Note that also the 3D symbol of the brake will change (it becomes a linear guide, instead of a disc). This is useful if you want to add the stick-slip frictional effect to other parts of your systems, for example you can add this link into your simulations of pneumatic actuators , in order to reproduce the effect of the friction of the rod in the cylinder.

2.10 Screw



.
This is a special link which implements a screw (i.e couples translation and rotation by a transmission ratio coefficient).

It is not necessary that the screw (slave part) is moving and the 'hole' (master part) is steady: you can build screw couplings where both the joined parts move and rotate.

This constraint can be used to simulate the ball-screws often used for robotics, automatic machines, linear systems, etc.

The constraint works a bit like the cylindrical link, where the Z axes of both the markers must be aligned, but in this case the motion of the slave marker **along the Z direction of the master marker** must be coupled to the rotation about that axis.

Pay attention to the fact that, for inverse kinematics and dynamical simulation, you should not have objects which spin too fast (in general you should have at least 20 steps of simulation for each revolution of the fastest spinning part). Therefore, if you simulate a screw whose translation is imposed, i.e. it 'pulled away', for small threads you can expect a very high rotation speed and the simulation may 'loose' some screw turnings if the integration step isn't small enough.

The properties of this object can be accessed and modified also through Javascript: see the documentation on screw javascript object .

2.10.1 Screw property window

The property window of the Screw link is the same of basic links , but also adds the following properties in the 'Custom' tab:

Figure 2.24 Custom properties of Screw link

Screw thread
The thread p of the screw, in meters $[m]$. The lower this value, the faster the screw must spin to move in and out. The transmission ratio between linear motion x and rotation α is $\tau = p/2\pi$, for $z = \tau\alpha$.

2.11 Gear

.
This is a special link which implements a gear between two wheels. (that is, simulates the effect of teeth contact between two wheels.).

The Gear link must be applied to two wheels which are **already constrained** in their rotation (for example, they already have two constraints of type revolute which let them spin about a truss). In fact the gear link adds just a single degree of freedom, represented by the contact between the teeth.

Unlike most other links , the Gear link does not use the two markers as geometric references. In fact it uses references to two 3D objects (each belonging to the corresponding wheel) which represent the Z axes of the two **shafts**. To implement referencing to the two shafts, inside the two markers you may put two Realsoft3D 's LINK objects (not to be confused with the Chrono::R3D links), each referencing the 3D object whose Z axis will be the shaft, as in figure **??**.
If there's no LINK to 3D object in markers, the bodies 's shaft will be considered as the Z axes of the coordinate systems of the bodies.
The link tool sets up the referencing to shafts automatically during link creation.

Note that the distance between the two shafts must remain the same during the simulation (as in most applications, the two wheels should be **connected to the same truss**, with cylindrical or revolute links). However, the truss can move in space as you like (as in epicycloidal gears), and the two shafts can be coincident (for conical gears).

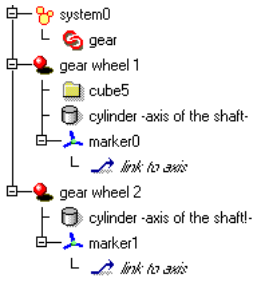


Figure 2.25 Gear link with referencing to 3D shafts.

The properties of this object can be accessed and modified also through Javascript: see the documentation on gear javascript object .

2.11.1 Gear property window

The property window of the Gear link is the same of basic links , but also adds the following properties in the 'Custom' tab:

Transmission ratio
The transmission ratio of the gear τ. It is the ratio between the angular speed of the two wheels (minus sign).

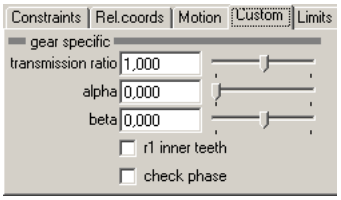


Figure 2.26 Custom properties of Gear link

Alpha

The pressure angle α, telling the angle of the tooth reaction force respect to the wheel tangent. Usually industrial gears are built for $\alpha = 20^o$.

Beta

The helix angle β, telling the angle of teeth on the primitive cylinder, for helicoidal gears. Different values of α and β will give different results for reaction forces on wheel ball bearings.

R1 inner teeth

If this is checked, the first wheel is a inner gear. This allow you to make inner gears, with positive transmission ratio.

Check phase

If this is turned ON, the Gear link constantly checks that the two wheels never ‘miss a tooth’, and phasing is always correct even after many turns. Otherwise it may happen that, after many turns (because of numerical approximations) there is a small drifting in phasing, as for friction wheels, or toothless wheels.

2.12 Pneumatic actuator

.
This is a special link which implements a pneumatic linear actuator applied between the two markers.

Unlike the Linear Actuator link, it does not enforce an exact geometric constraints on the distance between the two joined markers, but it rather **applies a force** caused by air pressure inside the piston.

You can consider the piston as being connected by means of two ball joints at the ends. The two joined markers are used to represent the ends of the actuator.

Like the Linear Actuator , this Pneumatic Actuator (if needed) may support a quick way to show telescopic struts in 3D views. In fact you can put two cylinder-like objects in the link level (see figure 2.27), and these shapes will will be automatically aligned with their Y axes along the joining line between the two markers. Also, origin of the two shapes will be moved to positions of the two ends, that is the positions of the two markers. The Pneumatic Actuator will automatically keep these two cylindric shapes aligned as in telescopic actuators. Remember: this is only an 'aesthetical' approach to show 3D pneumatic actuators, so the telescopic parts will be considered without mass! (If you need more advanced effects, you must build the telescopic strut with two real rigid bodies, two spherical joints, a cylindrical guide and the Pneumatic Actuator.)

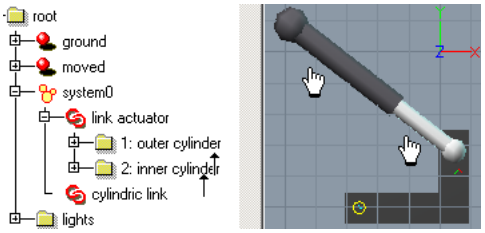


Figure 2.27 Easy way to show telescopic struts.

This link does not include friction and sticking effects, but you add them to your model by adding a brake link in 'linear' mode.

You can easily control the valve opening of this Pneumatic Actuator objects by using Javascript programming. For example, you can introduce a controls object in your device, where the 'controls' object executes a Javascript program each simulation step, which opens/closes valves depending on a simulated PID controller, or by simulating a PLC circuit, etc.

The properties of this object can be accessed and modified also through the Javascript: see the documentation on pneumatic actuator javascript object .

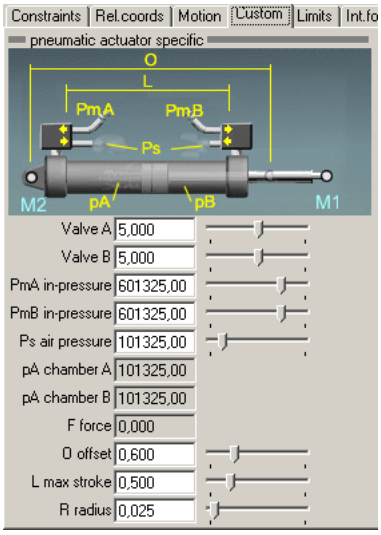


Figure 2.28 Custom properties of Pneumatic Actuator link

2.12.1 Pneumatic Actuator property window

The property window of the Pneumatic Actuator link is the same of basic links , but also adds the following properties in the 'Custom' tab:

Valve A
Opening v_a of the pneumatic valve A, as a proportional 3/2 pneumatic valve.
Default: $v_a = 0.5$, close.
Opening $v_a < 0.5$, open for intake. (max intake for 0.0)
Opening $v_a > 0.5$, open for exhaust. (max exhaust for 1.0)

Valve B
Opening v_b of the pneumatic valve B, as a proportional 3/2 pneumatic valve.
Default: $v_b = 0.5$, close.
Opening $v_b < 0.5$, open for intake. (max intake for 0.0)
Opening $v_b > 0.5$, open for exhaust. (max exhaust for 1.0)

PmA in-pressure
Pressure p_mA of intake for valve A. Measuring units: $[Pa]$. Note: $1atm \simeq 100000Pa$.
Usually $p_mA = 600000[Pa]$ for most industrial pneumatic applications.

PmB in-pressure
Pressure p_mB of intake for valve B. Measuring units: $[Pa]$. Note: $1atm \simeq 100000Pa$
Usually $p_mB = 600000[Pa]$ for most industrial pneumatic applications.

Ps air-pressure
Pressure p_S of external air, where exhausts will go. Measuring units: $[Pa]$. Note: $1atm \simeq 100000Pa$. Usually $p_S = 100000[Pa]$ on our planet.

pA chamber A
Pressure p_A in chamber A, read only. Measuring units: $[Pa]$. Note: $1atm \simeq 100000Pa$.

pB chamber B
Pressure p_A in chamber B, read only. Measuring units: $[Pa]$. Note: $1atm \simeq 100000Pa$.

F force
Instantaneous force F caused by pneumatic forces on the stroke rod. This is applied to the two markers. Measuring units: $[N]$.

O offset
Distance between the two markers (actuator ends) for pneumatic cylinder completely unextended.

L max stroke
Max stroke length (max clearance of the cylinder).

R radius
Cylinder radius, in meters [m].

2.13 Controls

You can use the Controls object to add Javascript programs to your mechanical systems. In this way, you can use the scripting features of Chrono::R3D in order to create automatic devices, digital controllers, etc.

The Controls object has **no geometry** (you can put it wherever you want in the hierarchy of your system, but it won't be visible in 3D views). Adding a Controls object to a device is a bit like adding an electronic digital controller to a machine: this controller will do the things that you write in the Javascript programs.

The scripts contained in the Controls object can be executed for different **events**, for example at each time step, or each time the simulation starts, or stops, etc.

The scripts contained in the Controls object can be executed for **different events**, for example at each time step, or each time the simulation starts.

The Controls object also implements a feature that allow you to **create 'panels'** (graphical user interfaces). Hence the user can interact with buttons and sliders of these 'panels' also during simulation playback, in order to control variables of the mechanical system in 'real time'.
For example, you can create a panel with two sliders, where each slider changes the opening of a valve of a pneumatic cylinder . This will allow the user to play with the pneumatic actuator ini real-time, while the simulation plays, without the need to open the property window of the actuator.

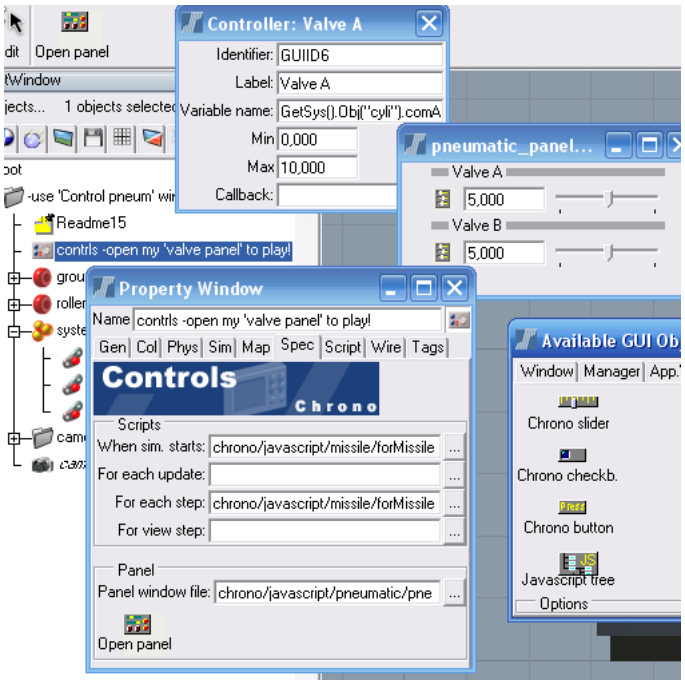


Figure 2.29 Custom properties of Controls object

To create a panel:

- Open a new empty window in Realsoft3D environment (just use the menu `Customization/ Available GUI Objects`, go to tab: `Window`, press button: `Float`, open a window with a name like 'Missile panel').
- Switch to tab `Misc` and drag'n drop the Chrono::R3D sliders or buttons into the empty window. Add also GUI items if needed (packers, splitters, etc.)
- For each slider or button, there is a small icon on the left, which allow you to open a window for gadget configuration. For example, rename a label of a slider to 'Thrust', set Variable name to `my_thrust`. Then, each time the user moves the slider, the Javascript variable `my_thrust` will get the proper value.
- Close the window. Note now that you will find the window 'Missile panel' in menu: Windows. This means that the window has been saved in the directory realsoft/windows/. You can move this file in another directory, for example in tmp/missile/. Then, in Controls object, set the 'Panel window file' field to tmp/missile/Missile panel, so that you can open this window when clicking on the 'Open panel' button of Controls, when you like.

The execution of large scripts in Javascript may affect the simulation speed.

Remember that after some seconds of continuous execution of Javascript programs, the environment may 'freeze' for a fraction of second: this happens because Javascript is an interpreted language and sometime its engine needs to clear the CG (Garbage Collector), that may be a CPU intensive operation. This happens somewhat randomly, after periods of seconds.
This must be remembered if you want to perform realtime or quasi-realtime simulations, where the small pauses caused by the GC clearing may affect the 'smoothness' of realtime execution.

2.13.1 Controls property window

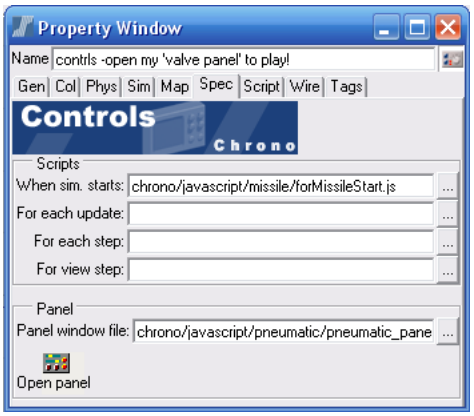


Figure 2.30 Custom properties of Controls object

When sim. starts
Name of a Javascript file to be executed each time the simulation starts.
This is a nice place to put programs which initialize variables and objects which will be used during the entire simulation: for example if you intend to use a Chrono::R3D Javascript PID

object to make devices with feedback, it is wise to set up and initialize it with this ‘When sim. starts’ script, before other scripts like ‘For each step’ will use it.

For each update
Name of a Javascript file to be executed each time the system state gets an update. This may happen many times during a single time step.

For each step
Name of a Javascript file to be executed each time step. This happens each time the integration algorithm of the simulation engine goes forward one time step.
Do not assume that time steps are constant, since some integration schemes may feature variable-step.

This is the most appropriate place to put programs which must update variables of your mechanisms as function of the time (for example you can put here a script which performs the feedback control of a pneumatic device, or a script which simulates automatic pilot for an airplane).

For view step
Name of a Javascript file to be executed each time step of the Realsoft3D interface. This happens each time the 3D view is refreshed because of a time step in Realsoft3D animation.

Remember that a single Realsoft3D ‘view step’ may contain multiple ‘integration time steps’ (see above). For this reason, it is wise to put here your scripts which need updating at constant time intervals (for example, print values on screen or save results on a file for each 25th of a second). Also, do not put here the scripts which need to be updated/executed for every small time step, like in cases of PID feedback controllers (put them into the ‘for each step’, see above). However, if the script does not requires strict updating for time changes, and less frequent updating is enough, this is a good place to put it (example: scripts which perform Artificial Intelligence tasks for decisional processes in automatic devices)

Panel window file
Here you can set the name of a file containing a Realsoft3D floating window with some GUI controls, for realtime interaction with Javascript variables (and realtime control of Chrono::R3D objects).
The ‘window file’ is a file like the ones in the directory realsoft/windows/.
In general, you can create a ‘panel window file’ by using the instructions above .

Open panel
Open the ‘panel window’ whose file name is set above, with the ‘Panel window file’ field.
This button is available also on the ‘select tool window’ when you select a Controls object, so you do not need to open the Property window to open the panel.

It is not necessary that there is a ‘panel’ for each Control object, but if you wish to modify variables in realtime (such as the steering and the braking of a simulated car) this is the appropriate way to go. For example, in case of a car, you can create a ‘panel’ with a GUI slider where the user can change the value of the `my_braking` Javascript variable. Then, write a Javascript program which use the variable `my_braking` to set the braking effect for the four brakes of the wheel, and use it in the ‘For each step’ field.

2.14 Probe

You can use the Probe object to record variables of simulations, for example the plots of accelerations and reaction forces in your mechanisms.

The Probe objects must be put **into the sub level** of the objects whose variables must be recorded.
For example, if you are interested in monitoring the value of the reaction force of a spring you should have the Probe contained in the spring object.

The Probe object can work in **different modes**: it can record variables into 2D plots, or draw trajectories, or plot vectors in 3D space, etc.

The variable to be monitored can be fetched by means of **Javascript syntax**. In fact each Chrono::R3D object has a tree of Javascript properties: these are described in scripting reference and are used to access the variables.

2.14.1 Probe property window

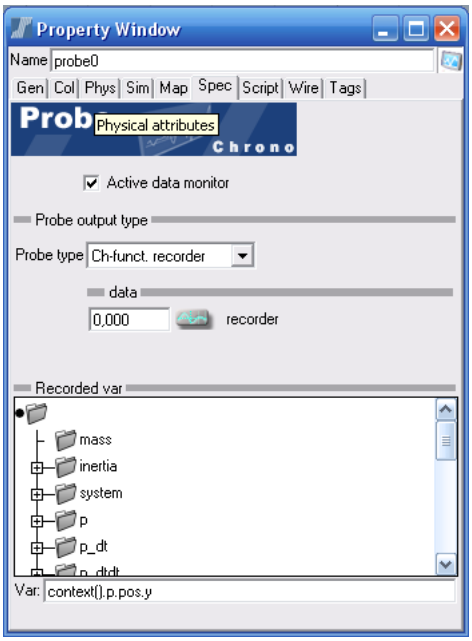


Figure 2.31 Custom properties of Probe object

Active data monitor
When checked, the probe object is working (if turned off, no data is monitored and saved).

Probe output type
Choose the recording mode of the Probe object. In fact, once the data is fetched from the parent 'monitored' object, this data can be either plotted, or stored in a ChFunction , or drawn as a 3D vector, etc, depending on this selector:

- **Ch-function recorder:** stores the recorded data in a ChFunction (the editor and plotting of that function is available in the 'data' field below). This is the default mode of the Probe object.
  Note that the stored data must be a scalar value (no vector, no matrix, etc.)
- **3D trajectory:** draws the fetched vector as a trajectory, whose coordinate $P(t)$ is the value of the vector data.
  Usually you use this mode when fetching the absolute position of a marker or a body , for example with Var=`context().p.pos`, so that you get the plotting of the trajectory in 3D space during motion.
  Note that the fetched data must be a vector value (example: position).
  Also note that the plotted curve will be put in the System level of your hierarchy, where a sub-level with name 'plotted curves' will be automatically created.
- **3D vector plot:** draws the fetched vector as a three-dimensional symbol shaped like an arrow, which represents a vector in 3d space.
  Usually you use this mode to show instantaneous speeds and accelerations of markers and bodies , or to show reactions in links .
  Note that the fetched data must be a vector value (example: forces, speeds).
  Plotted vectors have specific settings of thickness and length scaling, see later.
- **Set JS variable:** the fetched data is simply stored into another Javascript variable. For example, you can store the fetched value of `context().p.pos` into your variable `my_pos`, which has a simpler name and which will may be used later in your scripts.
- **Digit:** simply shows the values of fetched data into numeric fields of the Probe property window. Fetched data can be either vector or scalar.
- **R3D curve:** the fetched data is stored in a Realsoft3D curve object, which can be shown and edited below. Note: Realsoft3D curve object performs data recording significantly slower than with the ChFunction recording (see the 'Ch-function' mode above).
  Fetched data can be either vector or scalar.
- **3D snapshots:** makes copies of a shape for each step (creating an effect of multiple snapshots of a shape at equal time intervals). To use this mode, the probe object should contain two link objects, the first pointing to the shape to be copied, and the other pointing to the level where the copies of the shape will be stored (note, here we mean the R3D link objects, in menu "tools/Creation/Create Link", not the Chrono::R3D links).

Recorded var:
This tree browser allow you to get the correct name of the Javascript variable that you want to fetch from the parent object.
For example, if the Probe object is inside the level of a body object, this tree will show all the Javascript properties which can be fetched from the rigid body: by selecting 'p' you would have Probe fetching the coordinate system of the rigid body, and expanding the 'p' subtree you could select either the translation part 'pos' (the position), or the rotational part 'rot' of that coordinate system, and so on. As long as you navigate this property browser, the full name of the variable will be automatically written for you in the Var: field below.

Var:
Javascript name of the variable that you want to fetch from the parent object. Note: to avoid typos and mistakes, you can use the 'Recorded var:' property browser described above, so that it will fill this field automatically.

The Var: field can accept also complete Javascript statements and mathematical operations.
For example, if you want to fetch the kinetic energy of a rigid body, $Ec = (1/2)v^2m$*, you may write Var:* `1/2 *(pow(context().p_dt.pos.Vlength(),2) * context().mass)`

If you want to check that you are not doing syntax errors in the Var: field, you must open the scripting window in Javascript mode, (Use menu window/scripting) and look what happens when you refresh everything (for instance, by performing some animation steps). In case of error, the wrong statement and the syntax error will be reported in that scripting shell.

If you use the property browser (the field 'Recorded var:') you will soon discover that most Javascript statements for the choosen property will start with the `context()` *command. In fact, this command is used to return the Javascript object of the parent object (the object which contains the probe). For example, the* `context()` *command, executed by a probe which is inside a marker , will return a* `Marker` *Javascript object (that's why you could write* `context().p.pos.x` *to get the X position of that marker).*

3 Tools

When Chrono::R3D is installed properly on your computer, you should find a new tab 'Chrono' among the tabs for the Realsoft tools. Here you have direct access to many interactive tools for creating and manipulating Chrono::R3D objects.

This chapter will describe the features of the tools, however you can also see the tutorial manual for practical examples on how to use them.

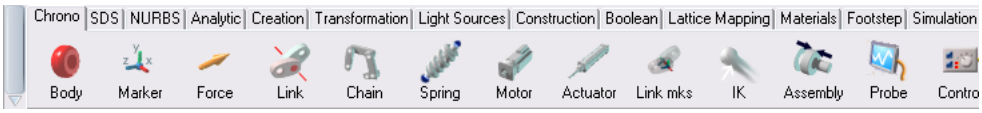


Figure 3.1 A typical configuration of a toolbar containing most Chrono::R3D tools.

Note that you can activate these tools not only by clicking on their buttons, but also using the menu 'Tools / Chrono Tools..'

If needed, you can also create your own toolbar by editing the Realsoft environment (use menu 'Customization / Available Objects Window', then go to the 'Tools' tab, then expand 'Chrono tools': all the icons can be placed in your interface by drag & drop). If you do this, remember to save the environment.

3.1 Body tool

This tool can be used to create rigid bodies with mouse input (just the basic coordinate system of an empty body will be created) or by enclosing the selected geometries into a new body.

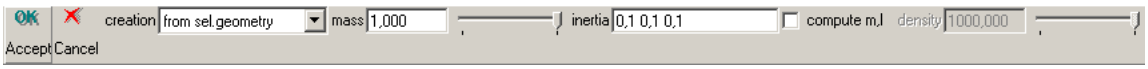


Figure 3.2 Tool bar for the 'body' tool

USAGE

MODE 1: Click with mouse cursor on the screen: an empty body will be created, with COG at the position where you clicked on the input plane.

MODE 2: Before starting the tool, select some already-created geometry, for example a cube or a polyhedron, then use this tool and click on Accept or click with cursor in whatever place of the 3d view: the selected geometry will be enclosed into a body level, with COG automatically placed where necessary.

TOOL SETTINGS

creation
Creation mode: from scratch or from selected geometry. **mass**

Total mass of rigid body $[kg]$. Default is 1, but there is also an option (the the 'compute m,I' button) which computes the mass of the body from its geometry and density. Later, in property window of the body object, you can still use the button 'Recompute mass, inertia, COG' to recompute mass from given geometry and density in case you changed its shape. Note: this mass value has nothing to do with the 'mass' value of default Realsoft3D simulation system (see property window/Phys). **inertia**

Values of XX inertia $[kgm^2]$ for body rotation. Computed about bodys x,y,z axes. They represent the diagonal values $I_{xx}I_{yy}I_{zz}$ of the typical I_{ij} 3x3 inertia tensor. These values can be automatically computed by using the 'compute m,I' button, given body density and geometry (otherwise a default value is used).

$$I_{ij} = \begin{Bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{Bmatrix} \tag{3.1}$$

compute m,I

Use this button to compute automatically the mass of the body, depending on its shape (and density, see 'Density' field). Also, computes the entire inertia tensor and moves the COG (Center of Gravity) in the correct position.

Some notes:

- *the selected rigid body geometry (or level) should include some solid geometric objects (ex. cubes, cylinders), otherwise an error will occur (zero mass is not admitted!).*
- *the geometry of the rigid body can be made of compound shapes, and freeform surfaces or SDS shapes can be used as well, but all surfaces must be closed otherwise resulting mass and inertia will be wrong.*
- *this function uses a low-order quadrature method (the space is volume-sampled to compute mass, inertia, etc.). This sampling has a finite precision, so maybe that the results are not 100% exact, you can expect some approximation if body shape has many thin details.*
- *the volume sampling of this feature can take some time for computing results, even some seconds with slow computers and complex shapes.*
- *center of mass (COG) is placed where needed, but axis are not aligned to main inertia axes (that is, body handles are translated to COG, but not rotated).*
- *rather than adding many details (ex. screw, holes) to bodys shape, it is more straightforward and time-saving to create a rough approximation of the full shape with few 3d objects (cubes, spheres): the inaccuracy can be negligible.*

density
In order to recompute mass etc., the 'compute m,I' feature needs the density of the rigid body: hence in this field you can set the density. CHtip Note that the density is expressed in $[kg/m^3]$ (assuming that you are using the default measuring system, 1 Realsoft length unit = 1 meter), so this value can be quite high. Some examples: density of water: $1000[kg/m^3]$, density of wood: $500-1000[kg/m^3]$, density of steel: $7000[kg/m^3]$, density of light alloys: $3000 - 5000[kg/m^3]$.

3.2 Marker tool

This tool can be used to create markers with direct mouse input. Markers are auxiliary coordinates which are used by Chrono::R3D as references for links etc.

USAGE

MODE 1: Select an already-created rigid body , then activate this tool and click in the 3D view, where you want the marker to be created. That marker will belong to the selected body.

MODE 2: Just click on the 3D view. A marker will be created (but remember that it is not contained in a rigid body , so it has little meaning...)

Remember that markers have a meaning only if they are contained in the sub levels of body objects, because they move with them.

When creating links , markers are automatically created with the proper link tool , so you will seldom create them by hand. However you may still need this marker tool when you have to add a marker to a body in a specific position, for example as a reference for recording accelerations or trajectories with a probe .

3.3 Force tool

This tool can be used to create forces with mouse input. Forces should stay into the rigid bodies objects which they affect.

Figure 3.3 Tool bar for the 'force' tool

USAGE

MODE 1: Select an already-created rigid body , then activate this tool and click twice in the 3D view: first click to set where you want the force to be created, second click to indicate the direction. That force will belong to the selected body.

MODE 2: Just click twice on the 3D view, with nothing selected. A force will be created (but remember that it is not contained in a rigid body , so it has little meaning...)

Remember that forces have a meaning only if they are contained in the sub levels of body objects, because they affect the bodies which contain them.

TOOL SETTINGS

force type
Choose if it works as a force or as a torque.

frame
Choose if the force moves together with the body or remains fixed to the world reference.

align
Choose if the force rotates together with the body or remains aligned to the world reference.

strength
Modulus of the created force.

3.4 Link tool

This tool can be used to create links with mouse input. Links are mechanical constraints between rigid bodies , in detail they connect two markers belonging to these bodies.

The Link tool is one of the tools you will use most frequently. By creating links between bodies, you can build complex mechanisms. You can constraint the relative motion of two parts as if in real life you build hinges, joints etc.

By default, the Link Tool puts the created links into a System object (if no System object is found, a new one it is created). In fact Link objects can be placed wherever in 3D object hierarchy, but a suggested place is the sub level of a System object.

The Link Tool creates a link between two selected bodies , but you do not need to have rigid bodies in your scene before using the tool, because if you select two plain Realsoft objects, the tool will automatically create the corresponding bodies.

This tool can work in different ways: it is smart enough to understand which is the proper behavior depending on what you select on the screen. Usually it requires four selection steps (shown with the 'Step' icons), the first two steps are used to select the two objects to connect, and the other two are most often used to choose the place where to create the constraint.

The first two steps (mouse selection of the parts to be connected) can be skipped if you start the Link Tool with two objects already selected, for example by using the select window.

The last two steps ask where you want to put the two marker references (the first is the 'slave' and belongs to the first selected body, the other is the 'master' and belongs to the second selected body).

By performing a fast double-click on the 3D view, the second marker is created exactly at the same position of the first one.

USAGE

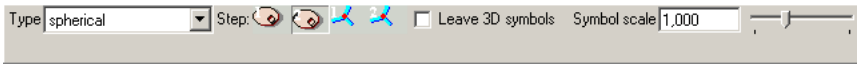


Figure 3.4 Tool bar for the 'link' tool

MODE 1: Start tool with two rigid bodies already selected (if you do not have bodies, just select two plain Realsoft objects, and bodies will be created for you). Now just double-click on the screen to create the constraint in that position.

MODE 2: Start tool with no objects selected. Then use mouse cursor, clicking in 3D view) to select the first body to constraint, then the second (if you do not have bodies, just select two plain Realsoft objects, and bodies will be created for you). Now just double-click on the screen to create the constraint in that position.

MODE 3: Say you want to create a constraint between a part and the absolute reference. A way to accomplish this task is to select just one body, then -instead of selecting the second one- just click on an empty area of the 3D view (then a 'grounded' object will be used as second body; it will created if none is found). For selecting the place of the constraint, complete the last two steps by double clicking on the desired position.

OTHER MODES : There are many other ways to use this tool. For example, one variation could be this: if your two parts are drawn apart, as dismounted, you can complete the two last steps by clicking in two different 3D positions (the joint position on first body, then the joint position on the second body). Etc.

Feel free to change the Type setting while you are choosing the constraint position: you will see a 3D preview of the link shape which tells you intuitively how the constraint works.

TOOL SETTINGS

Type
Choose the constraint type. See link reference for details.

Step
This ribbon shows the steps of link creation that you are performing. (By clicking on the icons, you can also rewind the process, as going back in creation steps).

Leave 3D symbols
If checked, the 3D symbols that you see under your mouse during creation are also copied in the final scene. (Half symbol will be put in the sublevel of the master marker, and half symbol will be put in the sublevel of the slave marker, so that these shapes will move if markers are moved).

Symbol scale
Size of the 3D symbols following the mouse cursor during creation.

Figure 3.5 Tool bar for the 'Spring' tool

3.5 Spring tool

This tool can be used to create springs and dampers with mouse input.

The Spring tool is very similar to the Link tool , so read its documentation for more details.

USAGE

[Look at the Link tool documentation: its behavior is the same].

TOOL SETTINGS

[Look at the Link tool documentation: most settings are the same].

Stiffness

Stiffness value of the spring. Non-linear stiffness and other advanced settings can be accessed later, in the Property Window of the spring object.

Damping

Damping value of the damper. Non-linear damping and other advanced settings can be accessed later, in the Property Window of the spring object.

3.6 Actuator tool

This tool can be used to create a linear actuator between two bodies, with mouse input.

The Actuator tool is very similar to the Link tool , so read its documentation for more details.

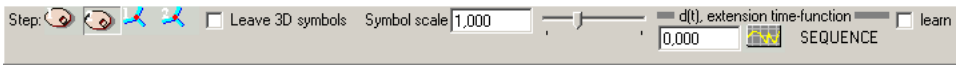


Figure 3.6 Tool bar for the 'Actuator' tool

USAGE

[Look at the Link tool documentation: its behavior is the same].

TOOL SETTINGS

[Look at the Link tool documentation: most settings are the same].

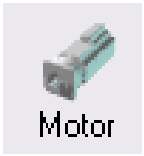
d(t) extension time-function

Length of the actuator as a function of time. Press this button to open the ChFunction editor to set the motion law. Note that total length is the sum of this variable extension and

a fixed offset (for zero extension) which corresponds to the lengths of the actuator when it is just created.

Learn
If checked, the actuator is passive and does not impose the length. However, if something else is moving the mechanism (ex: other motors) this actuator, being in learn mode, simply records its extension in the d(t) ChFunction ; it may be useful later.

3.7 Motor tool



This tool can be used to create a motor between two bodies, with mouse input.
The Motor tool is very similar to the Link tool , so read its documentation for more details.

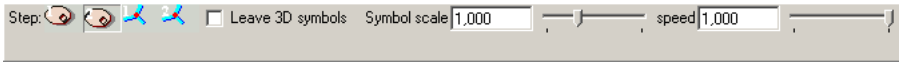


Figure 3.7 Tool bar for the 'Motor' tool

USAGE
[Look at the Link tool documentation: its behavior is the same].

TOOL SETTINGS
[Look at the Link tool documentation: most settings are the same].

speed
Speed [rad/s] of the motor, in radians per second. Note that the motor is created in 'impose speed' mode by default, but you can change later this mode by opening the property window of the motor (for example you can put it into 'impose rotation' mode or 'impose torque' mode).

3.8 Link markers tool

This tool can be used to create a link between two markers belonging to two distinct bodies.
Note that, usually, you can use the more powerful Link tool for interactive creation of constraints (it will automatically create the two markers for the two constrained bodies), but maybe that you already have two markers in the bodies: in such a case, you can use this 'Link mks' tool to join them.

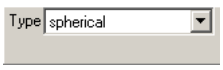


Figure 3.8 Tool bar for the 'Link mks' tool

USAGE

MODE 1: select two markers , belonging to two distinct rigid bodies. Activate the Link mks tool and use the toolbar to choose the type of link to create, then click in whatever area of the view: the link connecting the two markers will be created. Note that the 2nd selected marker will be used as the 'master' reference for the link.

MODE 2: select a marker belonging to a rigid body. Activate the Link mks tool and use the toolbar to choose the type of link to create, then click in whatever area of the view: a link connecting the markers and the ground reference will be created (a marker for the ground will be automatically created). Note that the marker in ground body will be used as 'master' reference for the link.

TOOL SETTINGS

Type
Choose the constraint type. See link reference for details.

3.9 Chain tool

This tool can be used to create a sequence of rigid bodies connected by links with few efforts. In this way it is easy to define kinematic chains like skeletons and articulated mechanisms just by entering the position of the joints.

The 'bones' of the skeletons are automatically created as 'beam-like' rigid bodies , and joints are automatically created as links of revolute type (or other types, depending on what type you select while entering the joint position).

Figure 3.9 Tool bar for the 'Chain' tool

USAGE

Select the tool and left-mouse-click on the 3d view to create the articulated chain, one joint after the other. When you want to terminate the chain, use the right mouse button to open the popup menu and choose 'Accept'.

If you enter a wrong position for a joint, just use the 'Back' option in the popup menu to rewind the sequence of entered joints.

If you start the tool with a rigid body selected, the first joint of the chain will be connected to that rigid body instead that to the ground.

If you start the tool with two bodies selected, the first joint of the chain will be connected to the first body, and the last joint will be connected to the second body.

All links are created on the current input plane. Turn on the grid drawing to understand better where is the input plane, especially if working in perspective mode.

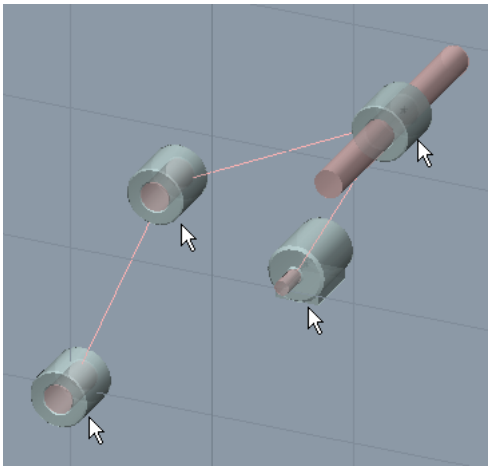


Figure 3.10 Example of chain creation in four steps.

TOOL SETTINGS

Joint type
Choose the constraint type. See link reference for details about the behavior of the many constraints.

Ground start
When this is turned ON, the first joint will be connected to the ground reference (otherwise the beginning of the chain will float). Default: ON.

Ground end
When this is turned ON, the last joint will be connected to the ground reference (otherwise the end of the chain will float). Default: OFF.

Linear mass
Value of linear mass for the rods, in [kg]/[m]. In fact the rods between the joints are bodies , so they need a reasonable value for mass and inertia in case of dynamical simulation.

Leave 3D symbol
If turned ON, the three dimensional symbols used for the interactive creation will be left into the created bodies.

Ground end
Scale for the three dimensional symbols used for the interactive creation.

3.10 IK tool

This tool can be used to interactively manipulate a mechanism, performing IK (Interactive Kinematics) by moving the mouse in the 3D view.

This Chrono::R3D 'IK tool' has nothing to do with the default IK (Inverse Kinematics) tool of Realsoft3D. In fact you must use this tool to move mechanisms made of Chrono::R3D parts, that is rigid bodies .

You can think at this tool as an 'invisible hand' which picks the mechanism and moves its parts.

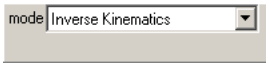


Figure 3.11 Tool bar for the 'IK' tool

USAGE

1- Select a body , or some other object contained in a body.
2- Activate the IK tool.
3- Click once in 3D view, to choose where is the picked point (on the input plane).
4- Move the mouse to move the mechanism -it will follow the cursor, if motion is possible in that direction-
5- Click again to accept.

Otherwise you can start without selected bodies, so the tool will require that you select a body with a first click.

If you do not like the result of the motion, you can use 'Cancel' instead of 'Accept'. Otherwise you can perform an Undo,

Do not expect that the selected body will follow exactly the mouse cursor: sometimes the motion is much 'slower', as if there were an elastic spring between the mouse and the moved parts. This may be more noticeable when you try to move a part in a direction which is constrained, or almost constrained, or when you are moving the cursor on an input plane which is in a weird position (much far from the manipulated object, or badly aligned).

When selecting the picked point to move, remember that it will be projected on the input plane: it is wise to have 'grid drawing' turned on, in order to understand where is the input plane, especially when performing modelling in perspective mode. Also, for the same mechanical part, not all the points are good choices for performing the interactive kinematics: some positions are better 'handles' for some movements.

TOOL SETTINGS

Mode
Interactor mode. Default is the Interactive Kinematics mode.

3.11 Assembly tool

This tool can be used to perform a 'mechanism assembly' operation on the system. That is, if some links are not satisfied because someone previously misplaced the mechanical parts, this button will keep the parts together, satisfying all the constraints (if possible).

In some cases, there are mechanisms (with impossible, difficult or ill-placed constraints) which cannot be assembled. In such cases, the Assembly tool can fail, and a 'WARNING ..' message will appear on the bottom of the screen, in the status bar.

USAGE

Press the tool button and see how the mechanism is assembled (of course, you can see the effect only if some parts are out of place thus opening the link constraints).

3.12 Probe tool

This tool can be used to create a probe object

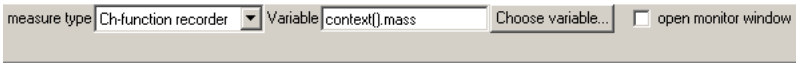


Figure 3.12 Tool bar for the 'Probe' tool

USAGE

Select a Chrono::R3D object, for instance a marker , or a body , or a link . Then, press the 'Choose variable' button and choose some item in the window which pops up (showing the available Chrono::R3D variables for the selected object). Then press 'Accept' (or click on whatever point of the 3d view), and a probe object is soon created and dropped inside the object whose variable must be recorded.

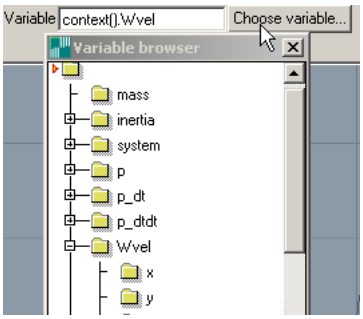


Figure 3.13 Example of the Choose Variable wizard.

The probe object can work in different modes: read the reference pages of this object to understand how to draw trajectories, plot vectors, etc.

In 'Ch-Function recorder' mode, only scalar (floating point) variables are accepted. When selecting a Chrono::R3D javascript variable from the tree list, note that you can expand (with the '+' buttons) the substructures in order to access other items. For example, when dealing with a body , you can expand the 'p' structure (coodrinate system for position), the select the 'pos' (position vector), then 'x'. In this way you record the x position of that body, that is the 'context().p.pos.x' variable. You cannot record directly a vector (like 'context().p.pos') because it is not a scalar.

In '3D trajectory' mode, only vector variables are accepted. For example, the 'context().p.pos' variable can be used in this mode to plot the trajectory of the center of mass of a body.

In '3D vector' mode, only vector variables are accepted. For example, the 'context().p_dt.pos' variable can be used in this mode to plot the vector of the speed of the body.

In '3D snapshots' mode, the selected object will be copied many times (one for each animation frame), then causing a 'onion skin effect'. In this mode, whatever object can be probed, most often they can be lines or shapes.

Note that the 'Choose variable' button wizard always puts a 'context()...' prefix in the Variable field. This is because the 'context()' keyword returns the Javascript reference to the object which is probed.

Read the Javascript manual to understand the naming of the Javascript variables of the Chrono::R3D objects. You can also record functions like 'context().p.pos.x / sin(2)'.

TOOL SETTINGS

Measure type
Type of probe object. See probe reference for details.

Variable
Expression for Chrono::R3D -Javascript variable to be recorded. See probe reference for details, and read also the Javascript manual for learning the Javascript syntax and available variables.

Choose variable..
Opens a wizard which helps you in choosing the variable to be recorded, by selecting it from a popup-window with the variable tree of the currently selected object. Note that the selected variable is automatically written in the 'Variable' field at the left.

Open monitor window
If ON, the monitor window is opened as soon as you accept the tool (only for 'Ch-Function recording' mode).

3.13 Controls tool

This tool can be used to create a control object in the current level. After you created the control object, open its property window to access its features.

USAGE

Press the tool button to create the control object.

3.14 Cam tool

This tool can be used to create the shape of a cam. (This is only a CAD tool, and the created cam does not work together with other Chrono::R3D parts).

The created cam profile can be used to extrude a 3D shape of a cam, for example to build a shape for a collision-detection project. Otherwise the cam profile can be exported in DXF format to a laser-cutting machine, EDM cutting machine, or to other NC milling machines -in order to build a true cam made of steel-.

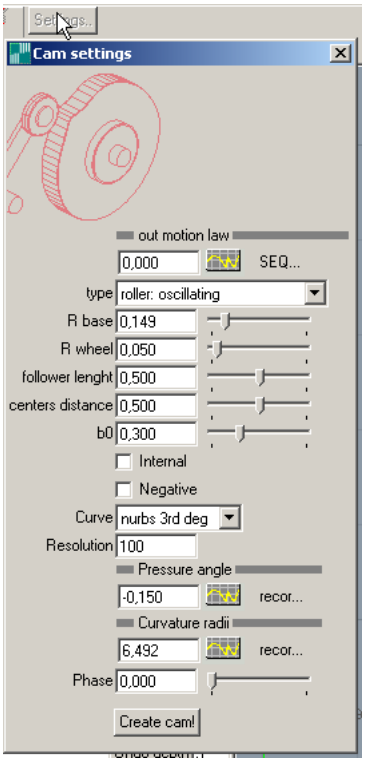


Figure 3.14 Settings for the 'Cam' tool

USAGE

Select the tool. Press the 'Settings..' button: the big settings window will appear. Play with the cam settings and see the cam shape changing in real-time in the 3D view. Move the 'Phase' slider to see the cam rotating in the 3D view. Use 'Accept' to create the cam profile, as a Nurbs line.

TOOL SETTINGS

Out motion law
This is the most important setting: press the button to open the ChFunction editor where you can define the output motion law. For example, if the cam is in 'roller: oscillating' mode, this function defines the rotation of the follower (in radians) as a function of the

cam rotation (in radians, 0 ... 6.28). In 'flat: sliding' mode, this function defines the displacement of the slider (in meters) as a function of cam rotation, etc.

Use the popup menu 'Load .chf' in the ChFunction editor to load some example functions, like 'example_cam_1.chf'.

Type
Types of cams and follower combinations. Test the different settings and look at the effect in the realtime display.

R base
Base radius of the cam (radius of the cam for zero follower displacement)

R wheel
If in 'roller:..' mode, this sets the radius of the wheel of the follower.

Follower length
Length of the follower (if in 'oscillating' mode).

Centers distance
Distance of the center of rotation of the follower from the center of the cam (if in 'oscillating' mode).

b0
Initial rotation of the follower, for zero displacement (if in 'oscillating' mode).

Internal
Turn this ON if an internal cam is desired.

Negative
Turn this ON if a negative cam is desired.

Curve
Type of curve used to approximate the cam. Nurbs curves, with higher degrees, are smoother. Polygonal curve is supported in most cases of conversions into other CAD formats.

Resolution
Number of points in the curve which approximates the profile of the cam. The higher, the more precise the approximation. For a polygonal curve, 360 points can be enough precise for obtaining a DXF profile to be machined with CN milling machines. The more points, the slower the refresh while tuning different parameters.

Pressure angle
This ChFunction automatically records and stores the value of pressure angle, as a function of cam rotation. You can keep this editor open to see if the graph of the pressure angle grows too high while changing the design parameters.

Curvature radii
This ChFunction automatically records and stores the value of curvature radii of the cam surface, as a function of cam rotation. You can keep this editor open to see if the graph of the curvature becomes critical while changing the design parameters.

Phase
Move this slider to see the cam which rotates and the follower which moves accordingly.

3.15 Optimization

This is an advanced engineering tool, which can be used for designing and optimizing robotic devices. Available only with Chrono::R3D Extreme license. For advanced users.

This tool can be accessed ONLY in this way: use the menu 'Windows/ Tool window', then go to the 'Robot [J]' tab.

The *Optimization* tool can be used to perform optimizations of mechanical structures. In detail, it tries to find the value of the 'optimization variables' which minimize an 'objective function'. In general, if we say that $\mathbf{q}$ is the vector of the optimization variables and $F(\mathbf{q})$ is the scalar function to be minimized, then this tool finds $\mathbf{q}_{opt}$ for minimizing $F(\mathbf{q})$.

For example, in a mechanical application, the objective function to be minimized could be the force reaction in a joint of a robot, while the variables could be (for example) the lengths of some struts. The user defined optimization function should return the reaction in the joint as a function of strut lengths.

Note that this tool features a two-stages hybrid optimizer, which is able to perform also global (i.e. not only local) optimization over a state range by means of a genetic algorithm.

In few words, a 'local' optimizer just modifies the initial configuration in order to approach the nearest optimum, but it is not guaranteed to be the best of all the field, while a global optimizer tries to find the absolute optimum, even for radical changes in initial state.

The genetic algorithm of this tool features modern synthetic-evolutive operators. It simulates the evolution of a population of synthetic individuals, each with a digital genomic data. Generation after generation, only the individuals whose features provide a good fitness can survive and can mate and make children. When the genomic data encodes the optimization variables, and the fitness is computed by evaluating the optimization function, after many generations the population will converge to the global optimum. The convergence speed depends on how crowded the initial population is (the more individuals, the more precise the convergence, but the higher the computation time).

As a second optimization stage, you can rely on a local optimizer of *conjugate gradient* type, which performs a refinement of the best solution obtained with the first stage (the genetic algorithm).

Details on local and global optimization can be found in many textbooks, for example:
- Singiresu S.Rao, *Engineering optimization, theory and pratice*, Wiley Interscience, 1996.
- David E.Goldberg, *Genetic Algorithms*, Addison Wesley Longman, 1997.
etc.

Even if there are theorems saying that genetic methods converge to the optimum solution, you cannot expect that this really happens in a reasonable amount of time. Also, in most cases, for a limited amount of initial individuals, there may be a 'premature' convergence to a false global optimum (which may be a simple local optimum, even if acceptable). Increasing the initial number of individuals and increasing the number of simulated generations can reduce this risk, but the computational time may grow too much, so it is up to the user to choose a good compromise in terms of speed and precision.

The objective function $F(\mathbf{q})$ must be implemented as a Chrono::R3D -javascritpt function returning a number (depending on the actual value of the optimization variables $\mathbf{q}$). See the Javascript reference manual for more details on how to implement functions.

You can use this tool to perform optimizations of whatever objective function, not only related to Chrono::R3D -made mechanisms, but also for economic problems, molecular modeling, geometric problems, schoolwork, etc.

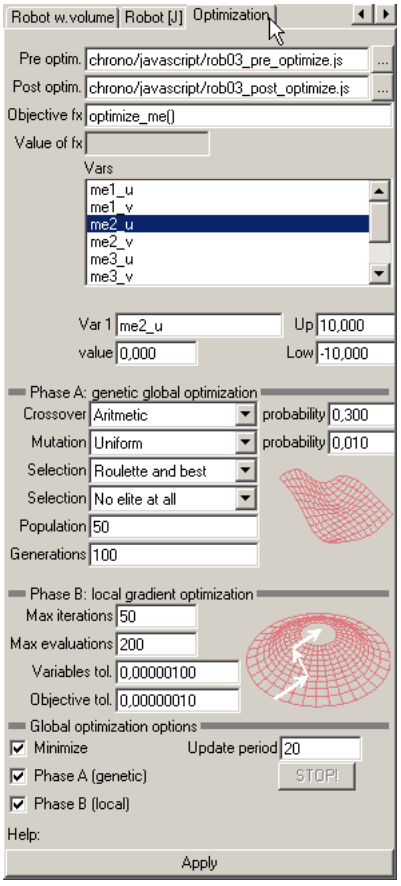


Figure 3.15 Settings for the 'Optimization' tool

USAGE

This tool requires that you define an objective function f returning a scalar value to be minimized (or maximized), that is $f = f(\mathbf{q})$, where $\mathbf{q}$ is a set of optimization variables. The function is a Javascript function, and variables are Javascript variables.

For example, suppose you have the two-variables function $f = (x_1 - 2)^2 + (x_2 - 3) + 4$, and you want to find the values of x_1 and x_2 which give the minimum f. In this simple case you could find the analytical solution by yourself (it is $x_1 = 2$, $x_2 = 3$), but here we can use the optimization tool of Chrono::R3D . In fact you can enter `pow((x1-2),2)+pow((x2-3),2)+4` in the "Objective fx" field, then use the popup menu in the "Vars" box to "Add variable" twice, adding two variables named `x1` and `x2` . Press the "Apply" button, and wait until the status bar tells that the genetic optimization has reached the optimum. Ok, you can select the variables in the "Vars" box, and see the optimal values of `x1` and `x2` (that is 2.0 and 3.0, as expected).

Of course this example can be developed in order to use a more complex objective function, for example in "Objective fx", instead of entering a formula, you can insert the

call to a very complex Javascript function like `optimize_me()`, maybe implemented in ”Pre optim.” Javascript file, which changes a mechanism according to actual value of the optimization variables, performs a simulation, then returns a value to be minimized (for example a displacement, a maximum stress, etc.).

In this way you can design mechanisms which fit best into some design goal (for example, find the best lengths for robot arms in order to get the highest speed, or find the best coordinates for anchor points of a part, in order to get the lowest forces in hinges, etc.)

TOOL SETTINGS

Pre optim.
Name of a script (Chrono::R3D Javascript) which is executed before starting the optimization. This is a convenient place to implement the function which must be optimized.

Post optim.
Name of a script (Chrono::R3D Javascript) which is executed soon after the optimization has completed. This can be useful to make reports, etc.

Objective fx
The function to be optimized. It must be a formula returning a scalar value (in Javascript syntax, like `x*x-2`), or a Javascript function returning a scalar value, like `optimize_me()`.

Value of fx
The actual value of the objective function. After the optimization has been performed, this is the best found value of the objective function, i.e. the minimum (or the maximum, depending on the settings).

Vars
This box contains the list of the optimization variables. You can insert or delete variables from this list by using the popup menu on this box.

Var
Name of the variable which is currently selected in the ”Vars” box. You can change it to whatever name you want (but do not make duplicate names, or names which cannot be Javascript variables because containing special characters like * or , or ; etc.)

value
Current value of the variable which is currently selected in the ”Vars” box. After optimization, this is the optimal value of the variable.

Up
Upper limit on the range the variable which is currently selected in the ”Vars” box. This is needed for the genetic optimization because at the beginning the first generation puts ’samples’ in a given range, and the up-low values are needed to specify such range. Be sure to provide up-low values which will contain the optimum which you believe to find. Setting very large up-low ranges will save you from guessing where the optimum may fall, but will slow the convergence of the method.

Low
Lower limit on the range the variable which is currently selected in the ”Vars” box. See notes for the ”Up” field.

Phase A- genetic global optimizer:

Crossover
Mating of two simulated individuals in the genetic optimization engine will cause a 'blending' of the chromosomes of father and mother for creating children. The way the genetic information is mixed can be set with this field: "Arithmetic" is a default method, "Blend" creates linear interpolation between fenotypes (best suited for mechanical optimizations), "Heuristic" tries to accelerate convergence but is less robust in case of non-smooth problems.

Mutation
Sets the probability (and type) of random genetic mutations. Mutation help preventing premature convergence and explores new zones for optimum search, but too high mutation slows down the optimization.

Selection
Sets the way the best individuals are selected to survive the generation, hence providing childhood.

Population
Amount of individuals in the simulated evolutive process. This is a very important setting, because the higher this value, the more precise the optimum finding. However, the computation time grows linearly with this value, because the objective function must be evaluated once for each individual, generation after generation. Also, remember that the more optimization variables, the higher this value should be.

Generations
Amount of generations in the simulated evolutive process. This is a very important setting, because the higher this value, the more precise the optimum finding. For simple problems, a value of 100 can be enough, for complex problems it may be needed 1000 or more. However, the computation time grows linearly with this value, because the objective function must be evaluated once for each individual, generation after generation.

Phase B- local gradient optimizer:

Max iterations
Maximum number steps in local optimum finding (when exceeding this, the process is stopped even if the optimum is not found.)

Max evaluations
Maximum number of objective function evaluations (when exceeding this, the process is stopped even if the optimum is not found.)

Variables tol.
The process is stopped if the variables do not change more that this amount, from step to step.

Objective tol.
The process is stopped if the objective function does not change more that this amount, from step to step.

General optimization options:

Minimize
If set ON, the optimizer will find the MINIMUM of the objective function (minimization). Otherwise, the MAXIMUM is found (maximization).

Phase A (genetic)
If ON, the global genetic optimizer is used for the first optimizing phase.

Phase B (local)
If ON, the local gradient optimizer is used for the second optimizing phase (local refinement).

Update period
Steps to perform before updating the status bar.

Apply
Press this button to START the optimization (it can be stopped with the STOP button).

4 ChFunctions

The ChFunctions components are modular **user functions** which can be easily modified thorough intuitive graphical interfaces. Such ChFunctions are extensively used in all the Chrono::R3D software. For example they are used to set time-dependent properties of objects (motion of markers), or non-linear properties of springs, etc.

By using the ChFunction editor you can change the type of function or set the parameters of the function. Also, a real-time plot of the function will be visible.

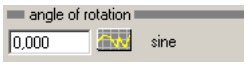


Figure 4.1 Press the button to edit the function parameter.

A ChFunction can be exported in many formats. By using the pop-up menu of the plotting window, you can access the Load/Save functions.
The default format is the binary format **.chf**, which can be used either for loading and saving.
You can also use ASCII load/save: in this case you will get Ascii files with a textual description of the function (which can be used in case you want to edit some parameters by hand with a text editor, for example).
By using the pop-up menu File/Import-export... you can get a window with some features for importing data from/to ascii tabulated data, from/to Matlab text files, etc.

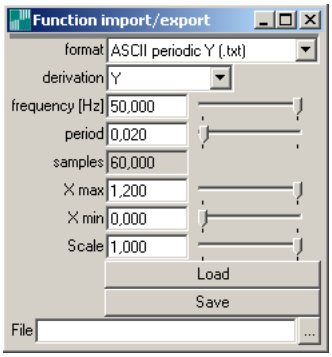


Figure 4.2 Function 'import/export window'.

By using the pop-up menu **Analysis..** of the plotting window, you can open a small window with some quick utilities for computing min/max values, integrals, etc.

By using the pop-up menu **Plot settings..** of the plotting window, you can open a window where you can change the name of the X or Y labels, or the graph title, and so on. Also, you can change some settings which affect the PostScript output, such as line colour and thickness, grid colour, font size, and so on. (To see the effects of changed colours etc. you can use the menu **File / PostScript preview..**)

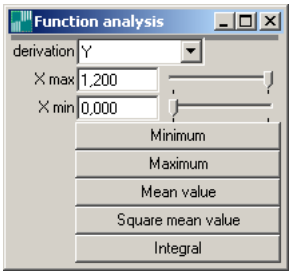


Figure 4.3 Function 'analysis window'.

By using the pop-up menu **File / Save as PostScript** you can save the plotting window in a .EPS file, wihch can be imported in most word processing programs. Note, however, that in some programs (es. MSWord) you will see an empty box instead of the picture -but the PostScript picture will be interpreted and printed nicely if you print the document with a printer which supports PostScript, such as in most laser printers-.

Some programs allow you to import .EPS files to generate PDF documents, such as the LaTEX software.

A free and powerful which allows the on-screen preview of PostScript files is GhostScript. It can be downloaded freely from many internet sites.

By using the pop-up menu **File / PostScript preview..** you can see the Postscript preview of the .eps plotting

The Postscript preview works only if the GhostScript software is installed, because the external GsView32 viewer is invoked for previewing purpose. GhostScript and its GsView32 viewer are freely downloadable from many internet sites.)

The properties of ChFunctions can be accessed and modified also through Javascript: see the documentation on ChFunctions javascript objects .

The plotting area can be **panned** *or* **zoomed** *simply by dragging the mouse with right button pressed and ALT key (for pan) or SHIFT key (zoom) pressed.*

There is a function for automatic zoom: the pop-up menu of the plotting area has the commands **zoom all (x)** *and* **zoom all (x-y)**.

Use the pop-up menu of the plotting area to show also the derivatives of the function (i.e. the speed and the acceleration, in case of motion laws), using the commands **Draw y, y', y"**.

Use the command **Hide controls** *of the pop-up menu of the plotting area to hide or show the controls of the function. If controls are hidden, only the plotting area will be visible (takes less space on the screen, if you just want to watch the graph).*

4.1 Constant

This ChFunction implements a constant function:
$y = C$

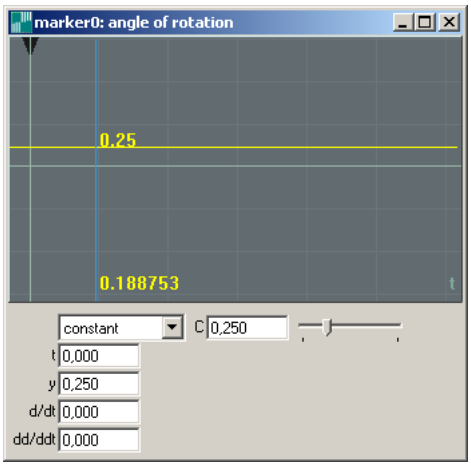


Figure 4.4 Custom properties of the Constant function

Parameters

C
Value of the constant function $y = C$.

4.2 Ramp

This ChFunction implements a linear function:
$y = Ax + B$

Parameters

y(0)
Value of the B constant in $y = Ax + B$.

dy/dt
Angular coefficient, that is the value of the A constant in $y = Ax + B$.

4.3 Sine

This ChFunction implements a trigonometric function:
$y = A \cdot sin(\omega x + \psi)$

Parameters

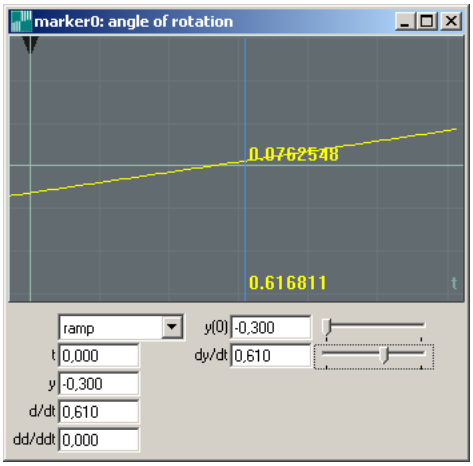


Figure 4.5 Custom properties of the Ramp function

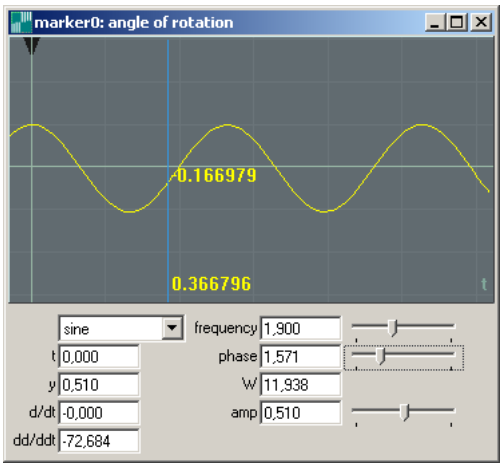


Figure 4.6 Custom properties of the Sine function

Frequency
Value of frequency f (periods P per second) in $[Hz]$, so that $w = f \cdot 2\pi$.

Phase
Value of phase ψ in radiants $[rad]$, for $y = A \cdot sin(\omega x + \psi)$

W
Pulsation ω in $[rad]/[s]$, function of frequency $w = f \cdot 2\pi$.

A
Amplitude A of function, for $y = A \cdot sin(\omega x + \psi)$.

4.4 Recorder

This ChFunction implements a data recorder:
$y = f < x_{recorded}[] >$

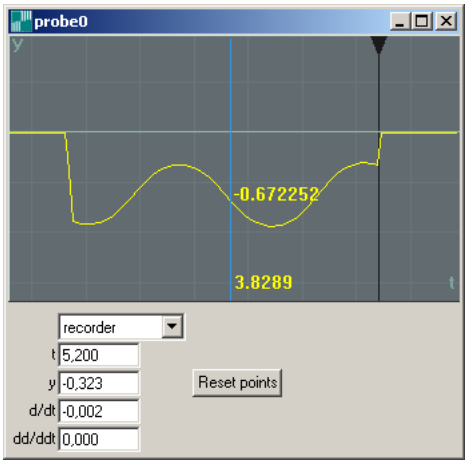


Figure 4.7 Custom properties of the Recorder function

Parameters

Reset points
Press this button to delete all recorded data.

4.5 Polynomial

This ChFunction implements a polynomial function of variable degree n, of the type:
$y = C_0 + C_1 x + C_2 x^2 + C_3 x^3 + \ldots + C_n x^n$

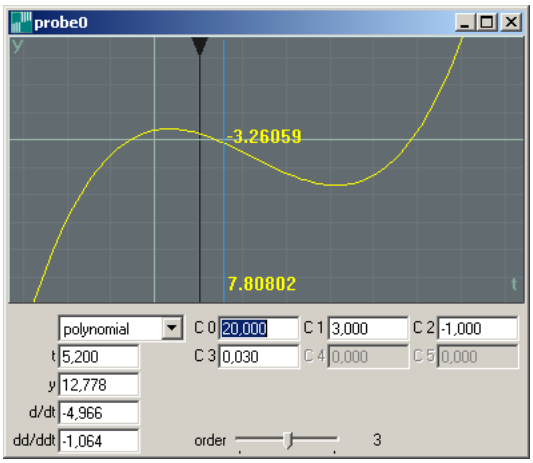


Figure 4.8 Custom properties of the Polynomial function

Parameters

C0
C1
...

C5
Coefficients C_0, C_1, etc. of polynomial.

Order
Order of polynomial (0= constant, 1= linear, 2=quadratic, etc. .

4.6 Sigma ramp

This ChFunction implements a smooth ramp transition between two constant values, using a cubic interpolation (as for motion laws with cubic position, parabolic speed, linear acceleration).

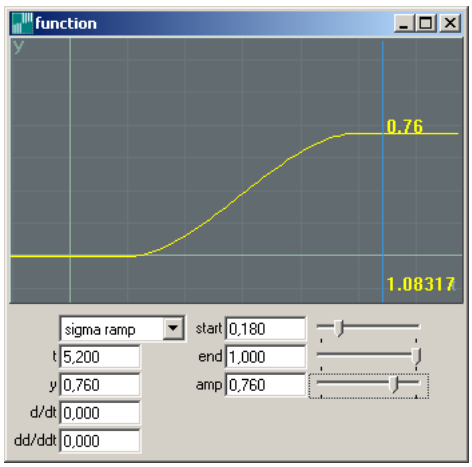


Figure 4.9 Custom properties of the Sigma ramp function

Parameters

Start
The smooth ramp starts at this value. Can also be negative.

End
The smooth ramp ends at this value.

Amp
Amplitude (height) of the ramp step. Can be negative too.

4.7 Curve

This ChFunction uses a Realsoft3D curve which can be point-edited. This can be useful to introduce hand-drawn curves and functions. Note that Realsoft3D curves are NURBS splines by default.

Parameters

Curve editor
Use this window to add and move points for the Realsoft3D curve .

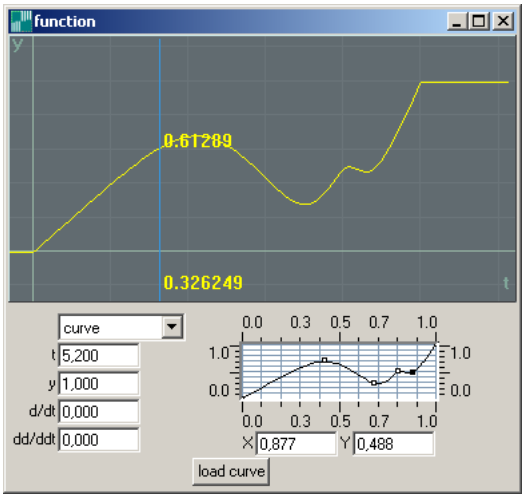


Figure 4.10 Custom properties of the Curve function

4.8 Polynomial 3-4-5

This ChFunction implements a polynomial function which is often used in robotics and automation as a motion law. In detail, it uses a polynomial with terms of 3,4,5 orders only, to perform a smooth transition between two constant values.

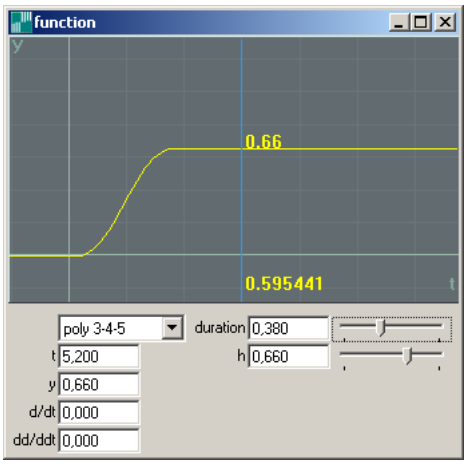


Figure 4.11 Custom properties of the Polynomial 3-4-5 function

Parameters

Duration
Duration of the smooth ramp.

h
Amplitude (height) of the smooth ramp. It can be negative too.

4.9 Constant acceleration

This ChFunction implements a constant-acceleration function which is often used in robotics and automation as a motion law. In detail, it creates a ramp where the acceleration and deceleration phases have constant values of second derivatives (constant acceleration/deceleration), in order to perform a smooth transition between two constant values. Also non-simmetric acceleration/deceleration phases can be set, and an intermediate phase of constant speed (zero acceleration) is supported.

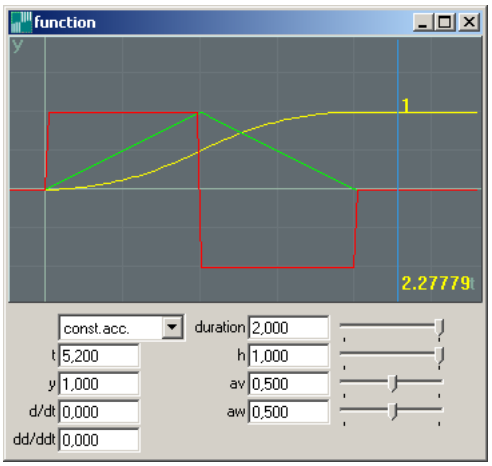


Figure 4.12 Custom properties of the Constant acceleration function

Parameters

Duration
Duration of the smooth ramp.

h
Amplitude (height) of the smooth ramp. It can be negative too.

av
Portion of duration used for the acceleration (0..1). Usually 0.5, for symmetric-acceleration laws.

aw
Portion of duration used when deceleration begins (0..1). Usually 0.5, for symmetric-acceleration laws.

4.10 Fillet

This ChFunction implements an interpolation between two other ChFunctions , using a polynomial of order 3 (a cubic interpolation: hence similar to a motion law with quadratic speed and linear acceleration).
This function can be used only as a part of the sequence function, where it keeps both C0 and C1 continuity between the two adjacent functions.

Parameters

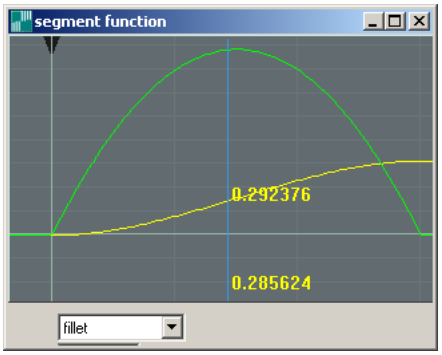


Figure 4.13 Custom properties of the Fillet function

4.11 Derive

This ChFunction derives another ChFunction , using analytical methods if available, otherwise using straightforward numerical differentiation.

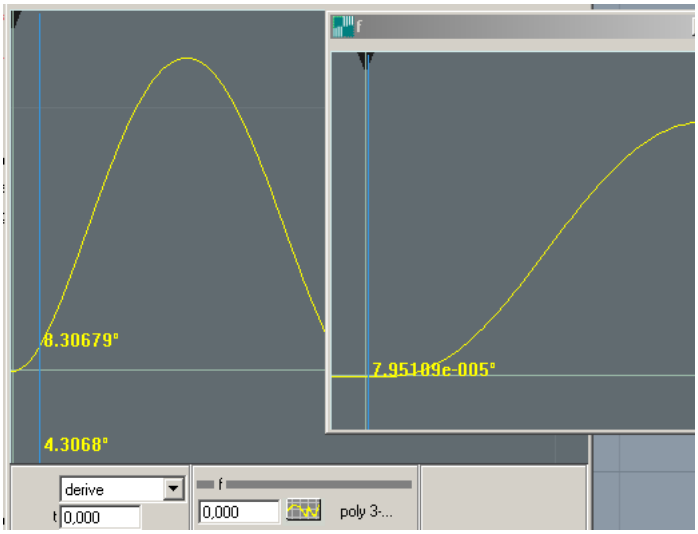


Figure 4.14 Custom properties of the Derive function

Parameters

Fa
Function to be derived.

4.12 Integrate

This ChFunction is the integral of another ChFunction . The integral of the argument function is computed by means of a numerical method (trapezoidal rule), within a range x_start - x_end, with initial value C_start. Note that this function works only on an interval of finite size. The numerical integration is performed on this interval with a finite number of samples: the higher, the more precise.

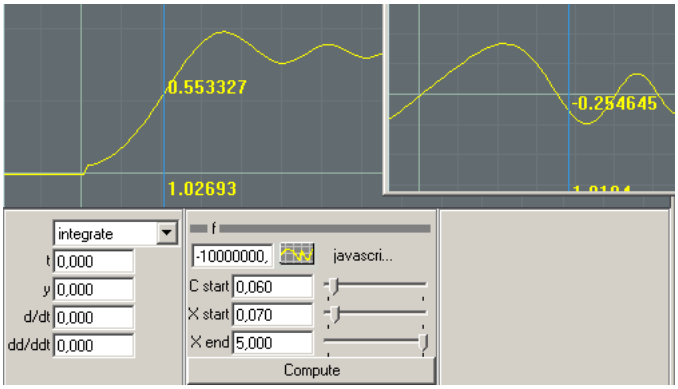


Figure 4.15 Custom properties of the Integrate function

Parameters

Fa
Function to be integrated. **C start**

Initial value of the integral. **x start**

Initial value for the interval to be computed. **x end**

End value for the interval to be computed. **Num. of samples**

Number of samples to be used for the trapezoidal rule. The higher, the more precise. (Avoid too high values for performance reasons).

4.13 Mirror

This ChFunction mirrors another ChFunction about a vertical axis.

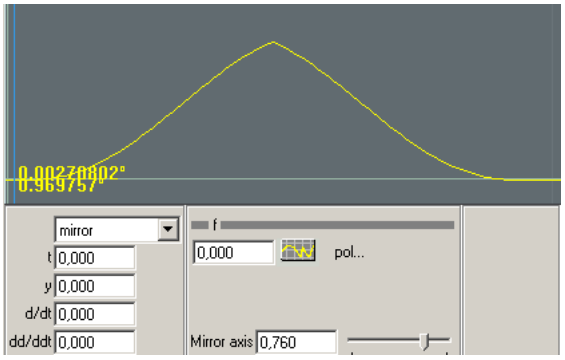


Figure 4.16 Custom properties of the Mirror function

Parameters

Fa
Function to be mirrored. **Mirror axis**

Value on x axis, after which the Fa function will be mirrored, in reverse direction.

4.14 Repeat

This ChFunction repeats another ChFunction in order to build periodic shapes.

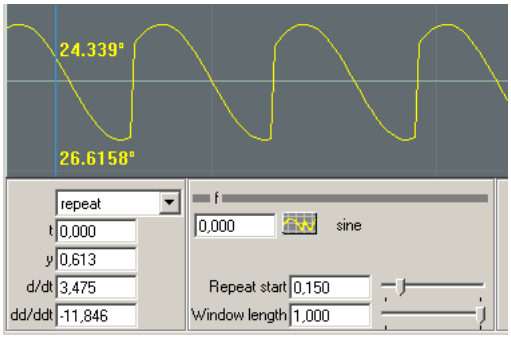


Figure 4.17 Custom properties of the Repeat function

Parameters

Fa
Function to be repeated. **Repeat start**

Value on x axis of the Fa function, after which the Fa function will be repeated. **Window length**
Duration of the period to be repeated.

4.15 Operation

This ChFunction performs an operation between two other ChFunctions .
This is a fast way to introduce functions like $y = \sin(\sin(x))$ or $y = C_1 x + sin(x)$ etc., which are composed by other basic ChFunctions .
Also, the operand functions can be 'operation' ChFunction in their turn, hence allowing the creation of hierarchical blocks of complex functions.
$y = f < f_a, f_b >$

Parameters

(operation)
You can choose the type of operation between the two ChFunctions Fa and Fb (see below).
There are many operation types available, for example:
`fa(x)*fb(x)`
`fa(x)+fb(x)`

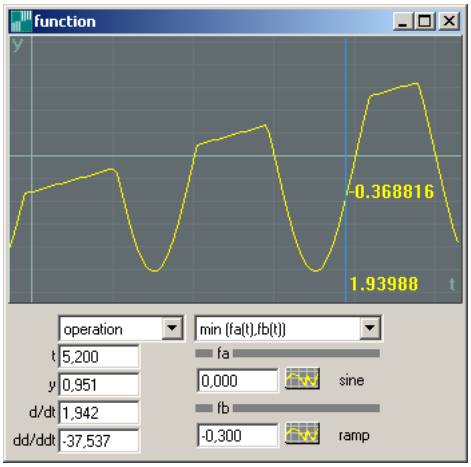


Figure 4.18 Custom properties of the Operation function

```
fa(x)-fb(x)
fa(x)/fb(x)
min<fa(x),fb(x)>
max<fa(x),fb(x)>
modulus<fa(x),fb(x)>
fabs<fa(x)>
fa(fb(x))
```

fa
The ChFunction of first operand, Fa.

fb
The ChFunction of first operand, Fb.

4.16 Noise

This ChFunction implements a cubic noise with multiple octaves.
It can be used to apply chaotic perturbations to mechanical systems, for example to simulate aleatory force fields (wind, disturbances, etc.) Note that this kind of noise does not have continuum spectrum.

Parameters

Amplitude
Maximum amplitude of noise wave, i.e. max height between the smallest and highest value.

Frequency
Frequency (in Hz, that is periods/second) of the lowest harmonic. The higher the frequency, the more ‘compact’ the noise is on horizontal axis.

Amp.ratio
Amplitude ratio, that is ratio between the amplitudes of two successive harmonics.

Octaves
Number of harmonics. Each harmonic has an octave jump in frequency, that is each harmonic has a frequency which is the double of the previous.

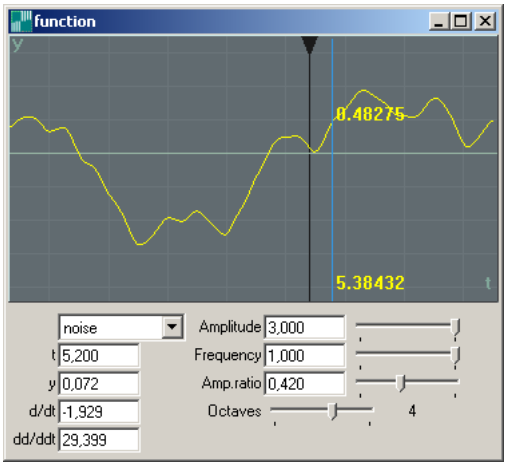


Figure 4.19 Custom properties of the Noise function

4.17 Javascript fx

This ChFunction implements a generic Javascript function.
This is useful if all other ChFunction types are not enough for your needs of customization, so here you can write an entire formula as a string with Javascript function/commands.
Some examples:

enter
`sin(2*x)+cos(pow(x,2))`
for $y = \sin(2x) + \cos(x^2)$),
enter
`pow(x,2.5)-x/3`
for $y = x^{2.5} - x/3$,
enter
`log(sqrt(x))`
for $y = \log(\sqrt{x})$,
etc.

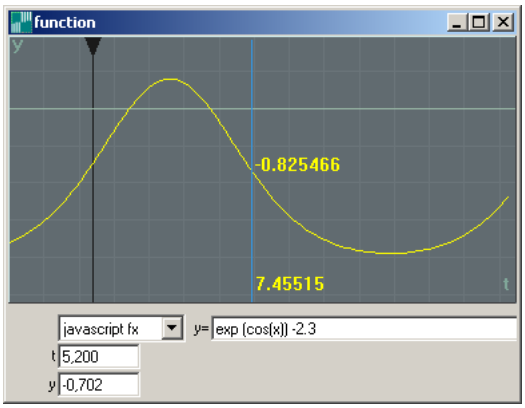


Figure 4.20 Custom properties of the Javascript fx function

Parameters

y=
In this field you can enter a Javascript formula whose evaluation returns the value that you want to set in output variable (y).
Remember that the variable `x` will be used to represent the independent variable, in Javascript formula parsing, and remember that you do not have to type `y=` at the beginning of the formula.
Also note that you can call Javascript functions or programs (which you may have defined before), for example enter `my_power_function(x)` if you have already implemented the function `my_power_function(x)` somewhere.

4.18 Sequence

This is the most important ChFunction .
In fact it can be used to build sequences of basic ChFunctions .
Whatever ChFunction can be used to build sequences: also functions of sequence type themselves can be used to build more complex sequences (in this way you may use ChFunctions as building blocks for a hierarchical construction of very complex functions).

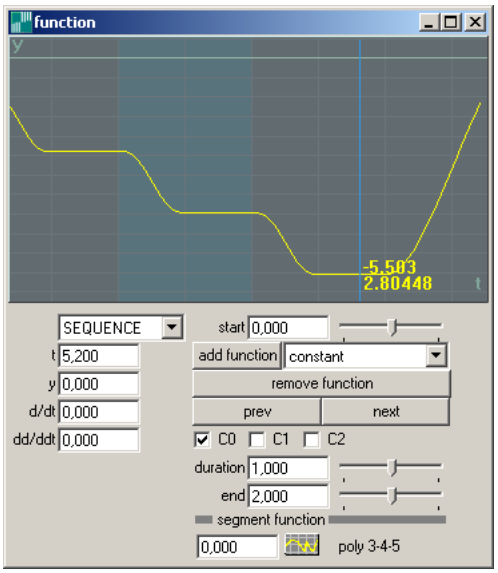


Figure 4.21 Custom properties of the Sequence function

Parameters

Start
Value of x for which the sequence starts. It can also be negative.

Add function (type)
Add a ChFunction to the sequence, right after the function interval which is currently selected. The type of the new ChFunction will depend on the type selector on the right of the button (you can modify the type later, by editing that segment).

Remove
Remove the selected ChFunction . The currently selected ChFunction can be seen in the graphical editor because it has a lighter background.

Prev
Next
Select the previous (or next) function interval. You can select the desired function interval also by clicking with the mouse in the plotting area.

C0 C1 C2
If you set these flags ON, the sequencer will impose C0 / C1 / C2 continuity between the selected interval and the previous (this means that a linear and/or parabolic term may be automatically added to the function of the selected interval).
Most useful when you are creating motion laws made of many segments. .

Duration
Duration of the selected interval

End
End time of the selected interval (depends on Duration) .

Segment function
This is the ChFunction of the selected interval. The selected ChFunction can be seen in the graphical editor because it has a lighter background.
Press this button to open the editor of the selected function (as you modify the parameters in that newly opened function editor, also the plotting in the complete sequence editor will update automatically).