
# Getting started with 20-sim 4.7

Windows 7 / 8 / 8.1 /10

# Getting Started with 20-sim 4.7


**Author: Ir. C. Kleijn, Ir. M. A. Groothuis**

### Disclaimer



### Reference

Kleijn, C., Groothuis, M.A.

Getting Started with 20-sim 4.7
Enschede, Controllab Products B.V., 2020
ISBN 978-90-79499212

### Information

Controllab Products B.V.
Address: Hengelosestraat 500
7521 AN Enschede
the Netherlands
Phone: +31-85-773 18 72
Internet: www.20sim.com
www.controllab.nl
E-mail: info@20sim.com

# Table of Contents

# 1 Welcome


This manual provides a basic overview of installing and using 20-sim. It is not a reference manual but intended as a guided tour to show you how to use 20-sim and how to create and simulate your own models.

If you are a first time user you are advised to read this manual carefully and run the various examples to get hands on experience with the package.

- **Installation**: This chapter describes the various versions of 20-sim and how to install 20-sim on your computer. It is useful for system managers and if you experience problems installing the package. You may skip this chapter if 20-sim is properly working on your computer.
- **Introduction**: This chapter describes the basic parts of the 20-sim package, the modeling representations that are supported and the various toolboxes. It gives a good overview of the package. Users who want to learn the package by trial and error.
- **Equation Models**: No one should skip this chapter! It describes the basic modeling representation of 20-sim: (differential) equations. You should run the examples of this chapter to get a good understanding of entering equations in 20-sim.
- **Block Diagram Models**: Extremely recommended once you have finished the equations. Everyone will now and then use block diagram elements and in this chapter you can learn all about this modeling representation.
- **Iconic Diagrams**: Iconic diagrams or physical components are the building blocks of models of physical systems. This chapter is absolutely worth reading for everyone who is involved in modeling physical systems.
- **Bond Graphs**: Bond graphs are a mathematical notation of physical systems. 20-sim has a large library of bond graph elements. This chapter does not explain bond graphs but how to use 20-sim to enter bond graph models.
- **3D Mechanics Toolbox**: The 20-sim 3D Mechanics Toolbox provides you with the tool that makes 3D dynamic modeling easier, the 3D Mechanics Editor.
- **Animation Toolbox**: The Animation Toolbox offers you an easy way to create 3D Animations and view graph animations.
- **Control toolbox**: The Control Toolbox of 20-sim contains several tools that can aid you in developing controllers for your modeled machines, the Controller Design Editor, the Filter Editor and the Neural Network Editors.
- **Frequency Domain Toolbox**: The 20-sim Frequency Domain Toolbox consists of the Linear System Editor, FFT Analysis and Model Linearization functionality.
- **Mechatronics Toolbox**: The Mechatronics Toolbox includes the Motion Profile Wizard, the CAM Wizard and the Servo Motor Editor.

- **Real Time Toolbox**: The Real Time Toolbox provides you with C-code generation tools and templates for all kinds of different targets and platforms.
- **Time Domain Toolbox**: During simulation, the time domain behavior of a model is calculated. Based on this time-domain behavior, the model can be analyzed. A set of powerful methods for time domain analysis is available in 20-sim.
- **Scripting Toolbox**: This chapter contains a description of the new scripting functionality that allows you to automate tasks with 20-sim using Octave, Matlab or Python.

# 2 Notation

In the 20-sim manual the following typographic notations are used:

- User instructions are numbered:

  1. Open the Simulator and start a simulation run.

- Specific 20-sim menus and menu commands are in bold:

  The simulator can always be started by the **Start Simulator** command from the **Model menu**.

- Files and directories are written in italic type:

  The file *ScaraRobot.emx* is located in *C:\Program Files\20-sim 4.7\Models\Examples \2D Mechanics* (or on 64-bit systems: *C:\Program Files (x86)\20-sim 4.7\Models \Examples\2D Mechanics)*.

- 20-sim commands, windows and window parts are started with an uppercase character and written in italic type:

  Drag and drop the model from the *Library Browser* to the *Graphical Editor*.

- Parameters, variables and other specific 20-sim elements are written in italic type:

  In the equation model the function *abs* is used to make the signal *output* equal to the absolute value of sum of variable *offset* and the signal *input*.

# 3 Installation

## 3.1 Versions

20-sim is available in two versions: Viewer and Professional.

- **Viewer/Demonstration version**: This is a freeware version that allows you to load and run models and evaluate the package. Saving of models is not possible in this version.
- **Professional**: This is the full version of 20-sim with all toolboxes.

The table below shows in detail the options that are available in the three versions:

| | Viewer | Professional |
|---|---|---|
| Library Models | v* | v |
| 3D Mechanics Toolbox | v* | v |
| Animation Toolbox | v* | v |
| Control Toolbox | v* | v |
| Frequency Domain Toolbox | v* | v |
| Mechatronics Toolbox | v* | v |
| Real Time Toolbox | v* | v |
| Time Domain Toolbox | v* | v |
| Scripting Toolbox | x | v |

**v** = included
**v*** = included but no saving possible
**x** = not available

20-sim is installed, using an Installation Manager that will lock 20-sim to your computer. There are three types of licenses available:

- **Viewer/Demonstration**: The *free* demonstration version comes with a license that is not locked to your computer. No actions have to be taken after installation of 20-sim to use this license. The limitation of this license is that you cannot save any modifications.
- **Single License**: A single license locks 20-sim to a specific computer. After installation you have to register to get a valid license.
- **Floating License**: A floating license allows multiple users to work with 20-sim at the same time. After installation you have to register to get a valid license.

## 3.2 Installing 20-sim

20-sim can be downloaded from the website www.20sim.com. This is an installation file that will install 20-sim on your computer. The first 4 steps are equal for all users. Depending on the type of license (single, floating) you have to continue differently.

1. **Install 20-sim** and **start** the program.

2. During Installation you will be asked to install the (optional) **Python 3.4 package**. We advise to keep the default setting: **Yes**.

3. **Start** 20-sim (from the **Windows Start Menu** choose **20-sim 4.7**).

If a valid license of 20-sim 4.7 was activated before, the program will start automatically and you can skip the rest of this section. If you have not installed 20-sim before, the *License Activation* dialog will open. You can also manually open the *License Activation* dialog:

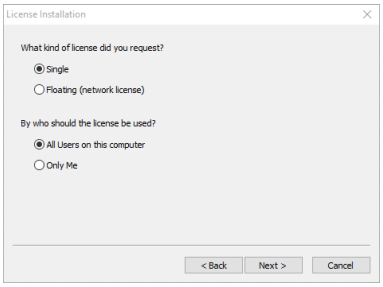
4. From the **Help** menu select **License Activation**.


*Use the Registration/Update window to request for a license.*

5. If you have a valid license key or license file, press the **Activation** button to enter your license key or browse for the license file.

If you do not yet have a valid license, press the **Trial License** button request an trial license or press the **Buy** button to purchase a license. If you want to continue in *Viewer* mode (no save functionality), just close the dialog without activating 20-sim.

6. Select **which kind of license** you have and **who** should use the license.

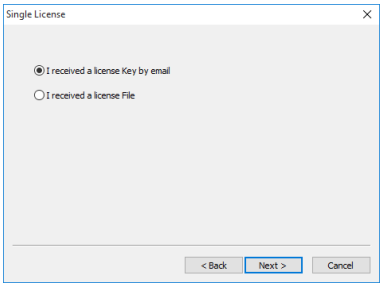



*License installation dialog.*

## Single License

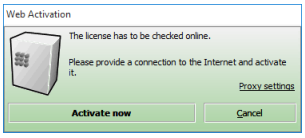
If you are using a single license, you have to enter a license key or license file.

7. On the next dialog, select ***I received a license key by e-mail*** and enter the **key** in the next dialog. When you received a **license file**, you have to enter the **location** of the license file.




*Single License dialog.*

You will be asked for confirmation (click **Activate Now**) and activation will be carried out. After a successful activation process the License Information dialog will show the new license.




*Web Activation dialog.*

### Floating License

#### Installing a floating license (Administrator)

If you are using a license that is shared by more users (floating license, also known as concurrent license or server license), you have enter the received license key and a location on the server (a normal Windows shared folder) first. This location on the server should be accessible to all users and have read/write permission. The floating license will be stored at the selected location.

6. On the next dialog, select **First Installation** and then enter the **license key** and the **location on the server** (Windows share).

On the location that you have given, a license file *20sim.lic* will be installed. Remember the location of this file because every new user of 20-sim will need to enter it. You will be asked for confirmation (click **Activate Now**) and activation will be carried out. After a successful activation process the License Information dialog will show the new license.

#### Using a floating license (Other users, Administrator)

If you are using a floating license that was already installed you have to enter the location of the license file.

7. On the next dialog, select **Administrator already installed server license** and then enter the **license location** (i.e. location of the file *20sim.lic*).

After a successful entry of the location of the license location, the License Information dialog will show the new license.

## 3.3 Uninstalling

You can uninstall 20-sim by clicking the Uninstall command from the 20-sim start menu. Uninstalling of 20-sim will not deactivate your license. If you want to move 20-sim to another computer, you have to deactivate your license first before uninstalling.

## 3.4 Deactivation

If you want to move 20-sim to another computer, you have to deactivate your license before uninstalling the program. On the new computer you can then install the program and activate the license. To deactivate your license:

1. From the Windows **Start menu** open **20-sim**.
2. From the **Help** menu choose **License Activation**.
3. Press the **Activation** button.
4. Choose **Deactivate the License** and click **Finish**

You will be asked for confirmation and deactivation will start. After a successful deactivation, you version of 20-sim has turned into the demonstration version. You can now uninstall the software and reinstall it.

## 3.5 Unattended Installation

An unattended installation is an installation that is performed without user interaction during its progress or with no user present at all.

To perform an unattended installation the default 'program files' installation directory run the following command :

```
20sim.exe /S
```

It is possible to set an alternative installation directory by specifying the /D argument. It must be the last parameter used in the command line and must not contain any quotes, even if the path contains spaces. Only absolute paths are supported.

```
20sim.exe /S /D=D:\My Installation Files\20-sim 4.7
```

## 3.6 Unattended Uninstallation

An unattended uninstall is an uninstall that is performed without user interaction during its progress or with no user present at all.
To perform an unattended uninstall from the default 'program files' installation directory run the following command on the 20-sim uninstaller:

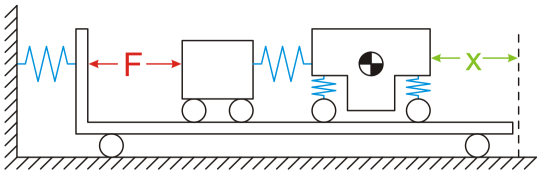
```
C:\Program Files (x86)\20-sim 4.7\Uninstall.exe /S
```

# 4 Introduction

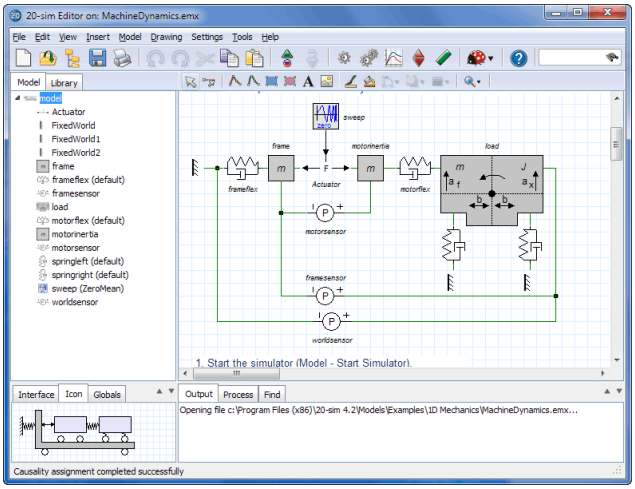
## 4.1 What is 20-sim

20-sim is a modeling and simulation program that runs under Microsoft Windows. With 20-sim you can simulate the behavior of dynamic systems, such as electrical, mechanical and hydraulic systems or any combination of these.

20-sim fully supports graphical modeling, allowing to design and analyze dynamic systems in a intuitive and user friendly way, without compromising power. 20-sim supports the use of components. This allows you to enter models as in an engineering sketch: by choosing components from the library and connecting them, your engineering scheme is actually rebuilt, without entering a single line of math!




From the engineering sketch,



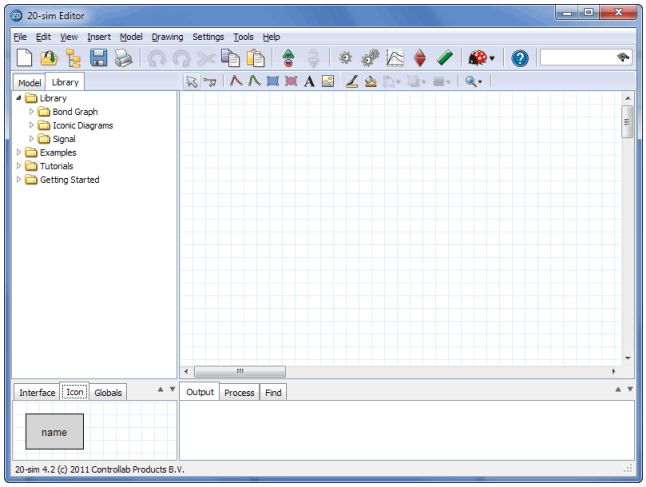

directly to a model, one on one!

## 4.2 20-sim: a quick tour

The best way to experience the capabilities of 20-sim is to open example models and run simulations. In this quick tour we will show you how to load models from the *Examples* library and run simulations.

1. **Start** 20-sim.

20-sim consists of two main windows (*Editor* and *Simulator*) and a lot of tools. The *Editor* opens when you start 20-sim. In the *Editor* you can create your models.




*The 20-sim Editor.*

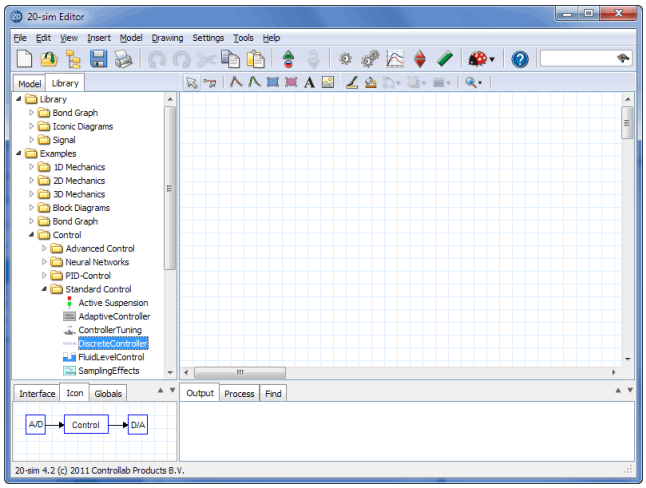
The *Editor* consists of four parts:

- *Model tab / Library tab*: This is the part at the middle left. The *Model tab* shows the *model hierarchy*, i.e. the hierarchical composition (all the elements) of the model that is created in the *Editor*. The *Library tab* shows the 20-sim library.
- *Graphical Editor / Equation Editor:* This is the big white space at the middle right. In this editor you can create graphical models and enter equations.
- *Output tab / Process tab / Find tab:* This is the part at the bottom right. The *Output tab* shows the files that are opened and stored. The *Process tab* shows the compiler messages. The *Find tab* shows the search results.
- *Interface tab / Icon tab:* This is the part at the bottom left. The *Interface tab* shows the interface of a selected model. Double clicking it will open the *Interface Editor*. The *Icon* tab shows the icon of a selected model. Double clicking it will open the *Icon Editor*.

We will open the model *DiscreteController.em* from the *Examples\Control\Standard Control* folder.

2. Select the **Library tab** to open the *Library Browser* (shows the 20-sim library).

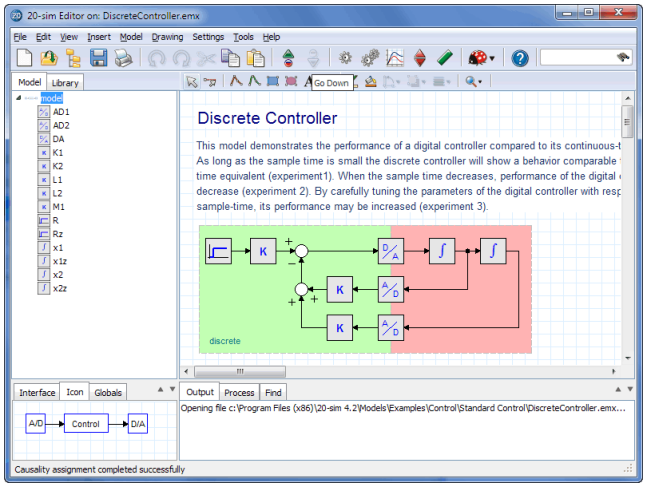
3. In the *Library Browser* select **Examples** - **Control** - **Standard Control** - **Discrete Controller**. Your *Editor* should now look like:




*The 20-sim Editor with the model library selected.*

Note: All models in 20-sim are stored on file with the extension *.emx*. Library models can be found where 20-sim was installed, e.g. *C:\Program Files (x86)\20-sim 4.7 \Models*. The model *DiscreteController* is stored in: *C:\Program Files\20-sim 4.7 (x86) \Models\Examples\Control\Standard Control\DiscreteController.emx*

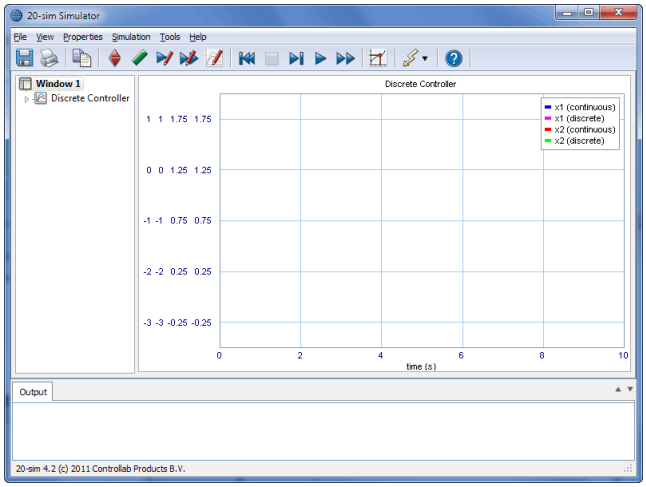
4. **Drag and drop** the *Discrete Controller* model to the graphical editor (large white area). Now the model is opened. Your *Editor* should look like:




*The 20-sim Editor with the model DiscreteController.emx loaded.*

You can inspect the model by enlarging the *Editor* window or using the zoom button. We will continue the quick tour running a simulation.

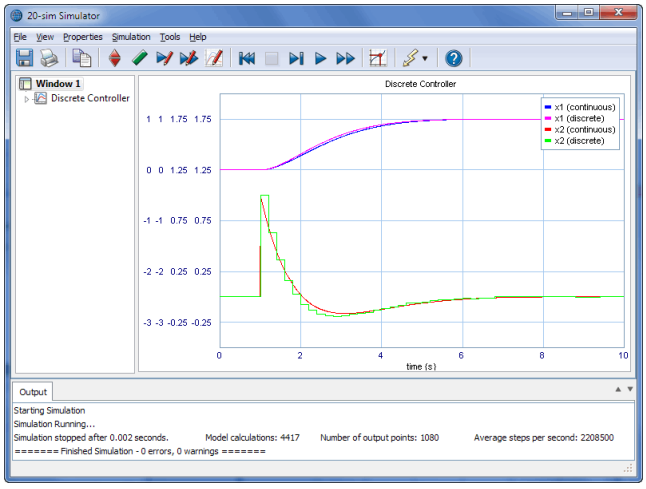
5. In the **Model** menu select **Start Simulator**. Now the *Simulator* will be opened.




*The 20-sim Simulator with the model DiscreteController.emx loaded.*

In the *Simulator* you can run a simulation and show the results in plots and animations. The *Simulator* contains various tools to analyze the simulation results.

6. In the **Simulation** menu select **Run**. Now a simulation run will be performed. Your *Simulator* should look like:




*The 20-sim Simulator with the simulation results.*

You have just learned how to open and run an example model. Try to load and run other models from the *Examples* library to find out more about the capabilities of the 20-sim package.

## 4.3 Library

In 20-sim, creating models only takes you just a few mouse clicks. By *dragging* an element from the library and *dropping* it in the graphical editor, your model is actually built the same way as you would draw an engineering scheme. 20-sim supports various model representations, such as block diagrams and iconic diagrams. These representations may be combined in one model.

### Library Browser

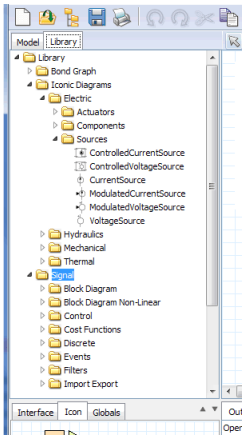
All models in 20-sim are stored on files with the extension *.emx*. The standard libraries can be found in the 20-sim folder:

*C:\Program Files\20-sim 4.7\Models*

or on 64-bit systems:

*C:\Program Files (x86)\20-sim 4.7\Models*

This folder contains all the models that are visible in the *Library Browser* on the left part of the *Editor.*




*You can find the library at the left of the Editor.*

The library contains 4 sections:

- *Bond Graph*: bond graph elements
- *Iconic Diagrams*: Physical components
- *Signal*: Block diagram elements
- *Tutorial*: example models that show you how to perform various tasks in 20-sim
- *Getting Started*: al the models that you need in the lessons of the Getting Started manual.

### Open Models

All library models are open source. You can inspect the content of any model in the *Editor*. If the model contains a hierarchy, you can use the *Go Down* command of the *Model* menu to descend in the hierarchy. If a model opens a specific editor, you can still inspect the underlying code by keeping the *shift key* pressed while clicking the *Go Down* command.

### Custom Libraries

You can create your own model libraries in 20-sim:

1. From the **Tools** menu click **Options** - **Folders** - **Library Folders**.
2. Add your **folder.**
3. Give it a useful **name** by clicking **Edit Label**.
4. Click OK to close the dialog.

Then you can add your own library models to the library:

5. Select the **submodel** that you want to store in your library.
6. From the **File menu** select **Save Submodel**.
7. Store the submodel in your library **folder**.

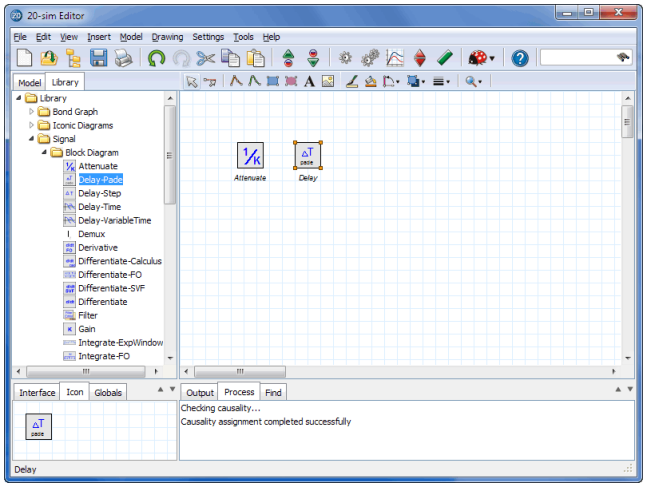
The next time you start up 20-sim, the library will show the new submodel.

## 4.4 Block Diagrams

Block diagrams allow you to graphically represent the mathematical relationships between signals in a system. They are especially suited to model control systems. In 20-sim a large library of block diagram elements is available. The elements are displayed in the Editor by icons. You can create block diagram models by dragging the elements to the Graphical Editor and making the proper connections between the elements.

### Library

20-sim has a large library of block diagram elements such as linear, non-linear, discrete and source elements. In 20-sim you can create custom made block diagram elements and add them to the existing libraries or combine them in newly defined libraries.




*From the Library Browser (left) you can drag and drop elements into the Graphical Editor (right).*

### Signals

The foundation of block diagram elements is the use of input and output signals. 20-sim allows you to create user defined block diagram elements with an arbitrary number of input and output signals. Signal sizes can be 1 (default) or larger.

### Custom Made Models

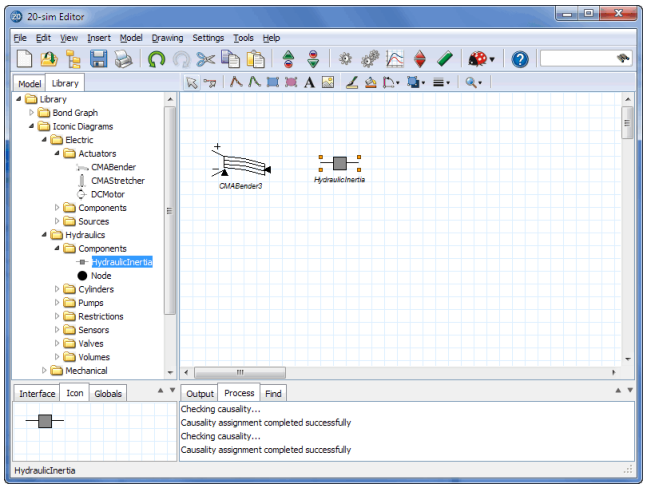
In 20-sim you can create your own block diagram elements and save them in your own model library. Models can have an arbitrary number of ports, input and output signals. A specialized drawing editor can be used to give the models any kind of representation.

## 4.5 Iconic Diagrams

Iconic diagrams or components are the building blocks of physical systems. They allow you to enter models of physical systems graphically, similar to drawing an engineering scheme. In 20-sim a large library of iconic diagram elements is available. The elements are displayed in the *Editor* by icons which look like the corresponding parts of the ideal physical model. You can create models by dragging the elements to the *Graphical editor* and making the proper connections between the elements.

### Library

20-sim has a large library of iconic diagram elements such as electrical, hydraulic, mechanic and thermal models. In 20-sim you can create custom made iconic diagram elements and add them to the existing libraries or combine them in newly defined libraries.




*From the Library Browser (left) you can drag and drop elements into the Graphical Editor (right).*

### Ports and Multiports

The foundation of iconic diagram elements is the use of power ports. Power ports enable the connection between elements by describing the power flow between the elements. A power port consists of two signals which are called across and through. 20-sim allows you to create user defined iconic diagram elements with an arbitrary number of power ports. Port sizes can be 1 (default) or larger (multiports).

### Algebraic Loops and Differential Causality

Algebraic loops and differential causalities are traced automatically. If possible, 20-sim will rewrite the equations symbolically to remove algebraic loops and differential causalities.

### Custom Made Models

In 20-sim you can create your own iconic diagram elements and save them in your own model library. Models can have an arbitrary number of ports, input and output signals. A specialized drawing editor can be used to give the models any kind of representation.

## 4.6 Bond Graphs

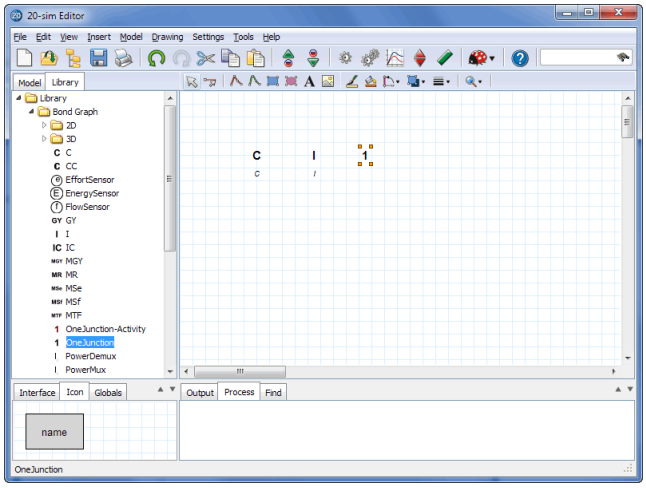
20-sim was the first commercially released software package to support bond graph modeling. The first version of 20-sim with a bond graph library was released in 1995. Since then a continuous effort to improve bond graph modeling has made 20-sim the standard in bond graph modeling.

## Bond Graphs

Bond graphs are a network-like description of physical systems in terms of ideal physical processes. With the bond graph method, the system characteristics are split-up into an (imaginary) set of separate elements. Each element describes an idealized physical process. To facilitate drawing of bond graphs, the common elements are denoted by special symbols.

## Library

20-sim has a large library containing all standard bond graph elements. Next to standard elements, 20-sim supports custom user made bond graph models.



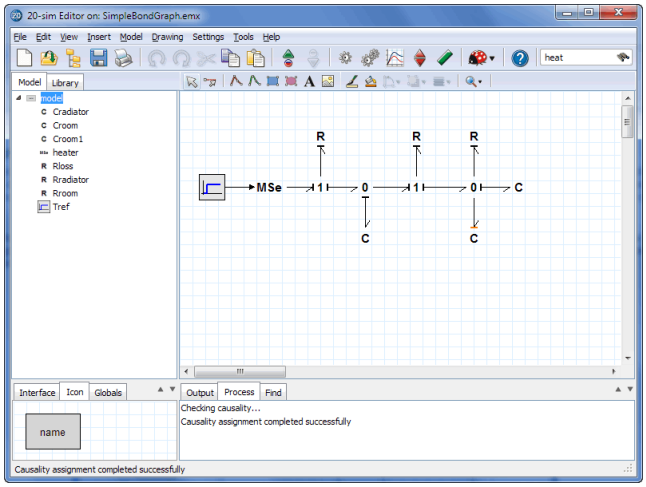

*From the Library Browser (left) you can drag and drop elements into the Graphical Editor (right).*

## Ports and Multiports

The foundation of bond graph modeling is the use of power ports. Power ports form the connections with other bond graph elements and consist of two signals which are called effort and flow and multiply to power. 20-sim allows you to create user defined models with an arbitrary number of power ports and signals. Ports sizes can be 1 (default) or larger (multiports). For every port you can specify the causality as fixed preferred, indifferent or depending on the causality of other ports.

## Causality

Causal strokes indicate the direction of the efforts and flows in a bond graph model. In 20-sim you have to enter the equations in one of the possible causal forms only. If causality is changed, the equations are rewritten automatically. 20-sim shows causal strokes in black color for preferred causality and in causal strokes in orange color for non-preferred causality. The Causality of a complete model is derived automatically but can be changed manually.



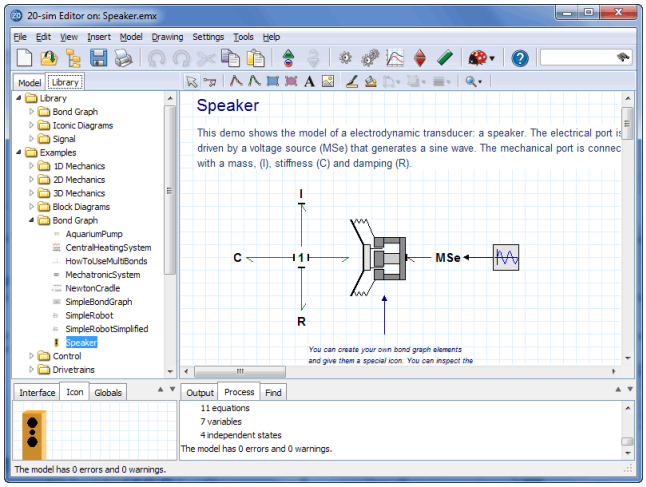

*20-sim will assign causality automatically to your bond graph model.*

## Algebraic Loops and Differential Causality

Algebraic loops and differential causalities are traced automatically. If possible, 20-sim will rewrite the equations symbolically to remove algebraic loops and differential causalities.

### Custom Made Models

In 20-sim you can create your own bond graph models and save them in a custom made model library. Models can have an arbitrary number of ports, input and output signals. A specialized drawing editor can be used to give the models any kind of representation.




*Create custom made bond graph elements.*

## 4.7 Editor

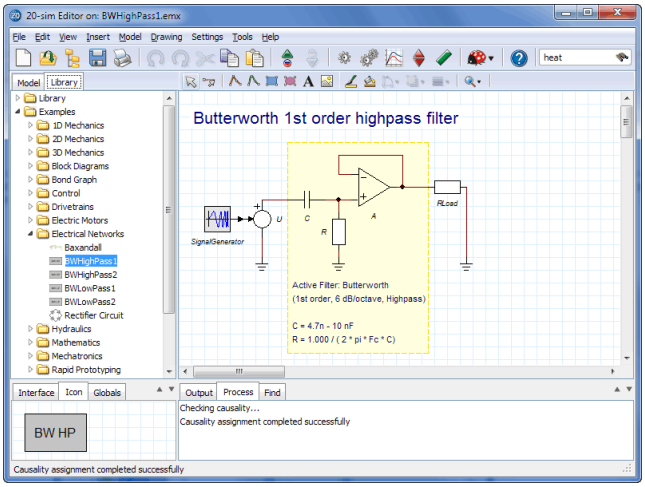
Models are entered and compiled in the 20-sim *Editor*. The *Editor* is a versatile tool that helps you to enter models supporting a wide variety of systems including linear, non-linear, discrete-time, continuous-time and hybrid systems, without restricting the user to a certain model representation. After entering and debugging, the model can be checked and compiled. This is performed automatically in the background, when opening the *Simulator*.

### Model Representations

Systems can be modeled in 20-sim, using equations, state space descriptions, bond graphs, block diagrams, and components or iconic diagrams. These descriptions can be fully coupled to create mixed models.

## Open Source

20-sim models are stored in files. A *Library Browser* is part of the program, but you can also use the Windows Explorer for library management. All 20-sim models are open! You can drag and drop them from the library browser into the graphical editor to build new models. You are allowed to store the original and changed models in separate folders to create your own library of models.




*You can drag and drop models from the library browser to the graphical editor.*

## Debug Mode and Fast Mode

20-sim can operate in two modes: *Debug Mode* and *Fast Mode*. This is indicated by the *Mode* button at the complete right of the toolbar. You can quickly change between these modes by clicking on the *Mode button*. In *Debug Mode* all possible checks will be performed and warnings will be generated for possible model errors. Always start modeling in *Debug Mode*!

## Graphical Models

You can construct graphical models (block diagrams, iconic diagrams, bond graphs) by dragging and dropping models from the *Model Library* to the *Graphical Editor*. Default the Editor is in *Selection Mode*. If you want to make connections between the models, you have to change to the *Connection Mode*.

## Drawing

The icon of every standard library model in 20-sim has been created using a special drawing editor: the *Icon Editor*. You can use the *Icon Editor* to change the standard library models or create your own library models. You can enter text and add bitmap pictures in every level of your model. With lines, arrows and other drawing objects you can enhance the understanding of your model.

## Hierarchical modeling

20-sim supports unlimited hierarchical modeling. The highest levels consist of graphical models (state space models, block diagrams, bond-graphs or components) and the lowest level is formed by equations models.

### Equations

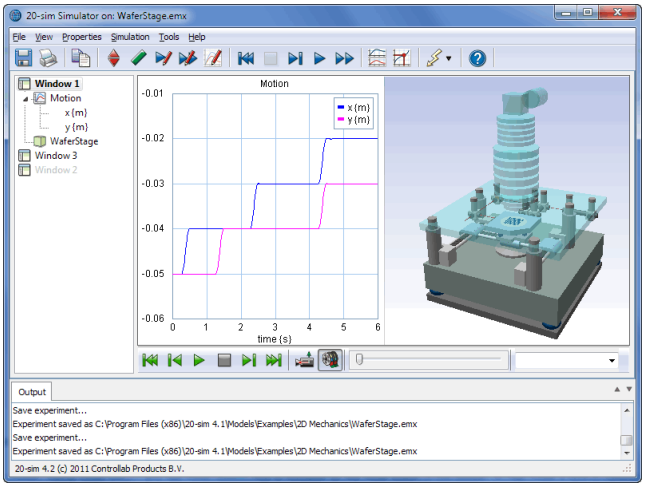
The lowest level in a 20-sim model is formed by equations. Equations in 20-sim follow the standard mathematical notation and can be changed by the user. A large collection of linear, non-linear, scalar and matrix functions are available for the use in equations.

## 4.8 Simulator

After entering a model in the *Editor* you can check and compile it. This is performed automatically in the background, when opening the *Simulator*. The *Simulator* is used for model simulation and analysis.

### Plot Windows

Simulation results can be shown in plots and animations. These plots and animations can be part of the main *Simulator* window or additional windows. Plots are fully configurable. Logarithmic views, true-type fonts, line styles, marker styles and backgrounds are supported. Plots and animations can be made ready for publication easily (copy to clipboard and paste in any document).




*Show simulation results in plots and animations simultaneously.*

### Debug Mode and Fast Mode

20-sim can operate in two modes: *Debug Mode* and fast Mode This is indicated by the *Mode* button in the toolbar. In Fast mode a built-in runtime compiler is used which compiles the simulation model into platform specific 32-bit machine code. The result is a dramatic increase of simulation speed. 20-sim machine code runs faster than the equivalent compiled C-code! Compiling the machine code, even with large models, is done while you start up the *Simulator*. The compiler is an internal part of the 20-sim software. No external compiler or program is required!

### Simulation Algorithms

20-sim contains powerful simulation algorithms for solving ordinary differential equations (ODE) and differential algebraic equations (DAE). It has a variety of numerical integration methods: one-step, multi-step and multi-order.

### Discrete-time models

20-sim will automatically detect discrete-time loops in a model and assign each independent loop a separate sample rate. Discrete signals are shown in the Editor in green. Discrete-time parts are activated by time events so that mixed continuous-time and discrete-time models are handled correctly.

### Events

20-sim can also handle state events based on zero-crossing algorithms. This results in a fast and accurate event detection and localization.

## 4.9 Toolboxes

20-sim contains a number of Toolboxes:

1. 3D Mechanics Toolbox: This toolbox consists of the *3D Mechanics Editor*.
2. Animation Toolbox: This toolbox consists of the 3D Animation and Graph Animation tools.
3. Control toolbox: This toolbox consists of the Controller Design Editor, the MLP Network Editor, the B-Spline Editor and the Filter Editor.
4. Frequency Domain Toolbox: This toolbox consists of FFT analysis and Linearization.
5. Mechatronics Toolbox: This toolbox consists of the Cam Wizard, the Motion Profile Wizard and the Servo Motor Wizard.
6. Real Time Toolbox: This toolbox allows you to create C-code out of any 20-sim model for the use in real-time applications.
7. Time Domain Toolbox: This toolbox contains powerful tools to inspect the behaviour of your model using time domain simulation: Parameter sweeps, Optimization, Curve Fitting, Tolerance analysis, Sensitivity analysis, Monte Carlo analysis and Variation analysis. You can also use External DLL's to run your time domain simulations.
8. Scripting Toolbox: This toolbox contains powerful tools to inspect the behaviour of your model using time domain simulation: Parameter sweeps, Optimization, Curve Fitting, Tolerance analysis, Sensitivity analysis, Monte Carlo analysis and Variation analysis. You can also use External DLL's to run your time domain simulations.

# 5 Equation Models

## 5.1 Introduction

Equations are the foundation for all models in 20-sim. At the lowest level of a model you will always find equations. Equations can be entered in the 20-sim *Editor*.

1. **Open** 20-sim and select **File**, **New** and **Graphical Model**.

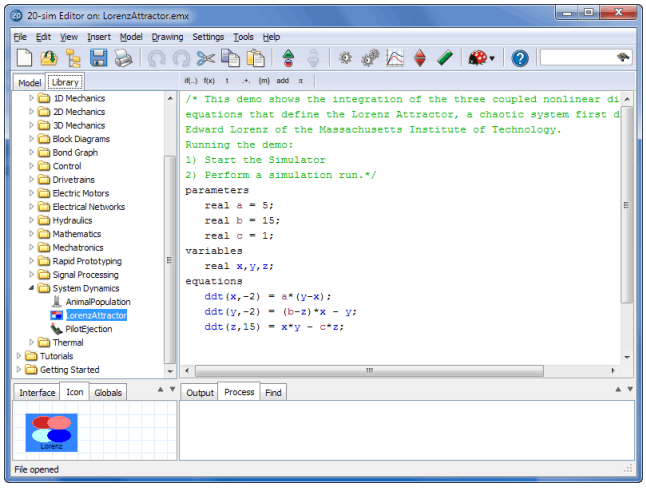
The right part of the *Editor* will now allow you to graphical models. That is why we have named this part of the *Editor* the *Graphical Editor*. The *Graphical Editor* will change into an *Equation Editor* if we go to the deepest level of any model.

2. Go to the left of the *Editor* and click the **Library tab**

Now the *Library Browser* will appear.

2. Click on **Examples** and **System Dynamics**.

3. Drag the model **LorenzAttractor** to the white space at the right (*Graphical Editor*).




*Equation model of the Lorenz Attractor.*

As you will see the *Graphical Editor* changes into an *Equation Editor and* equation model is opened. This model is called an *equation mainmodel*, because it has no input signals, output signals or ports. This can be verified in the lower left part of the *Editor* which shows the *Interface* of the model (empty). A *mainmodel* is a model that cannot be connected with other models.

4. Open 20-sim and select **File**, **New** and **Graphical Model**.

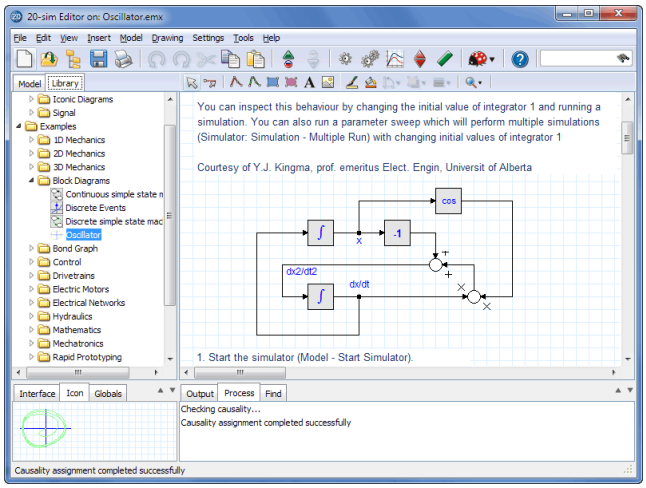
5. Go to the left of the *Editor* and click the **Library tab**.

Now the *Library Browser* will appear.

6. Click on **Examples** and **Block Diagrams**.

7. Drag the model **Oscillator** to the white space at the right (*Graphical Editor*).

As you will see a block diagram model is opened. We will inspect the integrate element of this model.

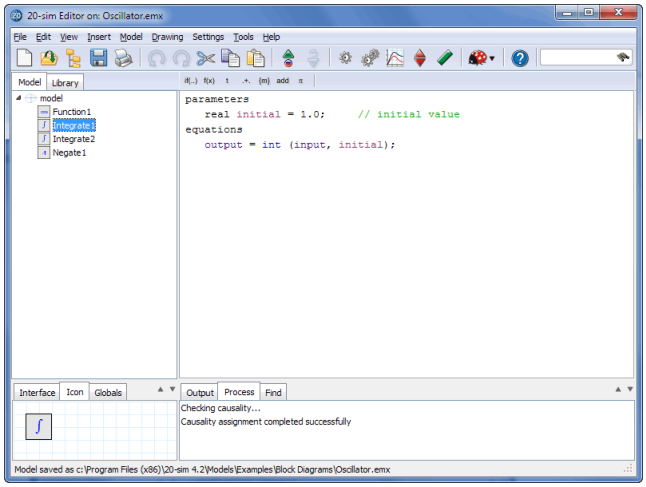



*Block diagram model of an oscillator.*

8. Go to the left of the *Editor* and click the **Model tab**.

Now the *Model Browser* will appear. The *Model Browser* shows the relevant block diagram elements of this model.

9. Select the **Integrate1** element.




*Equation implementation of an integration element.*

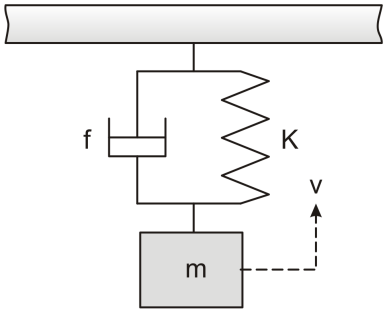
As you can see an equation model is shown. This is called an *equation submodel*, because it has equations which relate an output signal to an input signal. The signals are defined in the lower left part of the *Editor* (the *Interface*). A *submodel* is in general a model that can be connected to other *submodels* using *signals* or *power ports*.

In the next sections we will show how to create equation mainmodels and submodels. We will start with a simple mechanical system and show how it can be described by differential equations. We will enter these equations in 20-sim in the form of an equation mainmodel. In next section we will show how to model a pendulum with differential equations incorporated in an equation submodel.

## 5.2 Equation Mainmodel

### Differential Equations

We consider the mechanical system of the figure below.




*A mass with spring and damper.*

We will first derive the necessary equations for this system and then enter these equations in 20-sim and do a simulation. For the mass we can write the following equation:

$$F_m = m \cdot a = m \cdot \frac{dv}{dt}$$

where *Fm* is the force on the mass *m* due to its inertia, *v* the velocity and *a* the acceleration. In block diagram models integration is to be preferred above differentiation, so we rewrite the equation for *Fm*:

$$v = \frac{1}{m} \int F_m dt$$

For the spring we can write:

$$F_s = K \cdot x = K \int v \, dt$$

where *Fs* is the force on the spring and *K* is the spring constant. For the damper we can write:

$$F_d = f_v$$

where *Fd* is the force due to viscous friction and *f* is the friction parameter. For the gravity force we can write:

$$F_g = -m \cdot g$$

where *Fg* is the force due to gravity and *g* is the acceleration of gravity. Combining the forces leads to the following set of differential equations:

$$F_m = -m \cdot g - F_s - F_d$$

$$v = \frac{1}{m} \int F_m \, dt$$

$$F_s = K \cdot x = K \int v \, dt$$

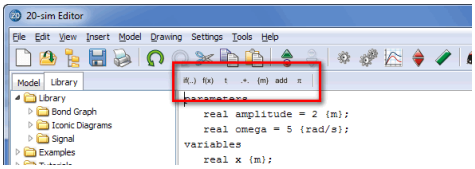
$$F_d = f_v$$

## Equation Model

These equations can be entered in 20-sim directly as equations. We will use an equation mainmodel. A *main* model means that the model has no input or output signals to connect it with the outside world.

1. Open 20-sim and select **File**, **New** and **Equation Model**.

Make sure that 20-sim is in **Debug Mode**. The right part of the *Editor* will now allow you to create graphical models. That is why we have named this part of the *Editor* the *Graphical Editor.* Equations in 20-sim written in a special language called SIDOPS+. This language is similar to mathematical equations and is easy to learn.

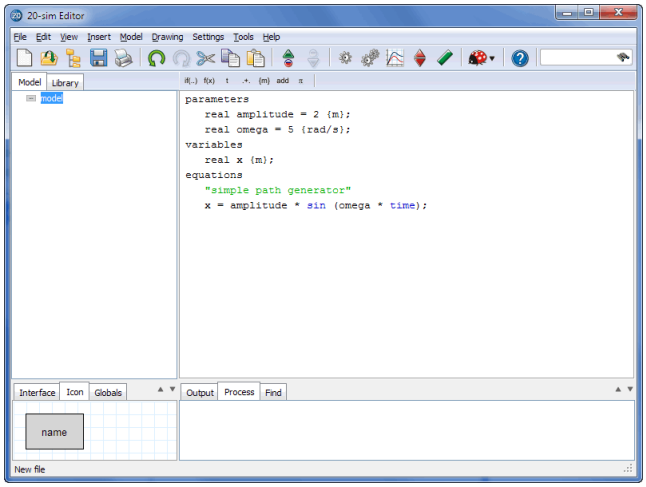



*The taskbar of the Equation Editor.*

Look at the *taskbar* at the top of the *Equation Editor*. The *taskbar* contains buttons to insert functions, statements and other language constructs of SIDOPS+.

2. From the *task bar* select the **add** button**.** Choose **Equation Examples** and **Simple**.

Now you will see a simple example of an equation model. Your *Editor* should look like:




*Using the add button you can insert the template of a simple set of equations.*

An equation model starts with the declaration of *parameters* and *variables*, followed by the actual *equations*. You can easily get help by putting the mouse pointer on top and pressing the **F1** key.

3. Put the mouse pointer on top of the word **sin**, click once with the left mouse button and press the **F1** key.

Now the 20-sim *Help file* should open with the topic on the *sine* function opened.

4. **Close** the Help File.

We will enter the differential equations from the previous topic.

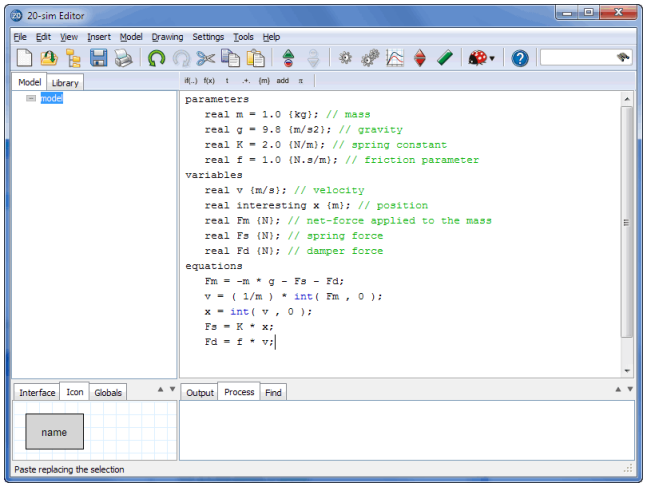
5. **Clear** all **equations** from the Editor (select them and press delete).

6. **Enter** the following **equations** (you can copy and paste them):

```
parameters
    real m = 1.0 {kg}; // mass
    real g = 9.8 {m/s2}; // gravity
    real K = 2.0 {N/m}; // spring constant
    real f = 1.0 {N.s/m}; // friction parameter
variables
    real v {m/s}; // velocity
    real interesting x {m}; // position
    real Fm {N}; // net-force applied to the mass
    real Fs {N}; // spring force
    real Fd {N}; // damper force
equations
    Fm = -m * g - Fs - Fd;
    v = ( 1/m ) * int( Fm , 0 );
    x = int( v , 0 );
    Fs = K * x;
    Fd = f * v;
```

7. **Delete empty lines** and use the tab-key to get a better layout.

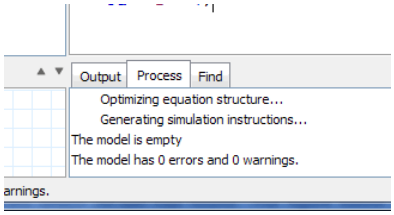
Your Editor should now look like:




*The equations of the spring-damper-mass model entered.*

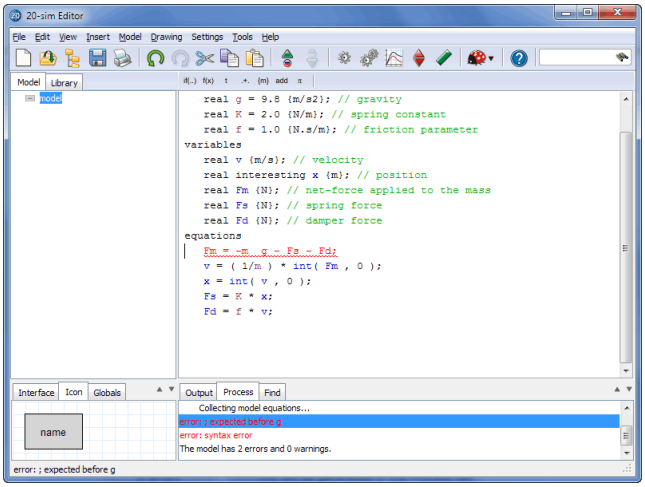
8. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings:




*The results of model compilation are shown in the Process tab.*

If any errors are found, a message window pops up, showing the errors which 20-sim has found. The figure below shows the errors that are generated when a multiplication is missing (*m g* should be *m*g*). You can click on the error in the *Process tab* to highlight the corresponding equation.




*If errors occur, messages will be generated in the Process tab.*

9. If any errors occur, try to solve them.

10. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *EquationModel.emx*.

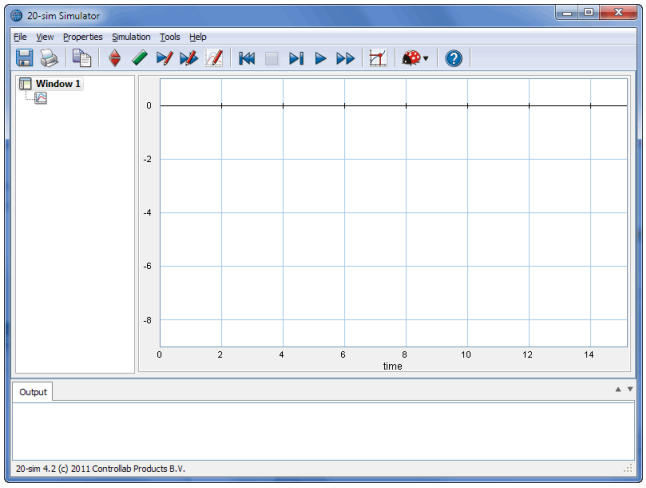
If you have problems entering the equations or checking the model, load the model *Equation Mainmodel* from the *Getting Started\Equation Models* section of the library.

## Simulation

Now we have entered the equation model, we will proceed and show how you can run a simulation.

11. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.

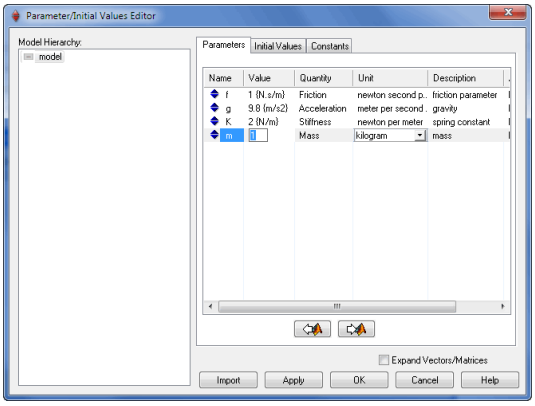
This opens the Simulator window. We will proceed with this window.




*The Simulator will open with an empty plot.*

12. In the *Simulator* toolbar from **the Properties** menu select the **Parameters** command.

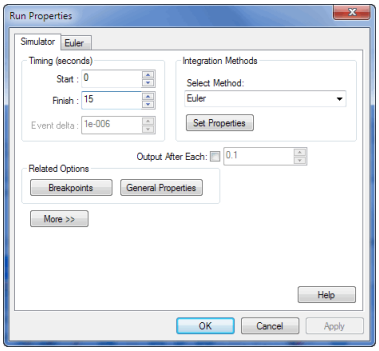
The *Parameters Editor* shows all the parameters of your model. It is a useful tool to quickly change the value of a parameter. Check that the parameter values of your model are equal to the values in the picture below.




*The Parameters Editor allows you to make quick changes to the parameters in a model.*

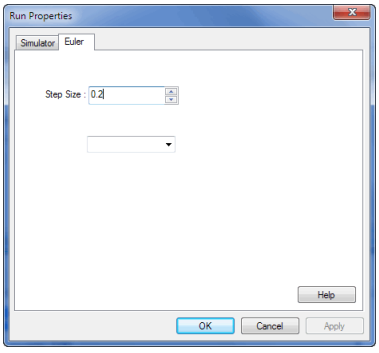
13. From the **Properties** menu select the **Run** command.

14. The *Run Properties Editor* shows the simulation settings. Change the **Finish Time** to 15 s and the Integration Method to **Euler**.



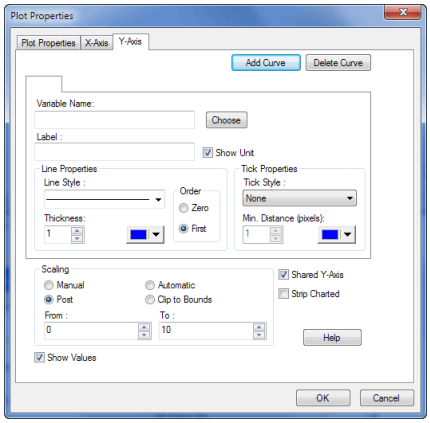

*The Run Properties Editor allows you change the simulation settings.*

15. Specific settings for the Euler method can be selected by clicking the **Set Properties** button. Set the **step size** equal to 0.2 s.




*The second tab shows the specific settings of the chosen Integration Method.*

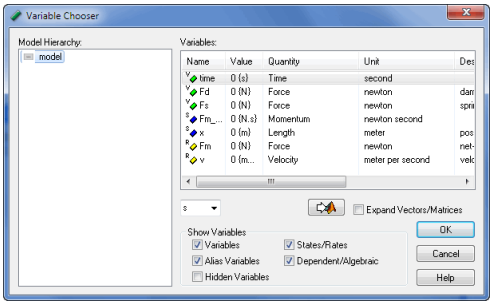
16. From the **Properties** menu select the **Plot** command.




*In the Plot Properties Editor you can define the settings of a plot.*

17. In the **Y-axis** tab click **Choose**.

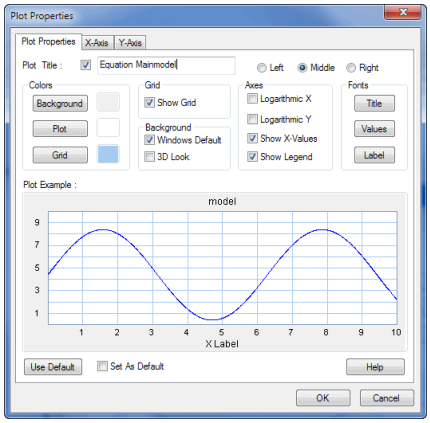
This opens the *Variable Chooser* which helps you to select the variable that should be plotted. It should look like:




*The Variable Chooser shows the model variables and their current values.*

18. Select the variable ***x*** and click **OK**.

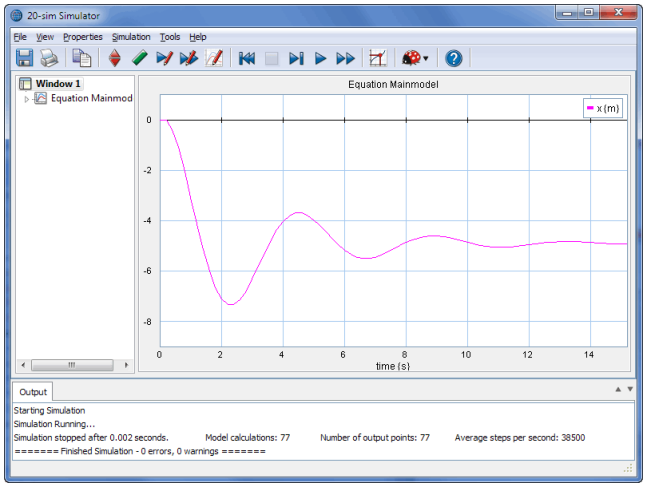
19. Select the **Plot Properties** tab and change the **Title** to *Equation Mainmodel*.




*You can give a plot any desired look and feel.*

20. Click **OK** to close the Plot Properties Editor.

21. From the **Simulation** menu select the **Run** command to start the simulation. The result should look like the figure below.




*The simulation results of our equation mainmodel.*

22. From the **File** menu click **Save**.

If you had problems running a simulation, load the model *Equation Mainmodel* from the *Getting Started\Equation Models* section of the library.

## 5.3 Equation Submodel

### Differential Equations

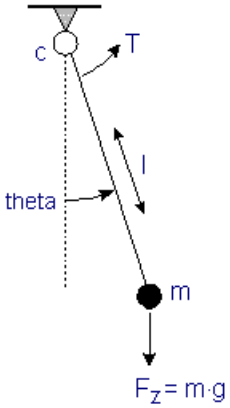
In the second part of this lesson you will learn how to enter the equations of motion of a simple pendulum in an equation submodel. If we transfer all equations to the hingepoint with angle theta, the equations of motion of a simple pendulum can be described as:

$$m \cdot l^2 \cdot s^2 \cdot theta = T - c \cdot s \cdot theta - m \cdot g \cdot l \cdot \sin(theta)$$

or

$$s^2 \cdot theta = (T - c \cdot s \cdot theta)/(m \cdot l^2) - g/l \cdot \sin(theta)$$

with:

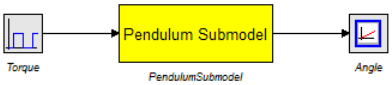



| | |
|---|---|
| theta | pendulum angle (rad) |
| T | applied torque on the pendulum (Nm) |
| l | pendulum length (4 m) |
| m | pendulum mass (0.25 kg) |
| c | friction at the pendulum hinge (0.1 Nms/rad) |
| g | gravity constant (9.8 m/s²) |

Written as first order differential equations:

```
theta_dot_dot = (torque - c·theta_dot)/(m*l2) - (g/l)·sin(theta);
theta_dot = int(theta_dot_dot);
theta = int(theta_dot);
```

The input variable is the applied torque and the output variable is the pendulum angle theta. This equation submodel will be used in a block diagram model as shown below. A square wave generator will act as torque and the signal monitor block will catch the output angle theta.




*A pendulum submodel implemented by equations.*

### Entering Equations

These equations can be entered in 20-sim directly as equations. We will use an equation submodel. A *submodel* means that the model has input or output signals to connect it with the outside world.

1. Open 20-sim.

Make sure that 20-sim is in **Debug Mode**.

2. Select **File**, **New** and **Graphical Model**.

The right part of the *Editor* will now allow you to graphical models. That is why we have named this part of the *Editor* the *Graphical Editor*. The *Graphical Editor* will change into an *Equation Editor* if we go to the deepest level of any model.

3. Go to the left of the *Editor* and click the **Library tab**.

4. Drag the following library models to the *Graphical Editor*:

| **model library** | **model** |
|---|---|
| *Library\Signal\Sources* | *WaveGenerator-Square* |
| *Library\Signal\Block Diagram* | *SignalMonitor* |

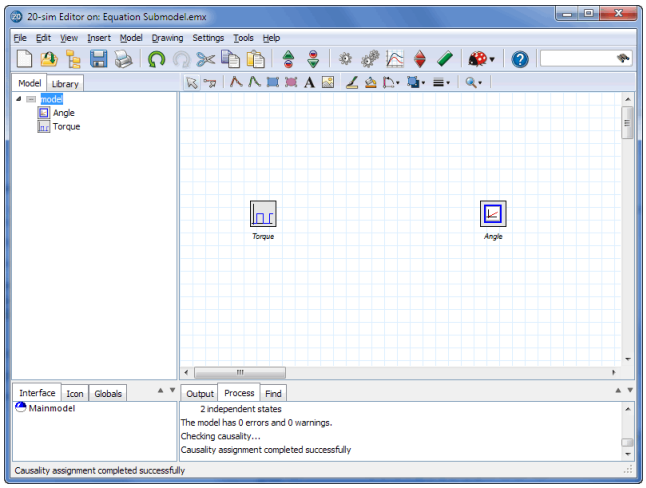
5. Select the *SignalMonitor* submodel.

6. From the **right mouse menu** or from the **Model** menu select the **Properties** command.

7. Change the **name** of the model to *Angle* and click **OK**.

8. **Rename** the other submodel to *Torque*.

The *Editor* now should look like:




*Drag and drop models from the library to the Graphical Editor.*

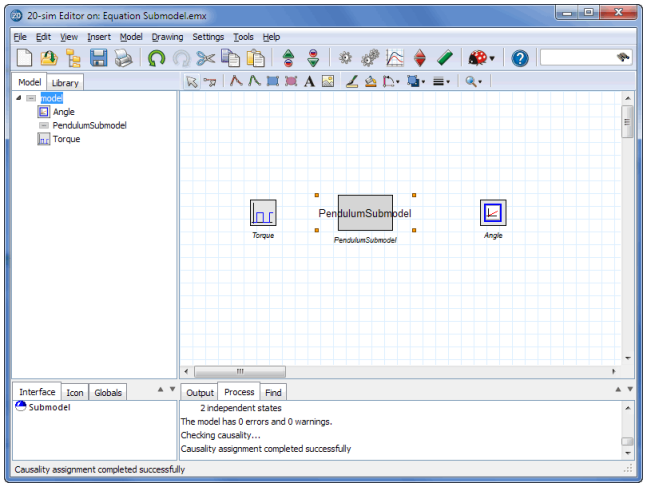
If the submodel names are not shown, select a submodel and from the **right mouse menu** choose **Show Name** and **Bottom**.

9. Put the mouse pointer in between the two submodels. From the right mouse menu or from the **Insert** menu select **Empty Submodel**.

This adds an empty submodel.

10. **Rename** the new Submodel to *PendulumSubmodel (right mouse menu - Properties*). Use the **right mouse menu** to **show the name** at the **bottom**.

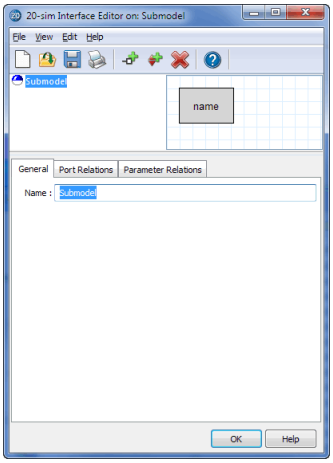
The *Editor* now should look like:




*Drag and drop models from the library to the Graphical Editor.*

11. Make sure that the *PendulumSubmodel* is **selected**.
12. From the **right mouse** select **Edit Interface**.

This will open the *Interface Editor*.




*The Interface Editor is used to define input and output signals.*

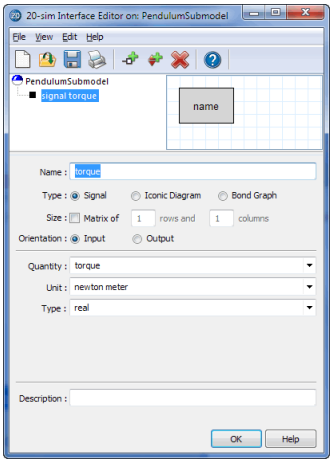
In the *Interface Editor*, you can define the model interface. We are going to define two signals: one input signal (*Torque*) and one output signal (*Angle*).

13. Change the **Name** to *PendulumSubmodel*.
14. From the **Edit** menu select **Add Port**.

15. Choose the following settings:

| Items | Values |
|---|---|
| *Name* | torque |
| *Type* | Signal |
| *Orientation* | Input |
| *Quantity* | torque |

Your *Interface Editor* should now look like:



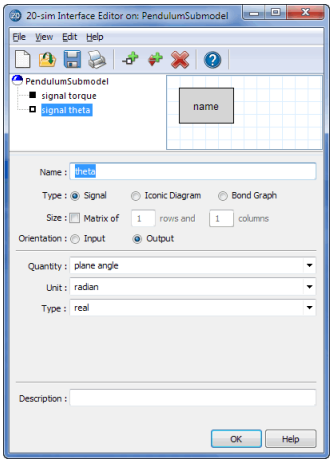

*The input signal represents a torque.*

16. From the **Edit** menu select **Add Port**.

17. Now choose the following options:

| Items | Values |
|---|---|
| *Name* | theta |
| *Type* | Signal |
| *Orientation* | Output |
| *Quantity* | plane angle |

Your *Interface Editor* should now look like:




*The output signal represents an angle.*

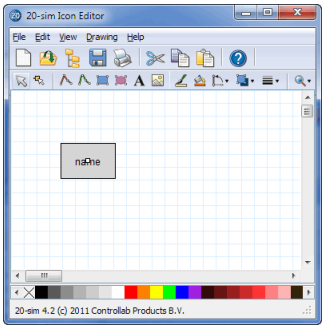
18. Close the *Interface Editor* by clicking the **OK** button.

Now we have defined the *Interface* of our equations submodel. Next we will change its appearance.

13. Make sure that the *PendulumSubmodel* is **selected**.

14. From the **right mouse** select **Edit Icon**.

Now the *Icon Editor* will be opened:




*The Icon Editor is used to create custom made model icons.*

In the *Icon Editor,* you can change the icon of the model. As you can see, a default icon has been generated. The terminals (dots in the center, indicating where the signals connections should be) are already available. We are only going to enlarge the gray square.

21. In the toolbar of the Icon Editor, click the **left arrow** to change to *selection mode*.

22. **Select** the gray box.

Four orange squares, indicating the corners of the box, will now be visible.

23. While the square is still selected, go to the colorbar at the bottom of the *Icon Editor*.

24. Point the mouse on the color yellow and click the **right mouse button**.

Now the square should have a yellow background and a black border (with the **left mouse button** the border color can be selected).

25. Select the text (name) in the middle of the square.

26. From the **right mouse menu**, click **Properties**.

A text editor pops up.

27. Change the default text to *Pendulum Submodel* and click **OK**.

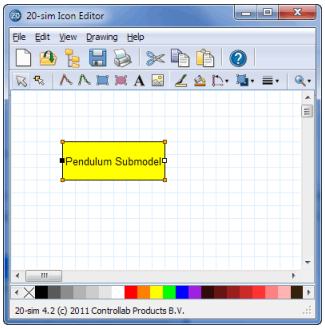
28. Select the square and **drag the corners** until it is large enough to contain the text.

29. Select the text ***Pendulum Submodel***.

30. Click the keyboard arrows to move the text to the center of the square (press the **Shift** button while dragging for fine movements).

31. Click on the small squares in the middle, representing the input signal (torque) and the output signal (angle) and **drag** them to the **borders** of the square.

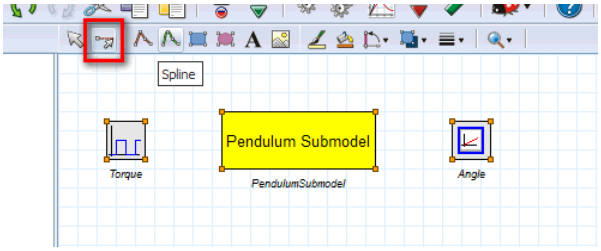
Now the icon should look like:




*The black squares indicate the connections for the input and output signal.*

32. From the **File** menu select **Exit**. A pop-up window will ask you to *update* the graph first. Choose **Yes**.

You *Editor* will now show the yellow icon. So far we have given the model an icon and an interface. now we are going to enter its implementation in the form of equations. Look at the *taskbar* at the top of the *Graphical Editor*. The *taskbar* contains buttons create connections, draw lines and more.



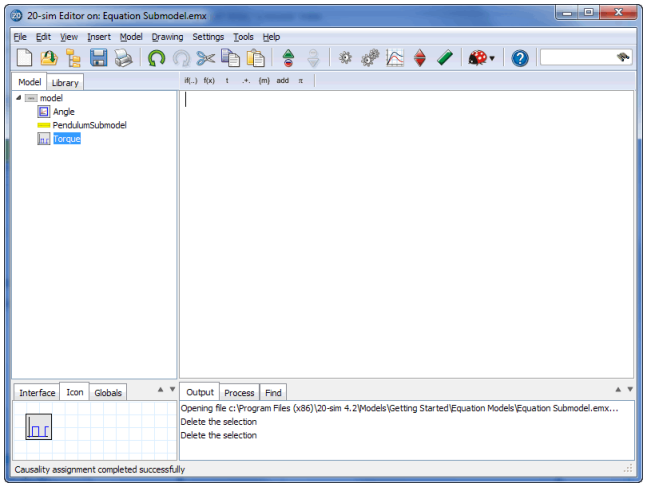

*The taskbar of the Equation Editor with the selection mode selected.*

33. In the **taskbar**, choose the **left button** . This is the *selection mode* button.
34. **Select** the *PendulumSubmodel*.
35. From the **Model** menu select the **Go Down** command.

The submodel has no implementation. A window will therefore pop-up, asking you what kind of implementation you would like.

36. Select **equation submodel**.

Now an empty *Equation Editor* will be shown:



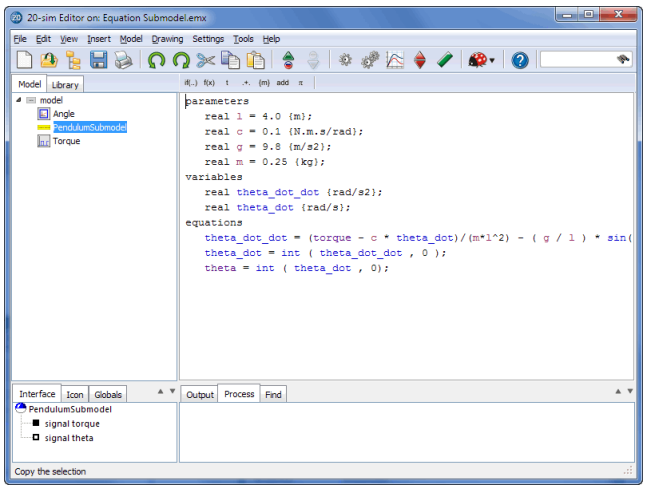

*The submodel has an empty implementation.*

We will enter the differential equations that where at the start of the topic.

37. **Copy** the following equations to the **Equation Editor**.

```
parameters
    real l = 4.0 {m};
    real c = 0.1 {N.m.s/rad};
    real g = 9.8 {m/s2};
    real m = 0.25 {kg};
variables
    real theta_dot_dot {rad/s2};
    real theta_dot {rad/s};
equations
    theta_dot_dot = (torque - c * theta_dot)/(m*l^2) - ( g / l ) * sin( theta );
    theta_dot = int ( theta_dot_dot , 0 );
    theta = int ( theta_dot , 0);
```

You model should now look like:




*The submodel is implemented by equations.*

38. From the **Model** menu select the **Check Submodel** command. Now the submodel will be checked. If any errors are found, a message window pops up.

39. Return to the main model level of the hierarchy: From the **Model** menu select the **Go Up** command.

Now we have completed our equation submodel. The only thing that is left is to connect it to the other models.

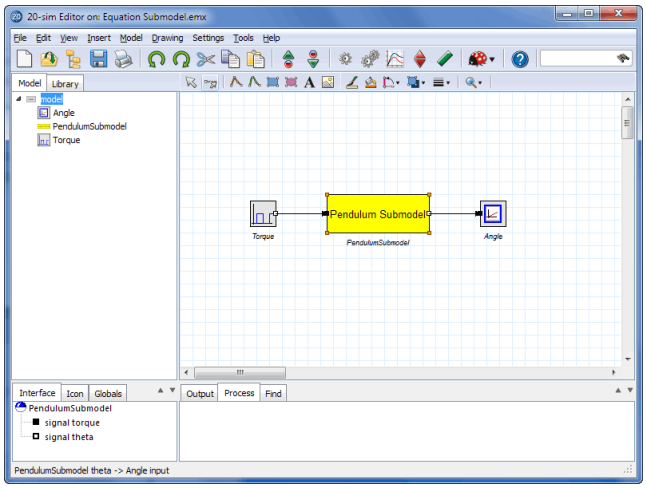
40. In the **taskbar**, choose the **second left button**  . This is the *connection mode* button.

41. **Select** the submodel ***Torque*** and then the submodel ***PendulumSubmodel***.

Now a signal should be visible pointing from *Torque* to *PendulumSubmodel*.

42. **Connect** the submodels *PendulumSubmodel* and *Angle*.

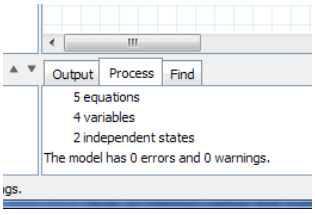
Now your model should look like:




*The equation submodel connected to the other models.*

43. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings:




*The results of a model check are shown in the Output tab.*

If any errors are found, a message window pops up, showing the errors which 20-sim has found. You can click on the error in the *Process tab* to highlight the corresponding equation.

44. If any errors occur, try to solve them.

45. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *EquationSubmodel.emx*.

If you had problems running a simulation, load the model *Equation Submodel* from the *Getting Started\Equation Models* section of the library.

## Simulation

We have entered the model and proceed with the simulation.

46. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.

This opens the *Simulator* window. We will proceed with this window.

47. In the *Simulator* toolbar from **the Properties** menu select the **Parameters** command and change the default parameter values to:

| | |
|---|---|
| *Torque\Amplitude* | *4* |
| *Torque\omega* | *0.1* |
| *PendulumSubmodel\l* | *4* |
| *PendulumSubmodel\c* | *0.1* |
| *PendulumSubmodel\g* | *9.8* |
| *PendulumSubmodel\m* | *0.25* |

Note that 20-sim will sometimes use prefixes like *m* (= 0.001), when working with units. The value of *PendulumSubmodel\c* may therefore be indicated as 100 m N.m.s./rad.

48. From the **Properties** menu select the **Run** command and change the default values to:

| | |
|---|---|
| Start | 0 |
| Finish | 100 |
| Method | Runge-Kutta 4 |
| Step Size | 0.1 |

49. From the **Properties** menu select the **Plot** command.

50. Select the **Plot Properties** tab and change the **Plot Title** to *Equation Submodel*.

51. Select the **Y-axis** tab.

The variable Angle\plot is automatically selected.

52. Change the label to *Angle*.

53. Click the **Add Curve** button.

This opens the *Variable Chooser* which helps you to select another variable that should be plotted.

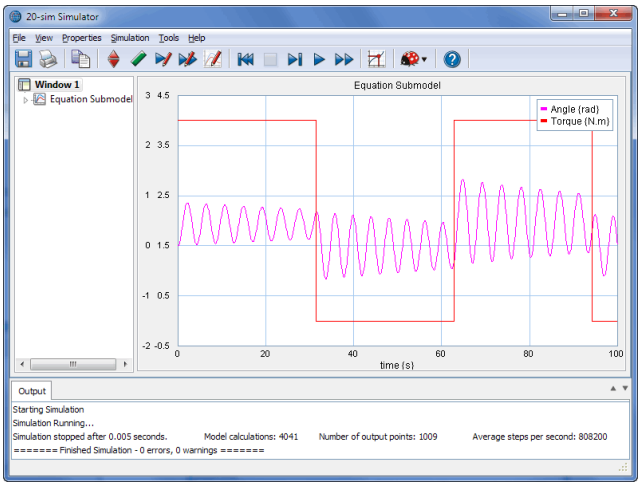
54. Select the variable ***Torque\output*** from the list and click the **OK** button.

55. Change the label to *Torque*.

56. **De-select** the *Shared Y-Axes* option.

57. Close the *Plot Properties Editor* by clicking the **OK** button.

58. From the **Simulation** menu select the **Run** command to start the simulation. The result should look like the figure below.




*The simulation results of the equation submodel.*

59. From the **File** menu click **Save**.

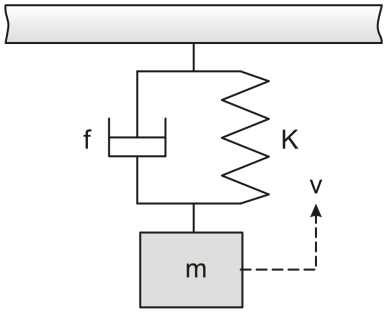
If you had problems running a simulation, load the model *Equation Submodel* from the *GettingStarted\Equation Models* section of the library.

# 6 Block Diagrams

## 6.1 Block Diagram Mainmodel

### Differential Equations

We consider the mechanical system that was also use for the equation main model. In this lesson on you will learn how to describe this system with a block diagram.




*A mass with spring and damper.*

The mechanical system can be described by the following set of differential equations:

$$F_m = -m \cdot g - F_s - F_d$$

$$v = \frac{1}{m} \int F_m \, dt$$

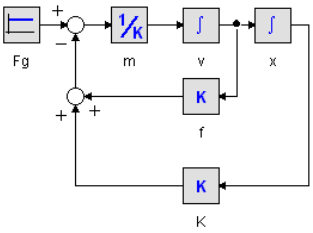
$$F_s = K \cdot x = K \int v \, dt$$

$$F_d = f_v$$

### Block Diagram Model

#### Drag and Drop

These equations can be entered in 20-sim using a block diagram. We want to obtain the following block diagram.




*Block Diagram model of the mechanical system.*

1. Open 20-sim and select **File**, **New** and **Graphical Model** and make sure that 20-sim is in **Debug Mode** .

The right part of the *Editor* will now allow you to enter graphical models. That is why we have named this part of the *Editor* the *Graphical Editor*.

2. Go to the left of the *Editor* and click the **Library tab**.

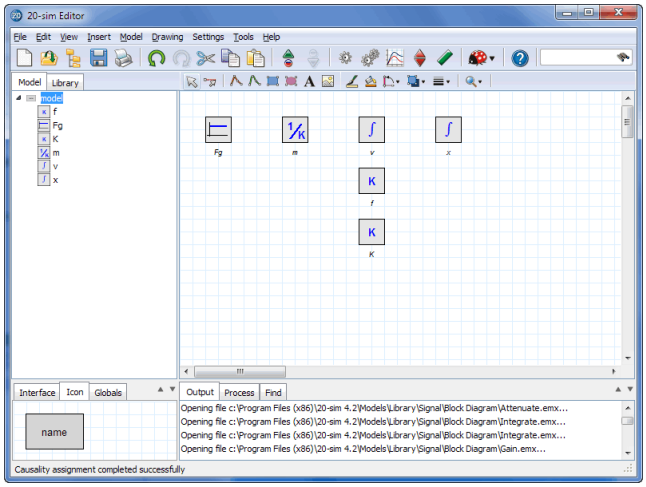
3. Drag the following library models to the *Graphical Editor*:

| **model library** | **model** |
|---|---|
| *Library\Signal\Sources* | *Constant* |
| *Library\Signal\Block Diagram* | *Attenuate* |
| *Library\Signal\Block Diagram* | *Integrate (2×)* |
| *Library\Signal\Block Diagram* | *Gain (2×)* |

4. Select the *Constant* model. From the **Model** menu select the **Properties** command. This will open the *Model Properties Editor*.

5. **Rename** the model to *Fg*.

6. Do so with all the models until your *Editor* looks like:




*You can give block diagram elements useful names.*

## Connections

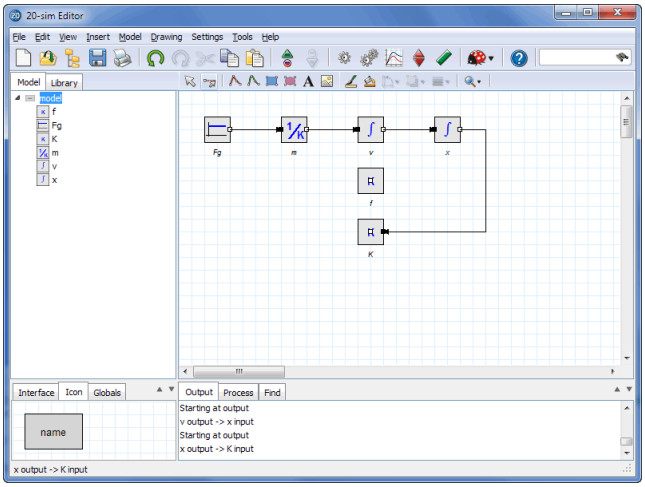
7. In the **taskbar**, choose the **second left button** (the mouse pointer with line). This is the *connection mode* button .

8. Select the model ***Fg*** (left mouse click on top of the *Fg* model) and then the model ***m*** (left mouse click on top of the *m* model).

Now a signal should be visible pointing from Fg to m.

9. Enter the other connections until your *Editor* looks like the figure below. You can make intermediate points (the corners of a connection) by quickly clicking the left or right mouse button, while dragging.

If you have problems making connections please note that there are two way to make a connection: **tapping** mode and **pressing** mode. With the **tapping** mode you click the left mouse button at the first model (do not keep it pressed but quickly "tap" the button) and keep tapping until the second model is reached. With the **pressing** mode you press left mouse button on the first model and keep it pressed until the second model is reached. Intermediate points in pressing mode are made by clicking the right mouse button.




*Using the connection model button, you can create connections between elements.*

If you want to change there position of the models or connections, you have to switch to the *selection mode* button. You can switch to selection mode by clicking **left button** (the mouse pointer) of the **taskbar**. Make sure to switch back to *connection mode* to make the other connections.

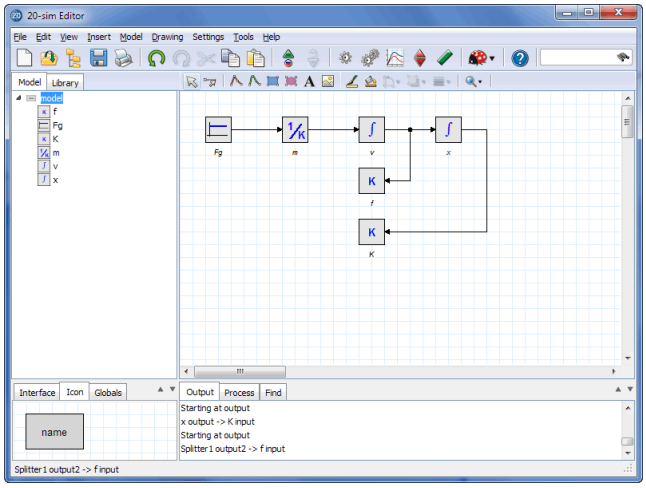
## Splitters

10. Click on the **middle** of the signal pointing from the submodel *v* to *x*.

A splitter will be inserted.

11. Make a **connection** from the **splitter** to the model ***f***.

Your model should look like:



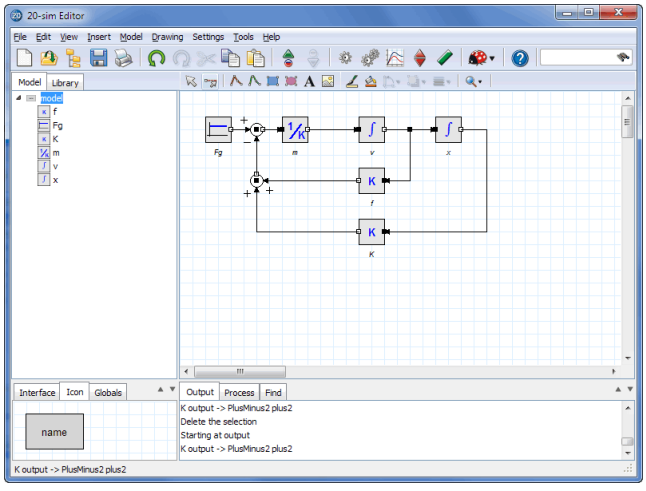

*Click on a signal to insert splitters.*

## Plus Minus

*12.* Click on the *K* model to start a connection and then click on the **middle** of the signal pointing from the submodel *Fg* to *m.*

*13.* 20-sim will open a dialog asking you to insert a multiplication or addition. Select the **PlusMinus** option and select the **minus** sign.

*14.* **Repeat** this action by making a connection from the *f* model to make your *Editor* look like:




*if you end a connection on another connection 20-sim will insert a multiplication or addition.*

### Compiling

15. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings. If any errors are found, a message window pops up, showing the errors which 20-sim has found. You can click on the error in the *Process tab* to highlight the corresponding equation.

16. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *Block Diagram Mainmodel.emx*.

If you have problems creating the model, load the model *Block Diagram Mainmodel* from the *Getting Started\Block Diagrams* section of the library.

## Simulation

17. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.

This opens the *Simulator* window. We will proceed with this window.

18. In the *Simulator* toolbar from **the Properties** menu select the **Parameters** command and change the default parameter values to:

| | |
|---|---|
| *Fg\C* | -9.8 |
| *m\K* | 1 |
| *f\K* | 1 |
| *K\K* | 2 |

19. From the **Properties** menu select the **Run** command and change the default values to:

| | |
|---|---|
| *Start* | 0 |
| *Finish* | 15 |
| *Method* | Euler |
| *Step Size* | 0.2 |

20. From the **Properties** menu select the **Plot** command.

21. Select the **Plot Properties** tab and change the **Plot Title** to *Block Diagram Mainmodel*.

22. Select the **Y-axis** tab and click **Choose** to open the Variable Chooser.

22. Select the variable *x\output* from the list and click **OK**.

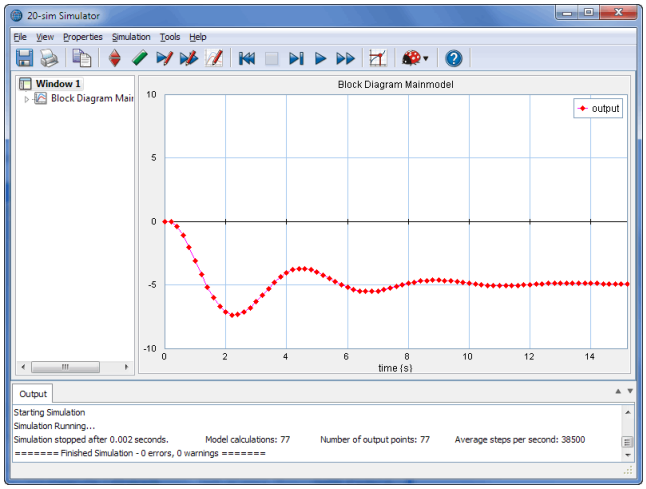
23. Set the following values:

| **Tick Style Properties** | |
|---|---|
| *Tick Style* | Diamond Closed |
| *Min. Distance (pixels)* | 2 |
| *Color* | Red |

| **Scaling** | |
|---|---|
| *Scaling* | Manual |
| *From* | -10 |
| *To* | 10 |

25. Close the *Plot Properties Editor* by clicking the **OK** button.

26. From the **Simulation** menu select the **Run** command to start the simulation. The result should look like the figure below.




*The simulation result.*

27. From the **File** menu click **Save**.

If you had problems running a simulation, load the model *Block Diagram Mainmodel* from the *Getting Started\Block Diagrams* section of the library.

## 6.2 Block Diagram Submodel

### Differential Equations

In a previous lesson we have found that a pendulum can be described by the differential equations:

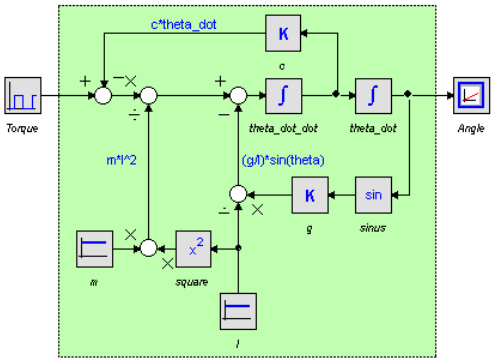
*theta_dot_dot = (torque - c·theta_dot)/(m*l2) - (g/l)·sin(theta);*

*theta_dot = int(theta_dot_dot);*

*theta = int(theta_dot);*

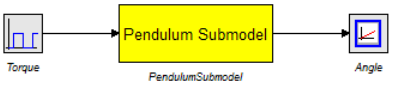
## Block Diagram Model

The input variable is the applied torque and the output variable is the pendulum angle theta. We can represent these equations by a block diagram:




*Block diagram model of a pendulum.*

In the block diagram a square wave generator acts as torque and the signal monitor block catches the output angle theta. Block diagrams tend to grow complex easily when more elements are involved. To keep a good overview, hierarchy must be used. In this model the elements of the pendulum part (inside the green square) will be hidden in a submodel as shown below. Still the square wave generator will act as torque and the signal monitor block will catch the output angle theta.




*Pendulum as a block diagram submodel.*

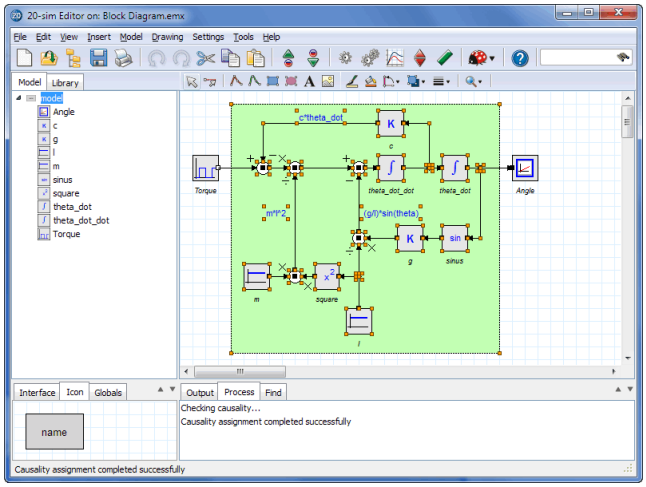
1. **Open** 20-sim.
2. Make sure that 20-sim is in **Debug Mode**.
3. Try to create the block diagram at the top of this topic yourself or load the model *Block Diagram* from the *Getting Started**Block Diagrams* section of the library.

To make a submodel out of all the elements in the green square we first have to select all these elements. A multiple selection (i.e. more than one element selected) is easy in 20-sim. Two methods can be used:

- Keep the Shift key pressed while you click your mouse pointer on various elements.
- Press your left mouse button and keep it pressed while you drag the mouse pointer diagonal down (you are creating a square). As you move the mouse, you will see a square and every element in the square will be selected. Now you can release the left mouse button.

4. In the *Editor* **select all elements** that are in the **green square**.

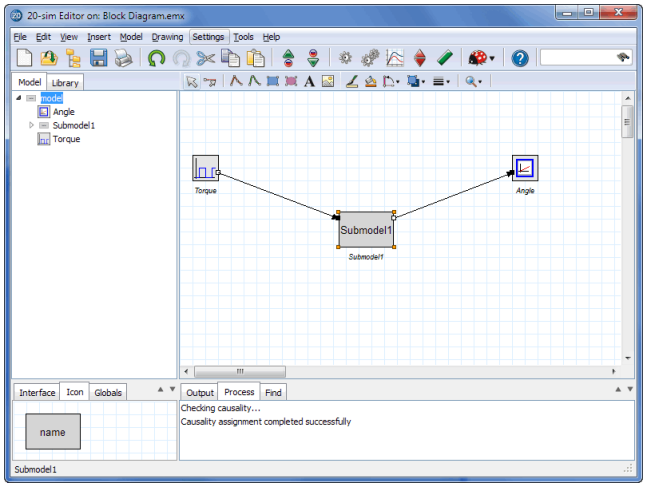
Your Editor should now look like (only the models Torque and Angle not selected):




*Drag a rectangle over an area to select multiple element.*

5. From the **Edit** menu select **Implode**.

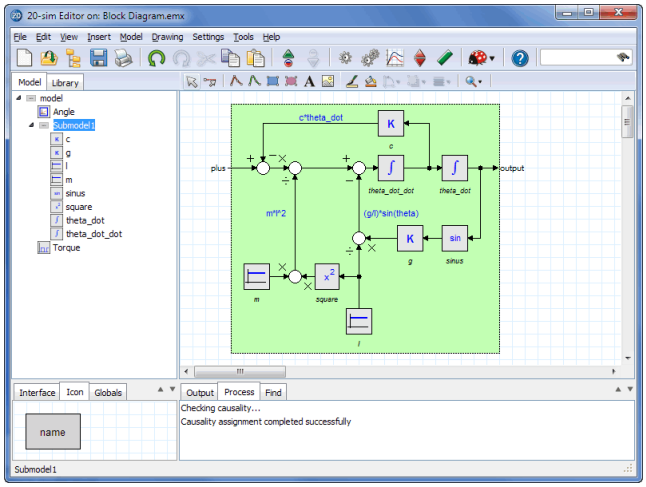
This will make all selected elements, part of a new submodel. Your model should now look like:




*Use the Implode command to combine the selected elements into one submodel.*

6. Select the new submodel and click **Go Down** (**Model** menu).

This command will show us the inside of the submodel. It should look like:




*The implementation of the submodel is a block diagram.*

As you can see, 20-sim has added ports (*plus* and *output*) automatically that connect the submodel to the outside world.

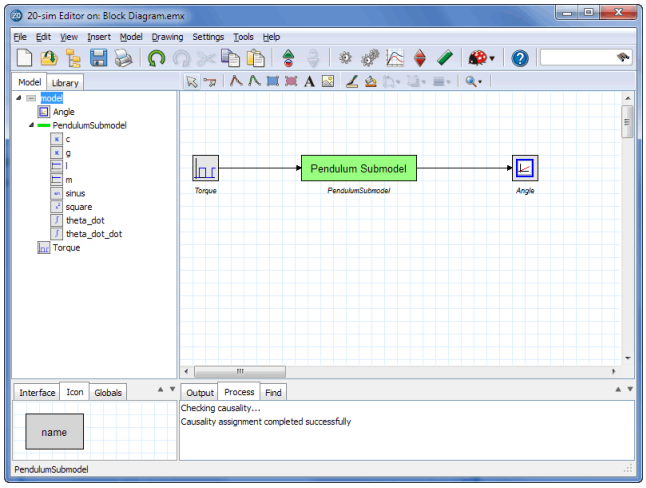
7. Click **Go Up** (**Model** menu) to go to the highest model level.

8. Select the new Submodel and change its name to *PendulumSubmodel* (select **Properties** from the **right mouse menu**).

9. **Select** the new Submodel and open the **Icon Editor** (select **Edit Icon** from the **right mouse menu**).

In the previous topic is explained how you can use the Icon Editor to change the appearance of a submodel.

10. Change the icon until it looks like:




*With the Icon Editor you can create custom made model icons.*

11. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings. If any errors are found, a message window pops up, showing the errors which 20-sim has found. You can click on the error in the *Process tab* to highlight the corresponding equation.

12. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *Block Diagram Submodel.emx*.

If you had problems running a simulation, load the model *Block Diagram Submodel* from the *Getting Started\Block Diagrams* section of the library.

## Simulation

We have entered the model and proceed with the simulation.

13. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.

This opens the *Simulator* window. We will proceed with this window.

14. In the *Simulator* toolbar from **the Properties** menu select the **Parameters** command and change the default parameter values to:

| | |
|---|---|
| *PendulumSubmodel\c\K* | 0.1 |
| *PendulumSubmodel\g\K* | 9.8 |
| *PendulumSubmodel\l\C* | 4 |
| *PendulumSubmodel\m\C* | 0.25 |
| *Torque\amplitude* | 4 |
| *Torque\omega* | 0.1 |

15. From the **Properties** menu select the **Run** command and change the default values to:

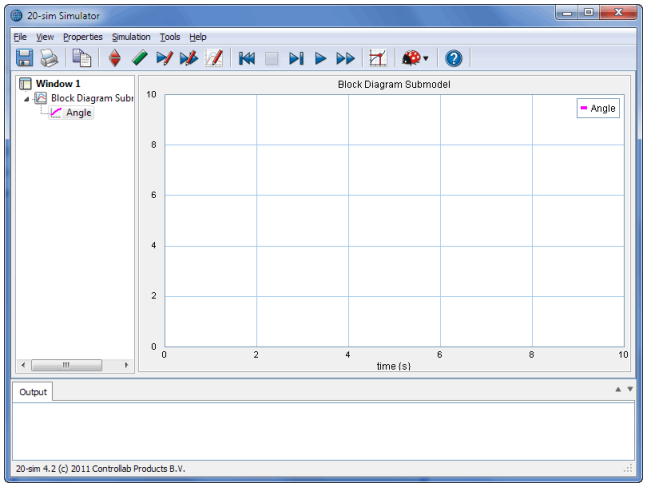
| | |
|---|---|
| *Start* | 0 |
| *Finish* | 100 |
| *Method* | Runge-Kutta 4 |
| *Step Size* | 0.1 |

16. From the **Properties** menu select the **Plot** command.
17. Select the **Plot Properties** tab and change the **Plot Title** to **Block Diagram Submodel**.
18. Select the **Y-axis** tab.

The variable *Angle\plot* is automatically selected.

*19.* Change the **Label** to ***Angle*** and click **OK** close the *Plot Properties.*

Now your *Simulator* should look like:

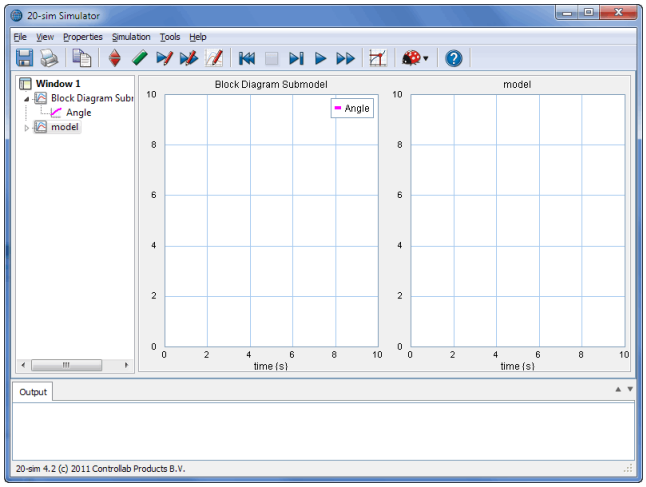



*The Simulator with one plot and one curve entered.*

*20.* In the tree at the left click on ***Window 1*** to select it.

*21.* From the right mouse menu select **Add Plot** - **Plot**.

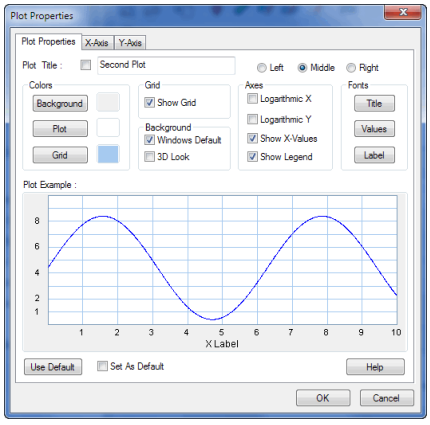
Now a second plot will be visible.




*The Simulator with the second plot entered.*

22. In the tree, select the second plot (**model**). From the **right mouse menu**, select **Plot Properties**.
23. Click on the **Choose** button to select the variable *Torque\output* and change the **Label** to *Torque*.
24. Select the **Plot Properties** tab and change the **Plot Title** to **Second Plot**.
25. De-select the **Plot Title** option.

Now it should look like:

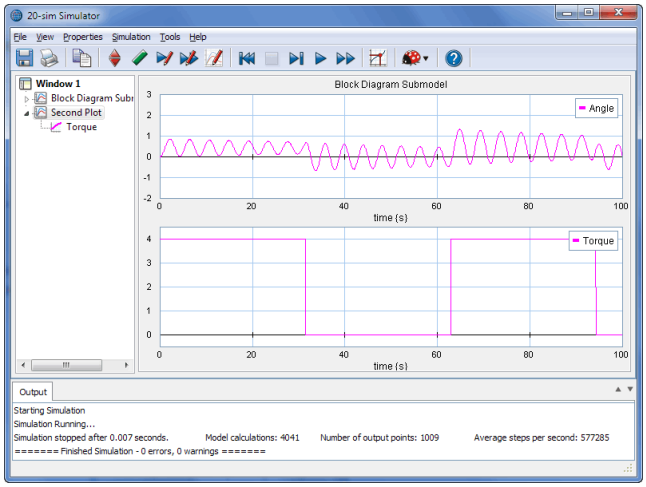



*Entering a plot title and hiding it for display.*

26. **Close** the Plot Properties by clicking **OK**.

27. In the tree at the left click on ***Window1*** to select it.
28. From the right mouse menu select **Tile Plots** - **Tile Vertical**.
29. **Run** a simulation

Now the Simulator should look like:




*The simulation results of the block diagram submodel.*

30. From the **File** menu click **Save**.

If you had problems running a simulation, load the model *Block Diagram Submodel* from the *Getting Started\Block Diagrams* section of the library.

# 7 Iconic Diagrams

## 7.1 Iconic Diagram (Electric)

In the previous lessons you have learned the basics of 20-sim and how to enter equation models and block diagram models. In this lesson you will learn how to enter iconic diagram models. In the first part an electric network will be entered and simulated. With help of this model, the importance of the **View** menu for iconic diagram models will be explained. After that a mechanical system will be entered and simulated.

1. Open 20-sim and select **File**, **New** and **Graphical Model** and make sure that 20-sim is in *Debug Mode* .

The right part of the *Editor* will now allow you to enter graphical models. That is why we have named this part of the *Editor* the *Graphical Editor*.

2. Go to the left of the *Editor* and click the **Library tab**.

3. Drag the following library model to the *Graphical Editor*:

| **model library** | **model** |
|---|---|
| *Library\Iconic Diagrams\Electric\Sources* | *VoltageSource (DC)* |

The *VoltageSource* model has multiple implementations: *DC* and *AC*. When you drag and drop the model to the Graphical editor, you are asked which implementation should be used.

4. Choose the implementation **DC**.

You can always change the implementation by using the right mouse menu: select the model, right mouse menu, choose implementation.

5. Drag the following library model to the *Graphical Editor*:

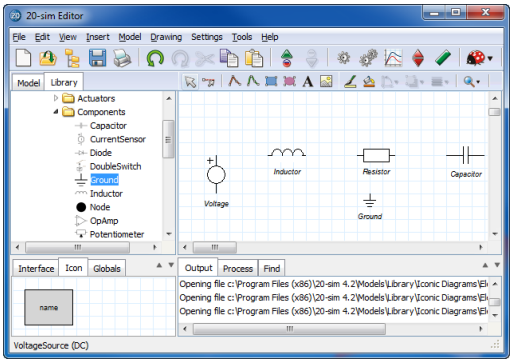
| **model library** | **model** |
|---|---|
| *Library\Iconic Diagrams\Electric\Sources* | *VoltageSource* |
| *Library\Iconic Diagrams\Electric\Components* | *Inductor* |
| *Library\Iconic Diagrams\Electric\Components* | *Resistor* |
| *Library\Iconic Diagrams\Electric\Components* | Capacitor |
| *Library\Iconic Diagrams\Electric\Components* | *Ground* |

6. Select the *VoltageSource* model. From the **Settings** menu select the **Submodel** command.

This will open the *Model Properties Editor*.

7. **Rename** the model to *Voltage*.

8. Do so with all the models until your *Editor* looks like:

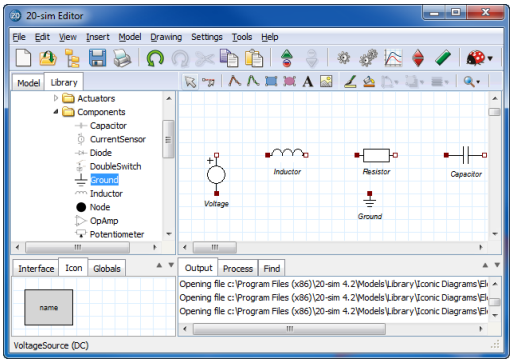



*Drag and drop models from the library.*

As in the figure above, leave a lot of space between the models. Otherwise some effects, explained later in this topic, might be hard to spot. Every iconic diagram model has one or more **terminals** to allow connections with other models.

9. To show these terminals, from the **View** menu click **Show Terminals**.

The terminals will be visible as open and closed rectangles:



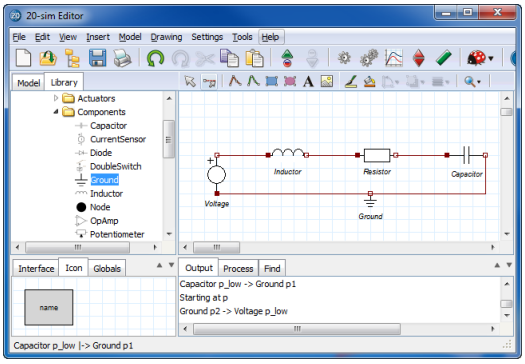

*Use the View menu to show the terminals of the models.*

10. In the **taskbar**, choose the **second left button** (the mouse pointer with line). This is the *connection mode* button.

11. Select the model *Voltage* and then the model *Inductor.*

Now a connection will be created between the two submodels.

12. Enter the other connections until your *Editor* looks like the figure below. You can make intermediate points (the corners of a connection) by clicking the right mouse button, while dragging.




*The complete model.*

13. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings. If any errors are found, a message window pops up, showing the errors which 20-sim has found. You can click on the error in the *Process tab* to highlight the corresponding equation.

14. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *IconicDiagramModelElectric.emx*.

If you have problems creating the model, load the model *Iconic Diagram Electric* from the *Getting Started\Iconic Diagrams* section of the library.

## Simulation

15. Select the **Start Simulator** command from the **Model** menu in the *Editor* window.

This opens the *Simulator* window. We will proceed with this window.

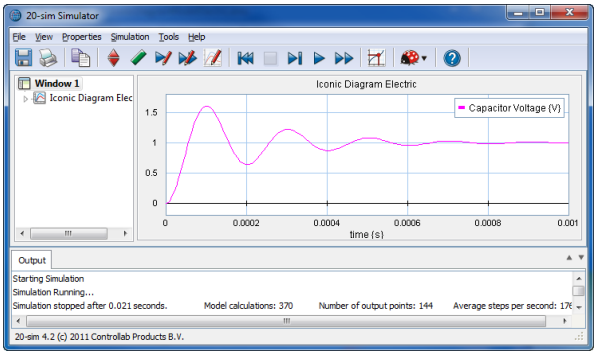
16. In the *Simulator* window, select from the **Properties** menu the **Parameters** command and change the default parameter values to:

| | |
|---|---|
| *Voltage\U* | 1V |
| *CapacitorC* | 1u |
| *Resistor\R* | 10 |
| *Inductor\|L* | 1m |

17. From the **Properties** menu select the **Run** command and change the default values to:

| | |
|---|---|
| *Start* | 0 |
| *Finish* | 1e-3 |
| *Method* | Backward Differentiation Formula (BDF) |

18. From the **Properties** menu select the **Plot** command.
19. Select the **Plot Properties** tab and change title to **Iconic Diagram Electric**.
20. Select the **Y-axis** tab and select the variable *Capacitor\p.u* for plotting.
21. Change the **label** to **Capacitor Voltage.**
22. Close the *Plot Properties Editor* by clicking the **OK** button.
23. From the **Simulation** menu select the **Run** command to start the simulation. The result should look like the figure below.




*The simulation results for the electric circuit.*

As you can see, creating a model and starting a simulation is easy in 20-sim. If you have problems creating the model, load the model *Iconic Diagram Electric* from the *Getting Started\Iconic Diagrams* section of the library.

## 7.2 View Menu

In this topic we will give some more insight into the model of the electric circuit.

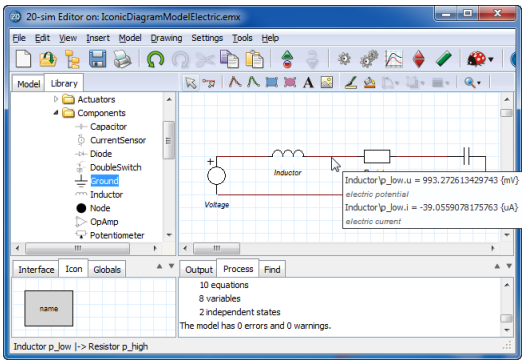
1. If you had problems creating this model, **load** the model ***Iconic Diagram Electric*** from the *Getting Started\Iconic Diagrams* section of the library.
2. Go to the ***Editor***.

### Across and Through

In iconic diagrams, connections describe the flow of energy from one model to another. This flow of energy can be characterized by two variables, of which the product is power. These variables are called **across** (**a**) and **through** (**t**). The **across** and **through** variables make up a combination that is typical for a physical domain. For electrical networks these variables are **voltage** and **current**. In 20-sim we can inspect the **across** and **through** variable of every connection.

3. Put your **mouse pointer** on top of a **connection**, until a little window pops up.

It should look like the figure below (don't worry when the values are different, these are just example values):




*If you put your mouse pointer on top of a connection, the corresponding values are shown.*

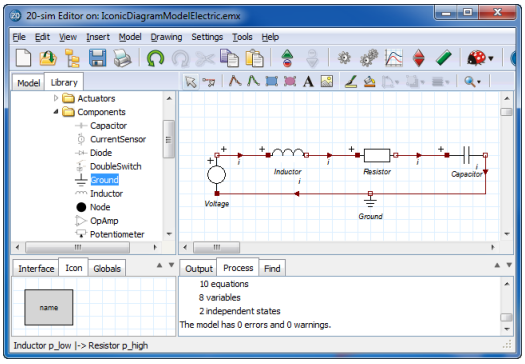
As you can see, the connection has a voltage (across) of 993 [mV] and a current (through) -39 [uA]. Across variables (voltages in this model) in the top level of a model, are always defined with respect to a single global reference of 0 V. This means that the voltage is defined with respect to the ground.

Through variables, like the current in this model, are always defined with respect to the models they are connected with. 20-sim automatically assigns an orientation for these through variables. This orientation can be made visible by selecting the *Orientation Info* command of the View Menu.

4. From the **View menu** enable the option **Show Terminals** (click it until the option is enabled).

5. From the **View menu** click the option **Orientation Info** (click it until the option is enabled).

Your model should look like:



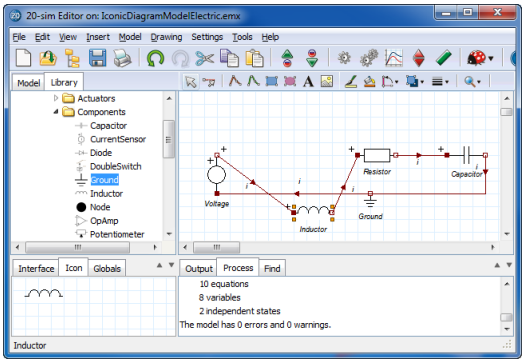

*With the View menu you can show the orientation that 20-sim has chosen for positive currents.*

The arrows indicate the direction of the **positive** currents. Now you will understand why open and closed rectangles are used to indicate the **terminals**. Open rectangles will have an outward oriented positive current and closed rectangles will have an inward oriented positive current. In the previous figure of this topic we saw a current value of i = -39 [uA]. With the given orientation this means a current i = 39 [uA] flowing from the resistor to the inductor.

## Internal Description

6. From the **View** menu select **Port Names** Info.
7. Drag the *Inductor* model to the bottom, to have a better view of the port names.

Your model should have all ports (input signals, output signals or power ports) indicated by their name:




*With the View menu you can show the port names.*

8. Select the model *Inductor*.
9. Click the **F1** key or from the **Help** menu click **Current Selection** to open the help file.

As the help file explains this model has one port *p* with two *terminals p_high* and *p_low*. These *terminals* are the points where the connections with the other models are made. Therefore *p_high.u* and *p_low.u* are the voltages on both sides of the inductor (the plus sign in the model indicates the high voltage):

```
p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;
```

*p.u* is the internal voltage difference and *p.i* is the current that flows through the inductor. The internal equation of the inductor is

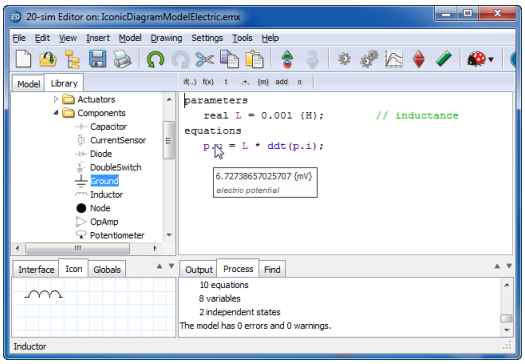
```
p.i = (1/L)int(p.u);
```

10. From the **Model** menu click **Go Down**.

Now we see the internal model description. In 20-sim we can inspect the value of every variable in an equation model.

11. Put your **mouse pointer** on top of a variable, until a little window pops-up.

It should look like the figure below (don't worry when this value is different, this is just an example value):



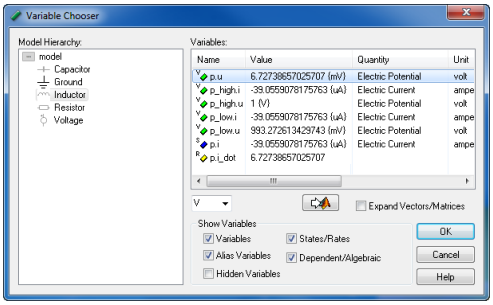

*Put your mouse pointer on any variable, to inspect its value.*

As you can see the voltage *p.u* is 6.72 [mV] which means:

*p.u = p_high.u - p_low.u = 6.72 [mV];*

12. From the **Model** menu click **Go Up**.
13. From the **Model** menu select **Show Variables**.
14. In the *Variable Chooser* window, select the *Inductor* model.

Now you can see all the variables of the inductor model.



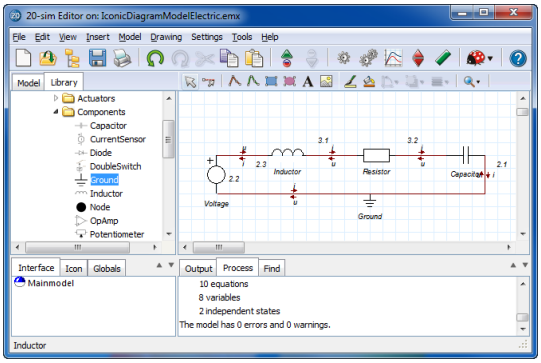

*Variable values can also be inspected in the Variable Chooser.*

Note that *p_low.u* = 993 [mV] (equal to the voltage we found at the start of this topic), *p_high.u* = 1.0000 [V] and again *p.u* = 6.72 [mV].

## Causality

15. Close the *Variable Chooser*.
16. From the **View** menu de-select all options and select **Causality Info**.

Your model should look like:




*The causal order of the model equations can be visualized with the Causality Info option.*

The arrows at the connections show the computational direction of the voltages and currents. This computational direction is called *causality*. The numbers show the order in which the causality is computed in 20-sim. For the *Inductor* model this means that at both sides current is computed as function of the voltages. Combining the previous findings, we can conclude that the following equations are derived:

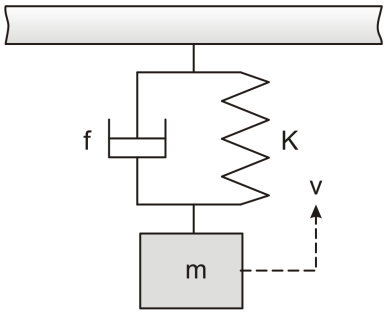
```
p.u = p_high.u - p_low.u
p.i = (1/L)*int(p.u);
p_high.i = p.i;
p_low.i = p.i;
```

As you see with *Causality Info* command you can manually inspect computational order of your model equations. This might be helpful when 20-sim has problems finding the correct computational order and you have to find the cause of these problems.

As you have seen the *View* menu of the 20-sim Editor is an important aid for understanding an iconic diagram. We advise you to use the options of this menu, as much as possible, when creating iconic diagrams. In the next section we will create a simple mechanical system. We will apply the options of the *View* menu, to correctly interpret the simulation results.

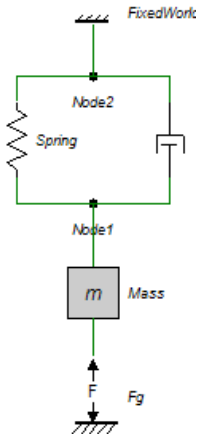
## 7.3 Iconic Diagram (Mechanical)

In this topic you will learn how to enter a simple mechanical system in the form of an Iconic Diagram. We consider the mechanical system of the figure below.




*Spring damper system.*

This system can easily be transferred into the iconic diagram model shown below.




*The equivalent 20-sim model.*

1. Open 20-sim and select **File**, **New** and **Graphical Model** and make sure that 20-sim is in *Debug Mode* .

The right part of the *Editor* will now allow you to enter graphical models. That is why we have named this part of the *Editor* the *Graphical Editor*.

2. Put the mouse pointer in the middle of the *Graphical Editor*.
3. From the right mouse menu or from the **Insert** menu select **Insert**, **Knot** and **Node**.

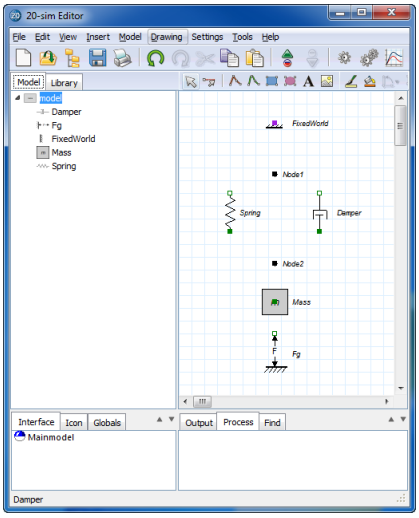
This adds a an iconic diagram *node*.

4. **Insert** a second ***node***.

5. **Drag** the following library models to the *Graphical Editor*:

| **model library** | **model** |
|---|---|
| *Library\Iconic Diagrams\Mechanical\Translation\Components* | *Damper* |
| *Library\Iconic Diagrams\Mechanical\Translation\Components* | *FixedWorld* |
| *Library\Iconic Diagrams\Mechanical\Translation\Components* | *Mass* |
| *Library\Iconic Diagrams\Mechanical\Translation\Components* | *Spring* |
| *Library\Iconic Diagrams\Mechanical\Translation\Actuators* | *Force* |

6. **Select** the *Force* model and **rename** it to *Fg* (select **Settings** menu and **Submodel** command).
7. Do so with all the submodels and use the names given in the iconic diagram at the top of this topic.
8. **Select** the *FixedWorld* submodel.
9. From the **Drawing** menu select the **Rotate Left** command.
10. Do the same for the *Spring*, *Damper* and *Fg* submodels.
11. **Select** the *Fg* submodel. From the right mouse menu select **Show Name** and **Right**.
12. Do so with all the other submodels.
13. From the **View** menu select **Show Terminals**. Now your model should look like:

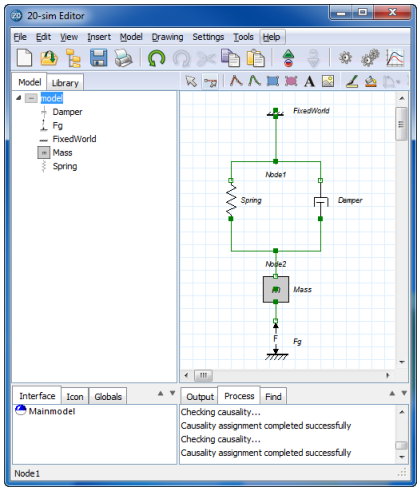



*All the models have been dropped in the Graphical Editor.*

14. In the **taskbar**, choose the **second left button** (the mouse pointer with line). This is the *connection mode* button.
15. **Select** the model *Fg* and then the model *Mass.*

Now a connection will be created between the two submodels.

16. Enter the other connections until your model looks like:

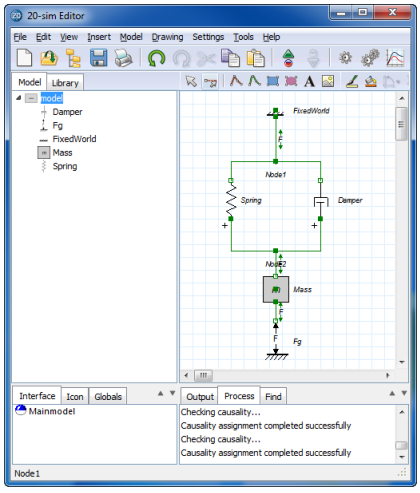



*With the proper connections the model is ready for simulation.*

The connections of an iconic diagram represent the power flow between the components. The power flow in iconic diagrams is always characterized by two variables, across and through (in this model velocity and force). The velocities are all defined with respect to a general reference (fixed world, v = 0). The forces are defined with respect to the various components.

17. From the **View** menu select **Orientation Info**.

Your model should look like:




*With the View menu you can show the orientation that 20-sim has chosen for positive forces.*

The orientation of the forces is now shown in the Editor by the little arrows →F← and →F←. When the arrows point inwards →F←, this means a positive force will pull both ends of the connection together. When the arrows point outwards ←F→, this means a positive force will push both ends of the connection outwards.

The connection between the *Fg* model and the *Mass* model show outward pointing arrows ←F→ . This means that a positive force will push the mass upwards and a negative force will pull the mass down. To generate a pulling gravity force we will therefore use a negative force value in the next topic.

18. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings. If any errors are found, a message window pops up, showing the errors which 20-sim has found. You can click on the error in the *Process tab* to highlight the corresponding equation or submodel.

19. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *IconicDiagramModelMechanical.emx*.

If you have problems creating the model, load the model *Iconic Diagram Mechanical* from the *Getting Started\Iconic Diagrams* section of the library.

## Simulation

20. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.

This opens the *Simulator* window. We will proceed with this window.

21. In the *Simulator* window, select from the **Properties** menu the **Parameters** command and change the default parameter values to:

| | |
|---|---|
| *Damper\d* | 1 |
| *Mass\m* | 1 |
| *Spring\k* | 2 |
| *Fg\F* | -9.81 |

22. From the **Properties** menu select the **Run** command and change the default values to:

| | |
|---|---|
| *Start* | 0 |
| *Finish* | 15 |
| *Method* | Euler |
| *Step Size* | 0.2 |

23. In the From the **Properties** menu select the **Plot** command.
24. Select the **Plot Properties** tab and change the title to **Iconic Diagram Mechanical**.
25. Select the **Y-axis** tab and click **Choose** to open the Variable Chooser.
26. Select the variable *Spring\x* from the list and click **OK**.
27. Change the label to *x*.

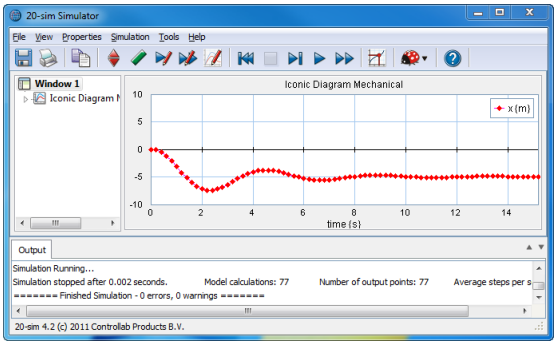
28. Set the following values:

| **Tick Style Properties** | |
|---|---|
| *Tick Style* | Diamond Closed |
| *Min. Distance (pixels)* | 2 |
| *Color* | Red |

| **Scaling** | |
|---|---|
| *Scaling* | Manual |
| *From* | -10 |
| *To* | 10 |

29. Close the *Plot Properties Editor* by clicking the **OK** button.

30. From the **Simulation** menu select the **Run** command to start the simulation. The result should look like the figure below.




*The simulation results.*

31. From the **File** menu click **Save**.

If you have problems creating the model, load the model *Iconic Diagram Mechanical* from the *Getting Started\Iconic Diagrams* section of the library.

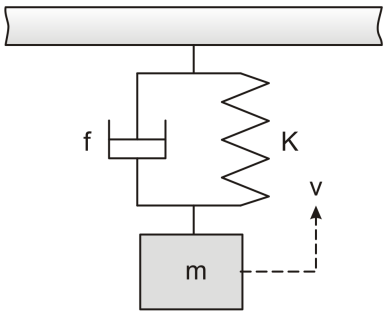
# 8 Bond Graphs

## 8.1 Bond Graph Model

In the previous sections you have learned how to enter equation models and block diagram models. You have also learned how to enter hierarchy into a model. In this lesson we will explain the basics of bond graph modeling in 20-sim. The lesson is meant for users interested in bond graph modeling. If you are not interested in bond graph modeling, we advise you to skip this section and continue with iconic diagrams.

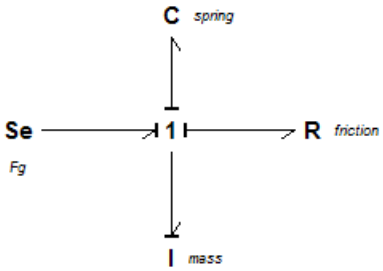
### Bond Graph Model

We consider the mechanical system of the figure below.




*A mass with spring and damper.*

This system can easily be transferred into the bond graph model shown below.




*Bond graph model of the mass-spring-damper system*

1. Open 20-sim and select **File**, **New** and **Graphical Model** and make sure that 20-sim is in *Debug Mode* .

The right part of the *Editor* will now allow you to enter graphical models. That is why we have named this part of the *Editor* the *Graphical Editor*.

2. Put the mouse pointer in the middle of the *Graphical Editor*.
3. From the right mouse menu or from the **Insert** menu select **Insert**, **Knot** and **OneJunction**.

This adds a *1*-junction.

4. Right click on this 1-junction and select the **Show Name** menu and the **None** command to hide the name of the 1-junction.

5. Go to the left of the *Editor* and click the **Library tab**.

6. Drag the following library models to the *Graphical Editor*:

| **model library** | **model** |
|---|---|
| *Library\Bond Graph* | *C* |
| *Library\Bond Graph* | *I* |
| *Library\Bond Graph* | *R* |
| *Library\Bond Graph* | *Se* |

6. Select the *Se* model. From the **Settings** menu select the **Submodel** command.

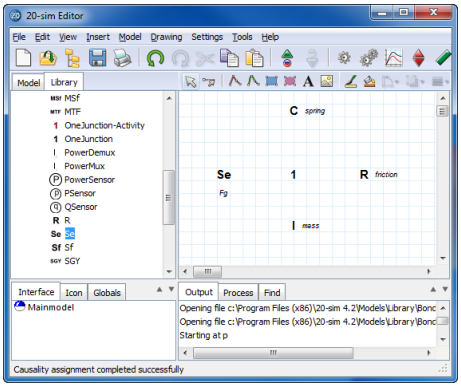
This will open the *Model Properties Editor*.

7. **Rename** the model to *Fg*. Do so with all the submodels and use the names given in the bond graph at the top of this topic.

8. **Select** the *C* model and from the right mouse select the **Show Name** menu and the **Right** command.

This will put the element name to the right of the element.

9. Do so with the other elements (except for the **Se** element) until your model looks like:

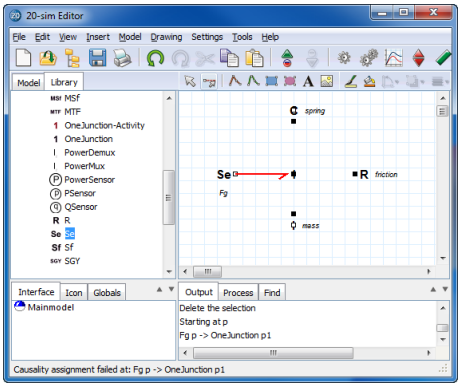



*The bond graph elements in the Editor.*

10. In the **taskbar**, choose the **second left button** (the mouse pointer with line). This is the *connection mode* button.

11. Select the model ***Fg*** (left mouse click on top of the element) and then the ***1***-junction (left mouse click on top of the element).

This will create a connection between both elements:

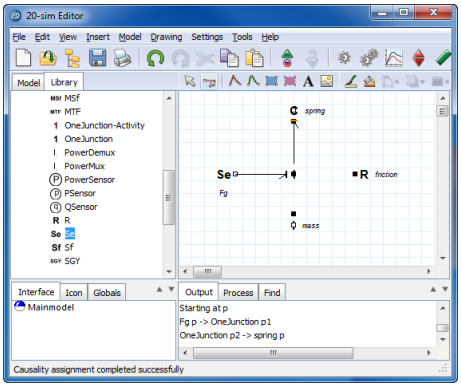



*The red color of the bond indicates a conflict. In this case more bonds need to be inserted.*

The bond is drawn red, which means causality could not be assigned. Don't worry! This is because 20-sim has an on-line causality assignment and notices there is only one bond connected to the *1*-junction.

12. Create a bond between the ***1***-junction and the submodel *spring*.

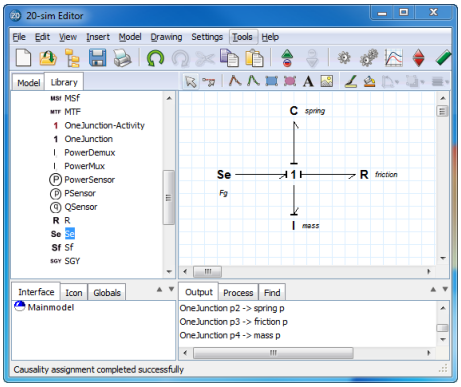
Now your model should look like:




*The orange stroke indicates a non-preferred causality.*

The bond to the ***C*** model has an orange causal stroke. This means that the *C* model in this configuration has a non-preferred causality.

13. Enter the other connections until your model looks like:




*The complete bond graph model.*

14. From the **Model** menu select the **Check Complete Model** command.

Now the complete model will be checked. If the model is correct at the bottom of the *Editor* the *Process tab* should show a message indicating 0 errors and 0 warnings. If any errors are found, a message window pops up, showing the errors which 20-sim has found. You can click on the error in the *Process tab* to highlight the corresponding equation.

15. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *BondGraphModel.emx*.

If you have problems creating the model, load the model *Bond Graph Model* from the *Getting Started\Bond Graphs* section of the library.

## Simulation

16. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.

This opens the *Simulator* window. We will proceed with this window.

17. In the *Simulator* window, select from the **Properties** menu the **Parameters** command and change the default parameter values to:

| | |
|---|---|
| *friction\r* | 1 |
| *Fg\effort* | -9.81 |
| *Mass\i* | 1 |
| *Spring\c* | 0.5 |

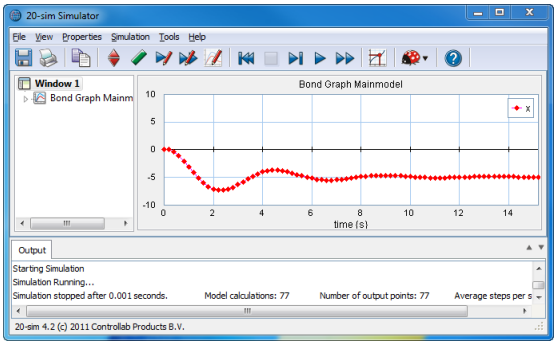
18. From the **Properties** menu select the **Run** command and change the default values to:

| | |
|---|---|
| *Start* | 0 |
| *Finish* | 15 |
| *Method* | Euler |
| *Step Size* | 0.2 |

19. Select from the **Properties** menu the **Plot** command.
20. Select the **Plot Properties** tab and change the title to **Bond Graph Mainmodel**.
21. Select the **Y-axis** tab and click **Choose** to open the Variable Chooser.
22. Select the variable *spring\state* from the list and click **OK**.
23. Change the label to *x*.
24. Set the following values:

| **Tick Style Properties** | |
|---|---|
| *Tick Style* | Diamond Closed |
| *Min. Distance (pixels)* | 2 |
| *Color* | Red |
| | |
| **Scaling** | |
| *Scaling* | Manual |
| *From* | -10 |
| *To* | 10 |

25. Close the *Plot Properties Editor* by clicking the **OK** button.
26. From the **Simulation** menu select the **Run** command to start the simulation. The result should look like the figure below.




*The simulation results.*

27. From the **File** menu click **Save**.

If you have problems creating the model, load the model *Bond Graph Model* from the *Getting Started\Bond Graphs* section of the library.

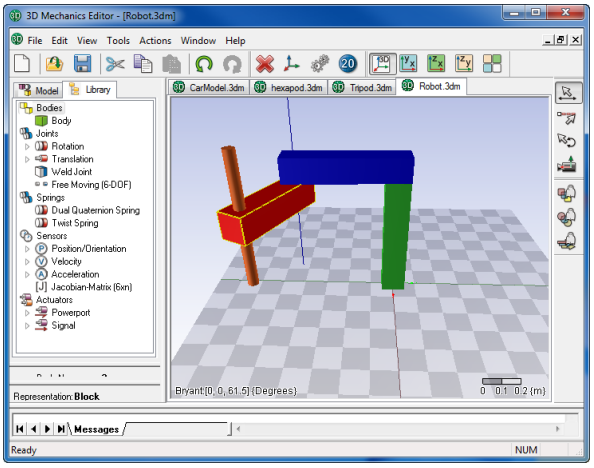
# 9 3D Mechanics Toolbox

## 9.1 3D Mechanics Toolbox

The 20-sim *3D Mechanics Toolbox* provides you with the tool that makes 3D dynamic modeling easier, the *3D Mechanics Editor*.

### Bodies

You can create 3D models by dragging bodies in a 3D workspace. The representations of each body can be changed to a sphere, block, cylinder etc. Furthermore, colors can be changed and descriptions can be added. The size and shape of a body are merely for representation, a body is fully characterized by its inertia coefficients and mass.




*The 3D Mechanics Editor helps you to create 3D models easily.*

### Joints

Bodies are interconnected by the use of joints. Several joints are present in the library, divided in two groups, rotational joints and translational joints. These joints can also be drag and dropped on the workspace. Constraints can be added to create closed loop systems like four bar mechanisms or Stewart platforms.

### Interface

The user interface has 4 different modes in which you can select, connect, translate and rotate bodies and joints. Much effort is done to keep the graphical user interface as natural as possible. Multiple views are supported. Besides the 3D environment, you can see 2D intersections in the xy xz, and yz plane.

### Models

The 3D Mechanics Editor can generate a 20-sim model from your 3D model. This 20-sim model comprises all dynamics and kinematics of the model. Forces can be applied to the joints or on to each body directly. You can also couple springs and dampers from the mechanics library in 20-sim, to the joints, because the whole model is port-based. Gravity can be set as an external force. Eventually, the dynamic response of the complete model can be shown by the *3D Animation Editor*.

## 9.2 Double Pendulum

### Introduction

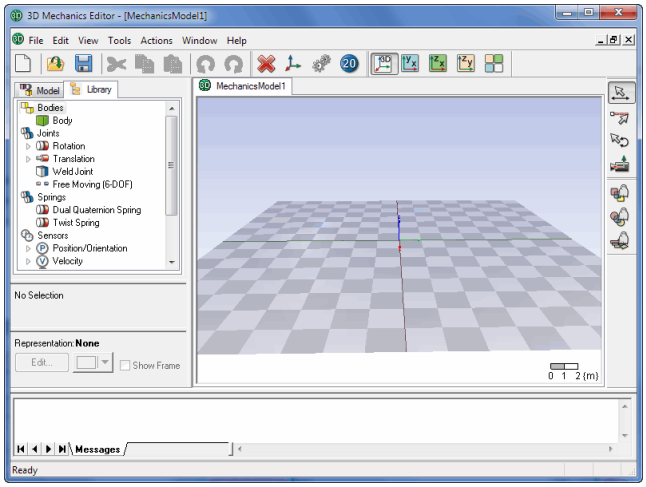
In this section we will use the *3D Mechanics Editor* to create the model of a double pendulum. In this editor you can define the geometry of the pendulum and inspect the possible movements. From the *3D Mechanics Editor* we will generate a 20-sim model that incorporates the equations of motion and a 20-sim scenery for the animation of the pendulum.

The pendulum model that we will create has actuated joints. This means we can apply a torque to the joints. In 20-sim, we will insert the pendulum model and connect passive actuation through dampers. After that we will simulate the model and show a 3D Animation, to see how the damping affects the pendulum behavior.

### Inserting Components

1. Open 20-sim and select **File**, **New** and **Graphical Model**.
2. In the Editor from the **Tools** menu select **3D Mechanics Toolbox** and **3D Mechanics Editor**.

A 3D mechanics model will be inserted in the *Editor* and the 3D Mechanics Editor will be opened:




*The 3D mechanics Editor.*

Like the *Editor, the 3D Mechanics Editor* consists of several parts:

- *Model / Library section*: This is the part at the middle left. The *Model* tab opens the *Model Browser* which shows a hierarchical composition (all the elements) of the model that is created in the *3D Mechanics Editor*. The *Library* tab opens the *Library Browser* which shows all the objects that can be used.
- *Object Properties*: This is the part at the right that in the picture above shows "No Selection". As soon as an object is selected, it will show the name of that object and allow you to change the object properties.

- *Object Representation*: This is the part at the right that in the picture above shows "None". As soon as an object is selected, it will show the name of that object and allow you to change its representation.
- *Graphical Editor:* This is the space with the checker board at the right. In this editor you can drag objects and create models.
- *Messages tab*: This is the part at the right bottom, where all messages are shown.

The Editor has several modes of operation which are indicated by the buttons at the right of the editor:

| button | mode | description |
|---|---|---|
|  | *Translation Mode* | *Use this mode to translate selected objects* |
|  | *Rotation Mode* | *Use this mode to rotate selected objects* |
|  | *Connection Mode* | *Use this mode to connect objects* |
|  | *Camera Movement* | *Use this mode to change the camera view* |

You can choose to make objects transparent:

| button | mode | description |
|---|---|---|
|  | *Ghost Mode for Bodies* | *Click this button to make all bodies transparent* |
|  | *Ghost Mode for Joints* | *Click this button to make all joints transparent* |
|  | *Ghost Mode for Sensors / Actuators* | *Click this button to make all sensors and actuators transparent* |

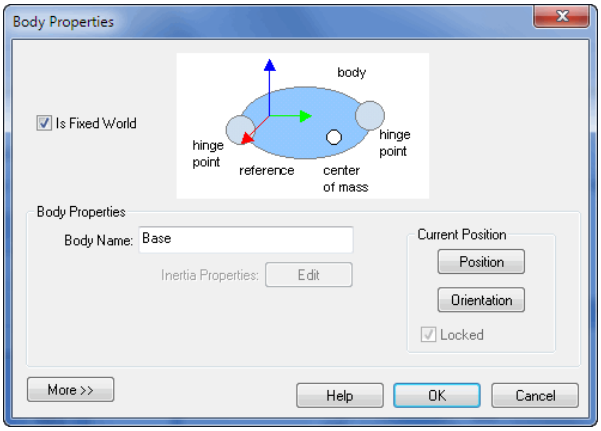
The Editor is in *translation mode*, which is suited for dragging and dropping components. We are going to insert the necessary components to construct a double pendulum.

3. Click on the Library tab.
4. In the **Library** click on **Bodies** - **Body** and **drag** it to the **Graphical Editor** (the checker board).

The first body that is inserted in the *Graphical Editor* will be fixed to the floor and will not move during simulation. Our pendulum will be attached to this body.

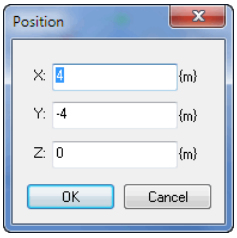
5. **Click** on **Edit Body** button (at the left) to open the Body Properties.
6. **Rename** the body to *Base*.

7. Select the **Is Fixed World** option.




*Use the Body Properties dialog to change the name of a body, its position and mass parameters.*

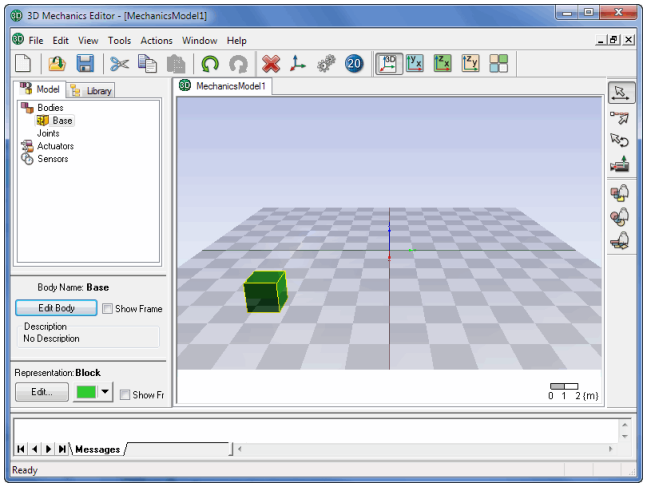
8. Click the **Position** button and change the position to (x = 4, y = -4, z = 0).




*Set the position of the body.*

9. **Close** the Body Properties.

Your 3D Mechanics Editor should now look like:

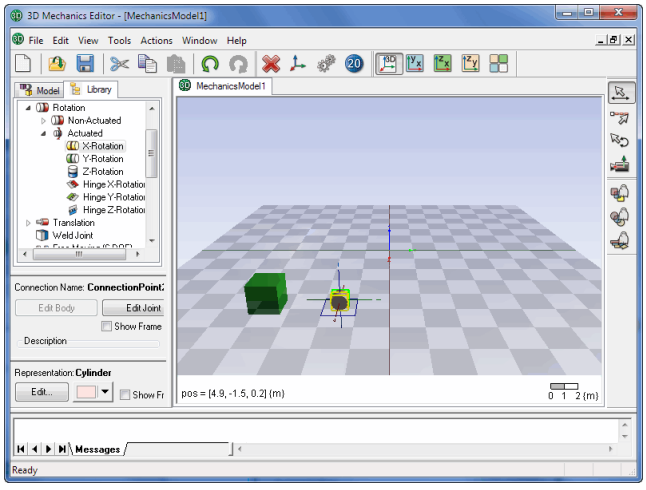



*The model tab shows that one body (Base) is fixed to the ground (indicated by the anchor).*

10. In the *Library tab* **click** on **Joints** - **Rotation** - **Actuated - X-rotation** and **drag** it to the *Graphical Editor*.

If a joint is selected, lines through the center of the joint indicate the possible movements that you can make with the body. If you click the left mouse button you can move the joint over the surface. If you press the *Ctrl-button* while keeping the left mouse button pressed, the body goes up and down.

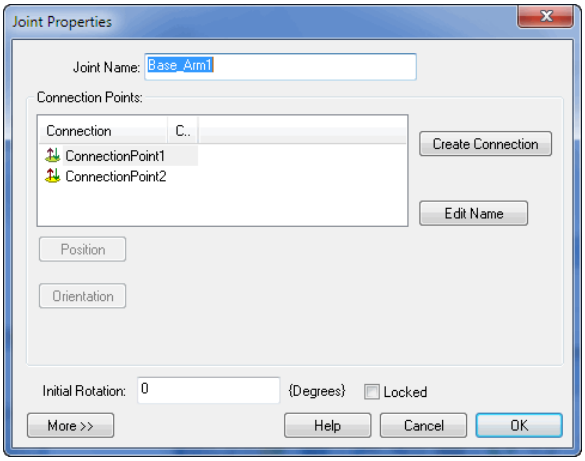
11. **Select** the joint and **place** it next to the body like:




*In Translation Mode you can move objects easily with the mouse pointer.*

12. **Click** on the **Edit joint** button to open the *Joint Properties*.

13. **Rename** the joint to *Base_Arm1*.

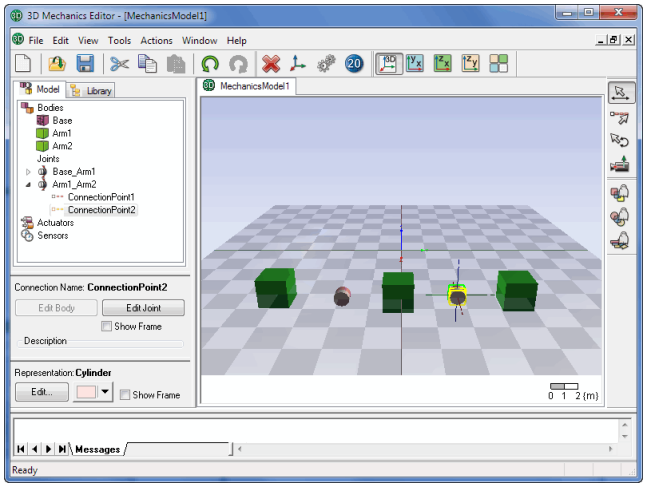



*Double clicking a joint will open the Joint Properties Dialog.*

14. **Close** the Joint Properties.

15. **Insert two** more **bodies** and another **actuated joint** (X-rotation) and rename the bodies to *Arm1* and *Arm2* and the joint to *Arm1_Arm2*.

Now the 3D Mechanics Editor should look like:




*The model tab shows three bodies and two actuated joints.*

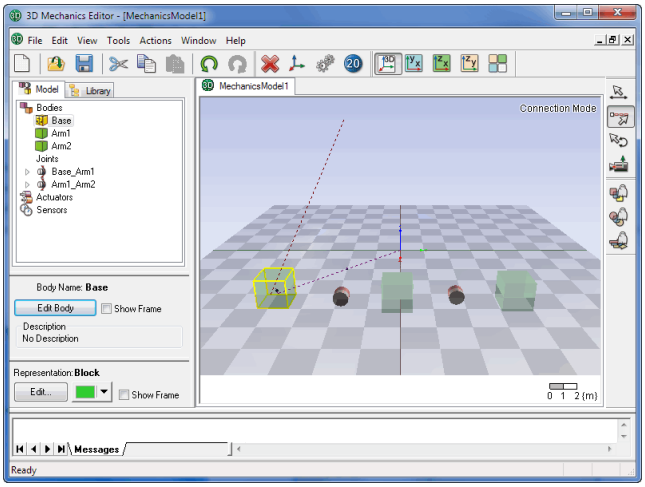
## Creating Connections

16. Click on to set the *3D Mechanics Editor* in **Connection Mode**.

Now we can define the connections by clicking on the components.

17. Put the mouse pointer on top of the ***Base*** body and **click** the **left mouse button**.

Now a dotted line should appear between the body and the mouse pointer to indicate that a connection is being defined between the body and a second component.




*In Connection Mode you can create connections clicking the left mouse button.*

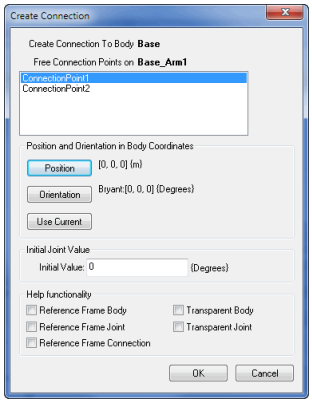
18. Put the mouse on top of the ***Base_Arm1*** joint and **click** again.

Now the connection is made and the **Create Connection dialog** is opened. In this dialog you can define the position offset of the *Base_arm1* joint with respect to the *Base* body. As shown in the Create Connection dialog, every joint has two connections. We will choose *ConnectionPoint1* to start with.

19. Click on **ConnectionPoint1**.

20. Click on the **position button** and set the position to x = 0, y = 0, z = 0.

Now the *Create Connection dialog* should look like:



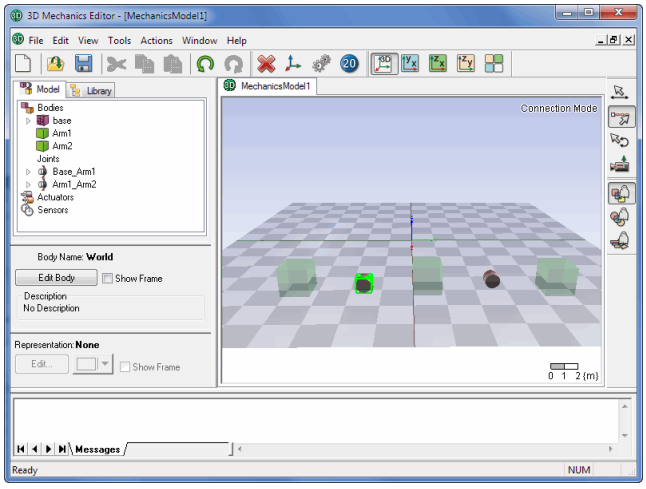

*After a connection is made the Create Connection dialog is opened.*

21. **Close** the Create Connection dialog.

Because we have set the offset position to zero, the *Base_Arm1* joint has the same position as the *Base* body. To better see the joint we can turn on the **Ghost Mode for Bodies**, to make the bodies transparent.

22. Click on  to set the 3D Mechanics Editor in **Ghost Mode for Bodies**.

The 3D Mechanics Editor should now look like:

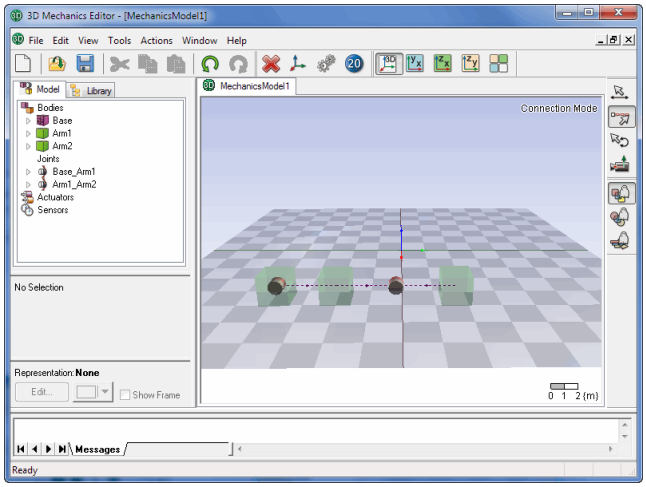



*Use the ghost modes to make objects transparent.*

23. Make the **connections** 2, 3 and 4 according to the table below:

| Action | First click | Second click | Position offset |
|---|---|---|---|
| 1 | Base | Base_Arm1 | x = 0, y = 0, z = 0 |
| 2 | Base_Arm1 | Arm1 | x = 0, y = -2, z = 0 |
| 3 | Arm1 | Arm1_Arm2 | x = 0, y = 2, z = 0 |
| 4 | Arm1_Arm2 | Arm2 | x = 0, y = -2, z = 0 |

The 3D Mechanics Editor should look like:




*All bodies are connected with joints.*

24. Click on  to **turn off** the **Ghost Mode for Bodies**.

## Checking Motion

To see if the model was defined properly we can check it and see how it can move.

25. Click on  to **set** the 3D Mechanics Editor in **Translation Mode**.

26. From the **Actions** menu, select **Check Model**.

The Message Dialog should show *Analysis Completed Successfully*. We will inspect the possible motions of the mechanism.

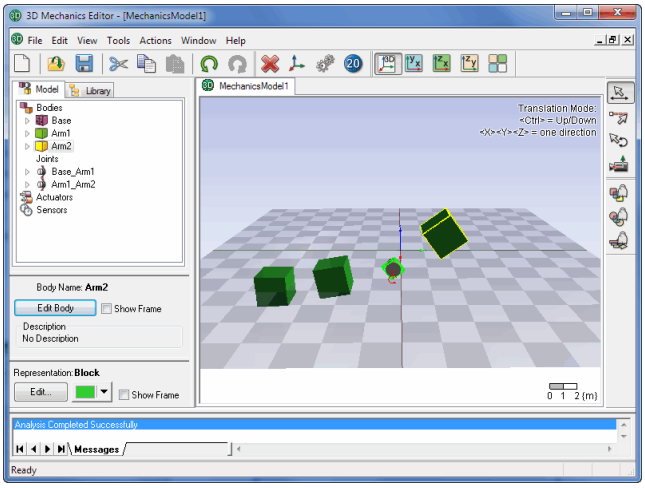
27. Put the **mouse pointer** on top of the *Arm1* body and click the **left mouse button**.

The *Base_Arm1* joint will show an arrow that indicates positive rotation.

28. Keep the left mouse button **pressed** an move it up and down.

The body will now rotate around the joint.

29. Repeat the movement with the *Arm2* body to see its movement.




*If the model is correctly assembled, you can inspect the motion by dragging the mouse pointer.*

30. Select **Check Model** from the **Actions** menu to reset the movement.

During the checking all joint angles will be reset to their initial value. Since we have not changed the initial values (default = 0) the mechanism will return to its original set-up.

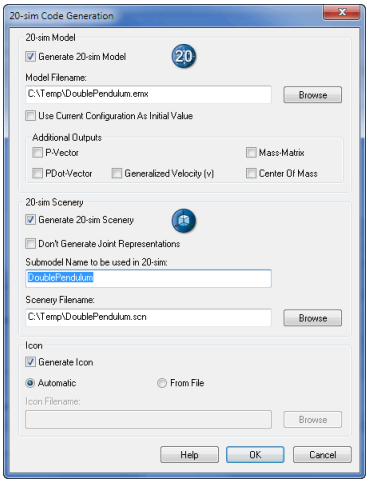
## Updating to 20-sim

31. From the **Actions** menu select **Generate 20-sim model**.

A window will pop up asking you to store the model. 3D Mechanical models are stored in a separate file with the extension *3dm*. If you open a 3D Mechanics model from 20-sim, this file will be read by the *3D Mechanics Editor*.

32. Click **Yes** and store the model in a temporary directory (e.g. *C:\temp*) using the name **DoublePendulum.3dm**.

After the storage of the *.3dm* file the *Code Generation dialog* will be opened.




*In the Code Generation dialog you can enter the settings for the generation of a 20-sim model.*

A number of options can be set in this dialog:

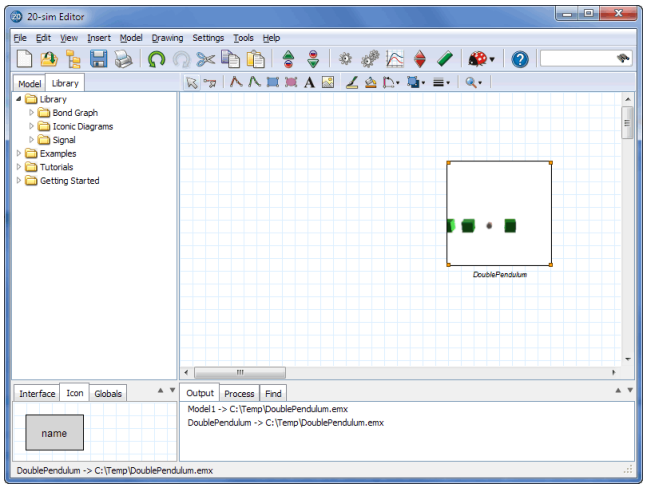
- **Model Filename**: The 3D Mechanics Editor will automatically generate the equations that describe the double pendulum. These equations are stored as a 20-sim submodel.
- **20-sim Scenery**: In 20-sim we will show the pendulum motion in an animation. The animation is generated by the 3D Mechanics Editor and stored on file.
- **Submodel Name**: The double pendulum will be loaded in 20-sim as a submodel, with the name that can be entered here.
- **Icon**: You can choose to make a snapshot of the model (automatic) or load an image from file to be used for the model icon.

In most cases the default entries will be sufficient. We will not make any changes.

33. Click **OK** to export the model to 20-sim.
34. **Close** the **3D Mechanics Editor**.
35. You will be asked to save the model. Choose **Yes** to save it.

## Editor

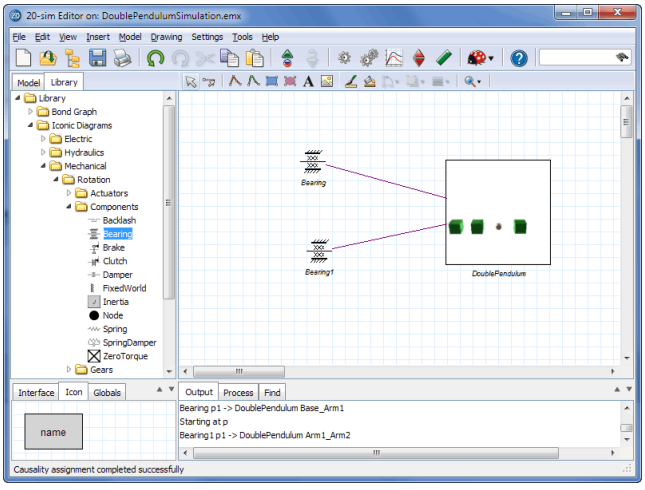
The 20-sim Editor will be updated with the double pendulum model. It should look like:




*The 20-sim model of the double pendulum in the Graphical Editor.*

Now it is a good time to store the model.

36. **Save** the model using the name **DoublePendulumSimulation.emx**.
37. Go to the **Library tab** and select the library **Library\Iconic Diagrams \Mechanical\Rotation\Components**.
37. Insert the model ***Bearing*** twice.
38. Connect the two **bearing** models with the *Base_Arm1* and *Arm1_Arm2* ports of the **DoublePendulum** submodel.



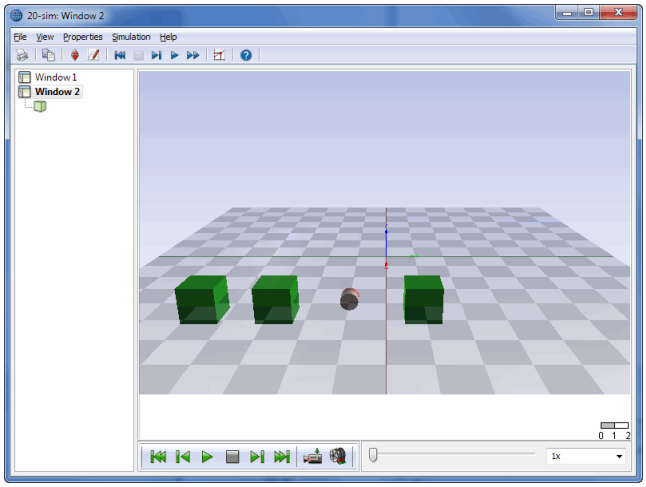
*You can connect two bearing models to the double pendulum.*

40. From the **File** menu click **Save**.

## Simulation

41. From the **Model** menu click **Start Simulator**.

This will open the Simulator with a second window showing a 3D Animation with our pendulum. You can now set the parameters values, choose an integration method, set the plot variables etc. We will use the default parameter values and run a 3D animation.




*The 3D Animation window with the double pendulum scenery loaded.*

42. Click the blue Run button to run a simulation

43. Click the green Run button below the 3D Animation to replay the animation.

If you had problems creating the model, load the model *DoublePendulumSimulation.emx* from the *Getting Started\3D Mechanics Toolbox* section of the library. You should see the pendulum swinging during the simulation. Try to change the bearing parameters to find the changes in the pendulum movement. You can click the Camera button to change camera views.

The pendulum model shows bodies as cubes with default sizes. A more elegant display would result if we could define arbitrary shapes. Moreover, we used default mass and inertia parameters. Both can be changed in the 3D Mechanics editor as will be shown in the next lesson.

## 9.3 Scara Robot

### Introduction

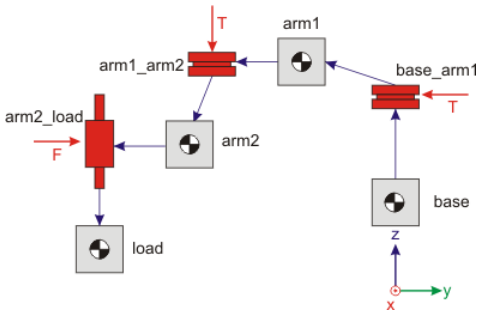
In this lesson you will learn how to create and simulate the dynamic model of a SCARA robot.

*A Selective Compliant Articulated Robot Arm (SCARA).*

- You will expand the knowledge learned from the previous lesson, by introducing mass and inertia parameters and making a more elegant display.
- We will start by inspecting the robot, identify the links and mass and inertia parameters. Then we will introduce the 20-sim model that will be used to simulate the Scara robot. This model contains setpoint generators, controllers and drives that will be connected to the robot.
- We will proceed by entering the Scara robot in the 3D Mechanics Editor. From the 3D Mechanics Editor we will generate a 20-sim submodel that incorporates the equations of motion of the Scara robot. From the 3D Mechanics Editor you can also generate a 20-sim scenery that describes the animation of the Scara robot.
- Finally the Scara robot model will be connected to the drives and simulated.

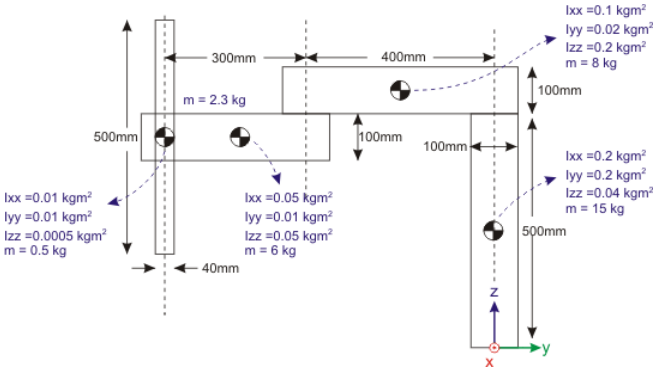
## Inspection

We start the modeling session by inspecting the Scara robot. The robot has a base with arms that can move in the horizontal plane and a load that can move vertically. The robot can thus be modeled using four bodies which are connected by joints. Between the base and the first arm, a rotational joint is mounted, which can be actuated (indicated by the torque T). Between the first and the second arm an identical joint is mounted. Between the load and the second arm a translation joint is mounted, which is also actuated (indicated by the force *F*).




*Arms and joints of the Scara robot.*

The robot model that we are going to create in the *3D Mechanics Editor* will therefore have to three power ports. Two power ports for the rotation joints and one for the translation joint. To simulate the robot model, we have to connect actuators to these joints.

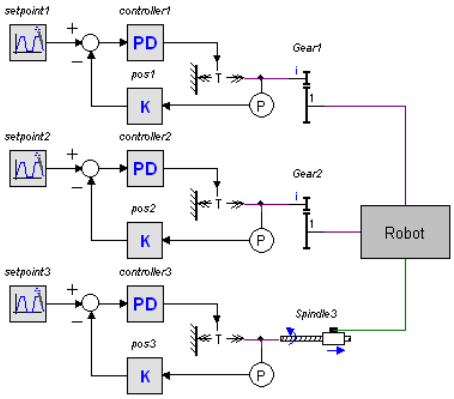



*Geometrical parameters of the Scara robot.*

The geometrical parameters and the masses and inertia's can be found in the design drawings of the Scara robot. We will use the figure above.

## Components

The complete model that will be used to simulate the Scara robot is shown in the figure below. The grey square with the local name robot is generated with the 3D Mechanics Editor. It represents the mechanical structure of the robot. The rotation joints are by driven electrical motors (modeled by ideal torque sources) with gearboxes. The translation joint is driven by an electrical motor (also modeled by an ideal torque source) with a spindle. The motors are controlled by PD controllers, which compare the desired motion (given by motion profile generators) with the actual position of the robot.

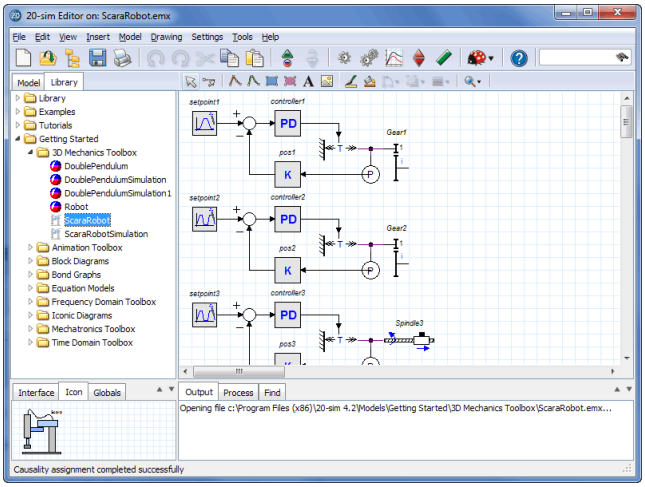



*The 20-sim model of the drive system and the 3D robot model (grey).*

1. **Open** 20-sim and **load** the model *ScaraRobot* from the *Getting Started Manual\3D Mechanics Toolbox* section of the library.

2. Use *File, Save As* to **Store** the model in a temporary directory (e.g. *C:\temp*) using the name **ScaraRobot.emx**.

As you can see, this model contains the drives and controllers for the robot. Only the mechanical part of the robot model is missing. We are going to create this part in the 3D Mechanics Editor.



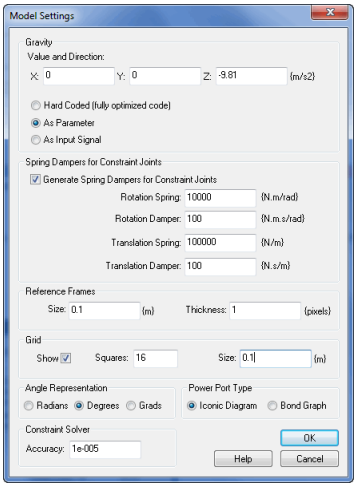

*The 20-sim drive system model is already created: load ScaraRobot.emx.*

3. In the 20-sim Editor from the **Tools** menu select **3D Mechanics Toolbox** and **3D Mechanics Editor**.

This will insert a 3D Mechanics model in the 20-sim *Editor* and open the *3D mechanics Editor*. The dimensions of the Scara robot are rather small. Therefore we are going to change the scaling of the 3D Mechanics Editor.

4. From the **Tools** menu in the *3D Mechanics Editor* window select **Model Settings**.

5. Change the **Size** of the **Reference Frames** to 0.1 and the **Grid Size** to 0.1 as show below.

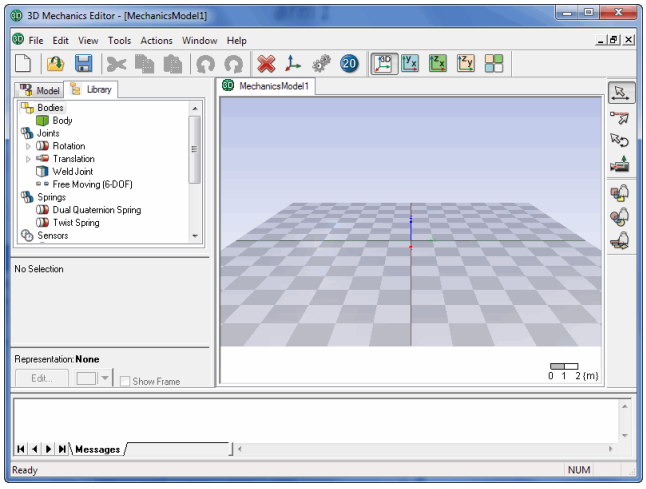



*Changing the frame size.*

6. Click **OK** to **Close** the **Model Settings** dialog.

If the reference frame is too far away you can move the camera closer to the origin.

7. Change the Editor into **Camera Mode** (indicated by  ), press the Ctrl-key and drag the mouse until a good view has been obtained.




*Use the Camera Mode to change the view.*

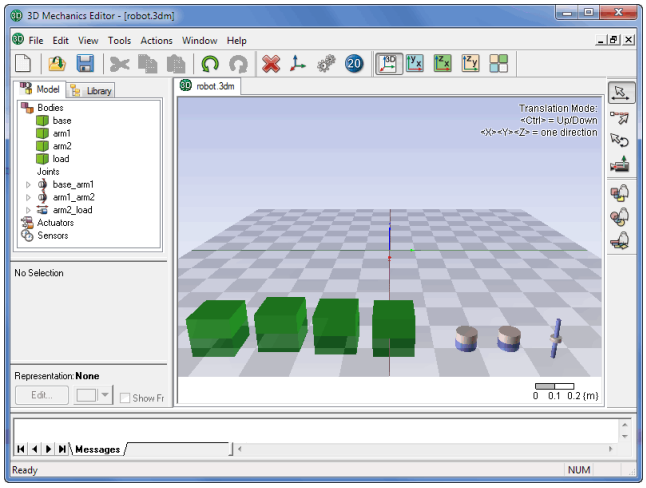
8. Return to **Translation Mode** (indicated by  ).

We are going to insert the necessary components to construct the Scara robot.

9. **Insert** the following components into the *Graphical Editor* and rename them according the table below.

| Action | Component | Name |
|---|---|---|
| 1 | *Bodies\Body* | *base* |
| 2 | *Bodies\Body* | *arm1* |
| 3 | *Bodies\Body* | *arm2* |
| 4 | *Bodies\Body* | *load* |
| 5 | *Joints\Rotation\Actuated\Z-rotation* | *base_arm1* |
| 6 | *Joints\Rotation\Actuated\Z-rotation* | *arm1_arm2* |
| 7 | *Joints\Translation\Actuated\Z-translation* | *arm2_load* |

Now the 3D Mechanics Editor will look like:




*For the Scara robot we need four bodies and three joints.*

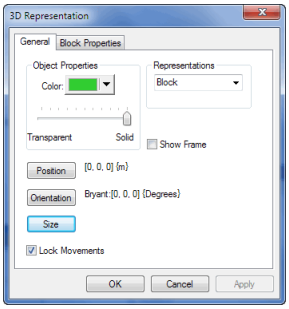
## Representation

To make the mechanism look like a real Scara robot, we have to change the representation of the bodies. We will use the geometrical parameters of the robot.

10. **Select** the ***base*** body.

The **Edit** button of the *Object Representation* (at the left of the 3D Mechanics Editor) should now be active.

11. Click on the **Edit** button to open the **3D Representation** dialog.




*In the 3D Representation dialog you can change the looks of an object.*

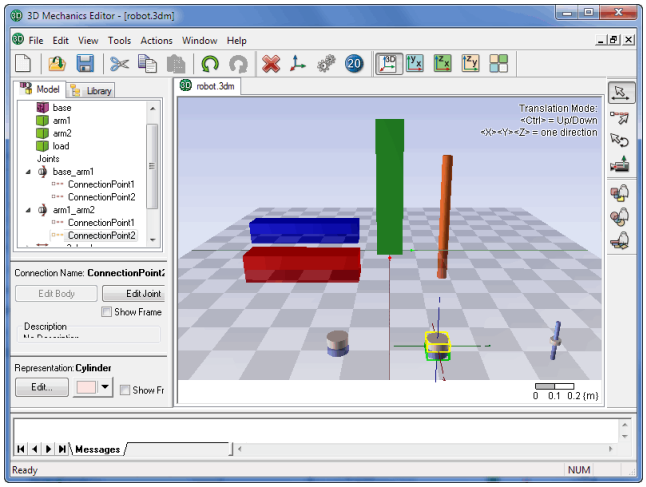
Using the 3D representation dialog, you can change the look of an object.

12. **Change** the representation of the *base* body according to the table below (action 1).

| Action | Body | Size | Color | Representation |
|---|---|---|---|---|
| 1 | *base* | x = 0.1, y = 0.1, z = 0.5 | Green | Block |
| 2 | *arm1* | x = 0.1, y = 0.5, z = 0.1 | Blue | Block |
| 3 | *arm2* | x = 0.1, y = 0.4, z = 0.1 | Red | Block |
| 4 | *load* | x = 0.04, y = 0.04, z = 0.5 | Orange | Cylinder |

13. Do so with the other bodies according to the table (actions 2, 3 and 4).
14. **Double click** on the base body which opens the **Body Properties** dialog.
15. Make sure this body is **Fixed to the World**.
16. Set the **Position** to x = 0, y = 0 and z = 0.25 and close the **Body Properties** dialog.

If desired, you can drag and drop the other bodies to a more favorable location like:




*You can give objects arbitrary shapes and colors.*

## Parameters

The bodies still have default mass and inertia parameters. We will change them according to the specifications given previously.

17. **Double click** on the *arm1* body which opens the **Body Properties** dialog.

18. Click the **Edit** button of the *Inertia Properties* and change the parameters of the *arm1* body according to the table below (action 1).

| Action | Body | Mass [kg] | Ixx [kgm$^2$] | Iyy [kgm$^2$] | Izz [kgm$^2$] |
|---|---|---|---|---|---|
| 1 | arm1 | 8 | 0.1 | 0.02 | 0.1 |
| 2 | arm2 | 6 | 0.05 | 0.01 | 0.05 |
| 3 | load | 0.5 | 0.01 | 0.01 | 0.0005 |

The *base* body is fixed to the ground, which make the inertia parameters irrelevant. If you want to insert then anyway, temporary remove the *Is fixed world* option.

19. **Do the same** with the other bodies according to the table (actions 2 and 3).

## Connections

To assemble the Scara robot we will create connections between the various bodies and joints.

20. **Click** on  to set the 3D Mechanics Editor in **Connection Mode**.

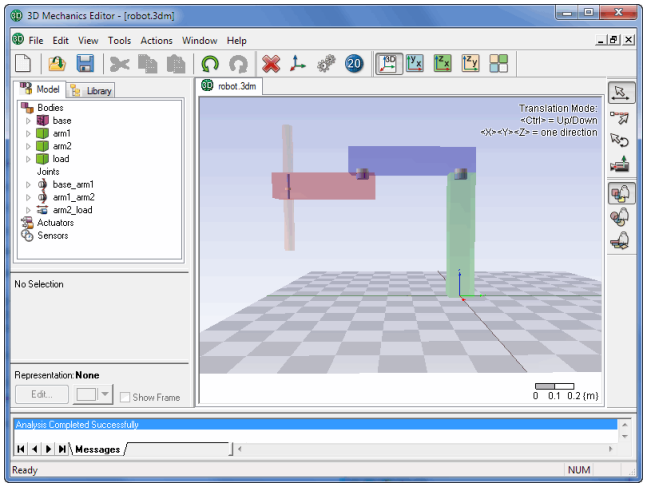
Now we can define the connections by clicking from joints to bodies. To make the joints better visible, we will use the Ghost Mode for bodies.

21. **Click** on  to set the 3D Mechanics Editor in **Ghost Mode for Bodies**.

22. Make the **connections** according to the table below:

| Action | First click | Second click | Connection Point | Position |
|---|---|---|---|---|
| 1 | *base_arm1* | *base* | ConnectionPoint1 | x = 0, y = 0, z = 0.25 |
| 2 | *base_arm1* | *arm1* | ConnectionPoint2 | x = 0, y = 0.2, z = -0.05 |
| 3 | *arm1_arm2* | *arm1* | ConnectionPoint1 | x = 0, y = -0.2, z = -0.05 |
| 4 | *arm1_arm2* | *arm2* | ConnectionPoint2 | x = 0, y = 0.15, z = 0.05 |
| 5 | *arm2_load* | *arm2* | ConnectionPoint1 | x = 0, y = -0.15, z = 0 |
| 6 | *arm2_load* | *load* | ConnectionPoint2 | x = 0, y = 0, z = 0 |

When all bodies and joints are connected, 20-sim will start to assemble the robot. Your *3D Mechanics Editor* should now look like:




*20-sim will assemble all objects as soon as they are all connected.*

## Checking Motion

To see if the model was defined properly we can check it and see how it can move.

23. **Click** on  to set the 3D Mechanics Editor in **Translation Mode**.

24. **Click** on  to **de-select** the **Ghost Mode**.

25. From the **Actions** menu, select **Check Model**.

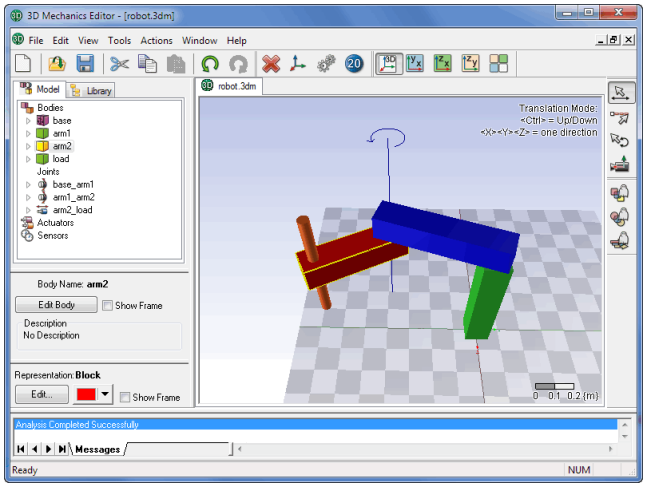
The Message Dialog should show *Analysis Completed Successfully*. We will inspect the possible motions of the mechanism.

26. Put the mouse pointer on top of the *arm1* body and **click** the left mouse button.

The *base_arm1* joint will show an arrow that indicates positive rotation.

27. Keep the left mouse button **pressed** an move it up and down.

The body will now rotate around the joint.




*If the model is correctly assembled, you can inspect the motion by dragging the mouse pointer.*

28. Repeat the movement with the *arm2* and the *load* bodies to see their movements.

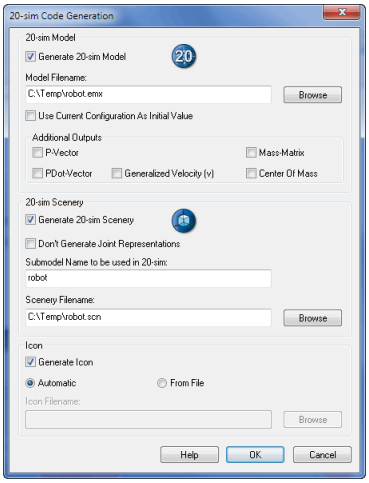
## Updating to 20-sim

29. From the **Actions** menu select **Generate 20-sim model**.

A window will pop up asking you to store the model.

30. **Store** the model in the same temporary directory as the 20-sim model (e.g. *C:\temp*) using the name **Robot.3dm**.

After the storage of the *.3dm* file the *Code Generation dialog* will be opened.

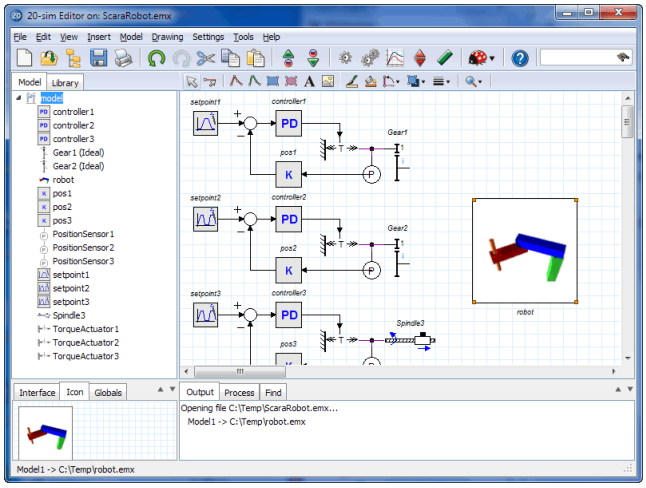



*In the Code Generation dialog you can enter the settings for the generation of a 20-sim model.*

31. Click **OK** to export the model to 20-sim.
32. **Close** the **3D Mechanics Editor**.
33. You will be asked to save the model. Choose **Yes** to save it.

## Editor

The 20-sim Editor will be updated with the double pendulum model. It should look like:



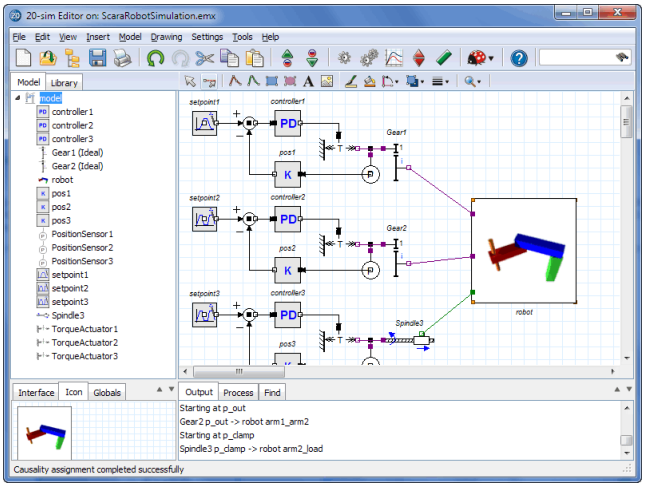

*The 20-sim model of the drive system and the robot model.*

Now it is a good time to store the model.

34. **Save** the model.

If you had problems creating the model, load the model *ScaraRobotSimulation.emx* from the *Getting Started\3D Mechanics Toolbox* section of the library. If you look at the *Robot* model, you can see that it has three ports. One for the *arm base_arm1* joint, one for the *arm1_arm2* joint and one for the *arm2_load* joint. We are going to connect these joints to the actuators.

35. **Connect the actuators** to the robot model (*Gear1* to *base_arm1*, *Gear2* to *arm1_arm2* and *Spindle3* tot *arm2_load*), until the model looks like:

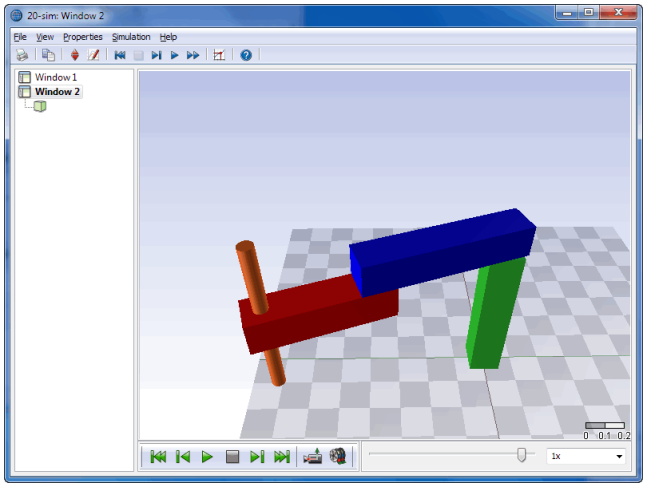



*The robot model is connected to the drive trains.*

## Simulation

36. From the **Model** menu click **Start Simulator**.

This will open the *Simulator*. As you can see a predefined experiment is loaded, showing a plot with setpoint and position variables. The 3D animation is filled automatically with the generated scenery file.




*3D Animation of the Scara robot.*

37. Click the **Run** button to start a simulation and show the corresponding animation.

38. You should see the robot move during the simulation. You can click the **Camera** button to change camera views.

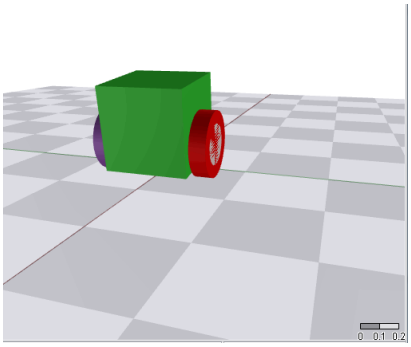
You can also show the robot animation in the main Simulator window. Try to insert a 3D Animation plot. You must open the 3D mechanics model and generate a 20-sim model (see number 29) to make it effective.

## 9.4 Contact Modeling

### Introduction

In the previous section, we have seen how to create a dynamic model of a robot with *3D Mechanics Editor* using Scara Robot. This section describes how to create a model with some basic contact in it.

In order to explain contact modeling, we use a model of a two wheel robot, that should balance itself. It is a small "Segway" robot as shown below. The contact of the wheels with the ground are modeled. At the end, the robot should be able to drive in straight line while balancing itself.



*Model of two wheel "segway" robot*

## Model of the Robot

### Assumptions

The following assumptions are made during modeling.

- The ground is a horizontal plane where where z==0.
- The robot wheels will always stay level w.r.t. the ground.

### Geometry

The robot will have the following geometry:

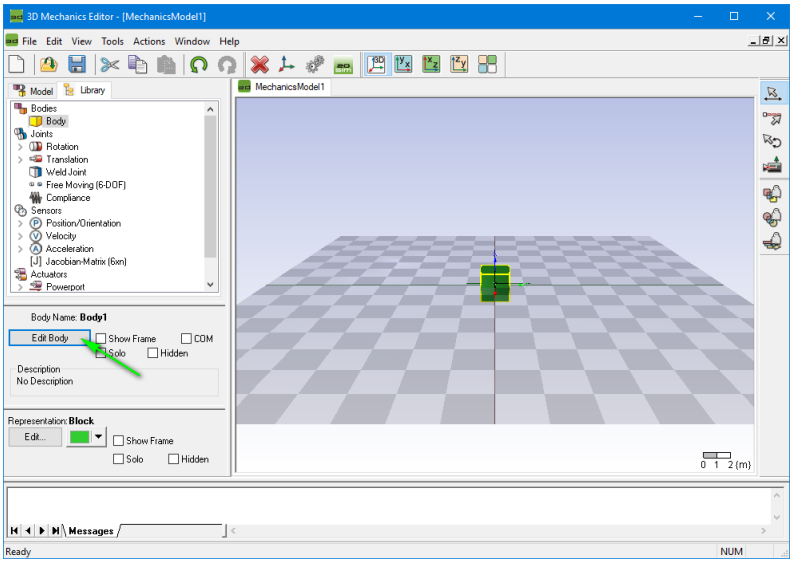
- The main body will be modeled by a cube with mass=1{kg}
- The size of the main body (cube) is 10 {cm} in all directions.
- The wheels will have a radius of 4{cm}.
- The width of the wheels will be 2{cm}
- The mass of each wheel is 0.5 {kg}
- The geometry of the wheel is a cylinder.
- Connection of wheels is made at 2.5 {cm} distance from the bottom of the cube

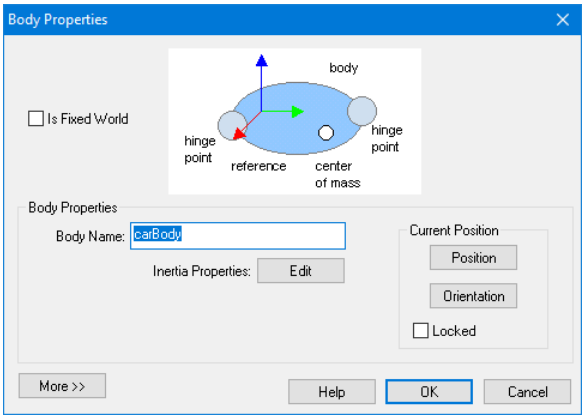
### Step 1: Modeling main body

1. First start 20-sim.
2. In the 20-sim Editor from the **Tools** menu select **3D Mechanics Toolbox** and **3D Mechanics Editor**.

This will insert a 3D Mechanics model in the 20-sim *Editor* and open the *3D mechanics Editor.*

3. From the ***Library*** tree in *3D Mechanics Editor*, drag a **Body** to the workspace.
4. Select the body and click **Edit Body**, as shown below, and change the name to **'carBody'**.

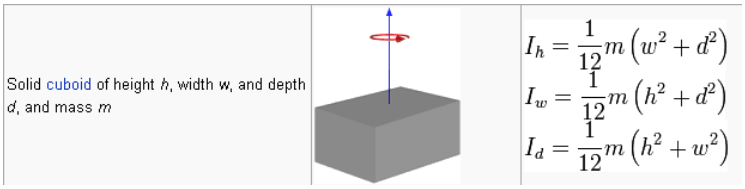



*Edit body dialog*

5. Then press the **Edit** button and edit the *Inertia* and *Mass* properties for this object.

Our body has a mass ($m$) of 1{kg} and a width ($w$), height ($h$) and depth ($d$) of 10{cm}. The inertia can therefore be calculated as:

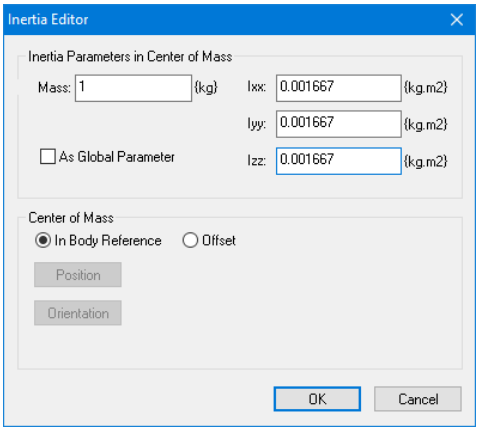


*Inertia calculations*

Substituting the *m*, *w*, *h* and *d* values, we get:

$$I_h = I_w = I_d = 1/12 * m * (h^2 + w^2) = 1/12 * 1 * (0.1^2 + 0.1^2) = 0.001667$$

In our case, $I_h$ , $I_w$ and $I_d$ correspond to $I_{xx}$, $I_{yy}$ and $I_{zz}$ respectively.



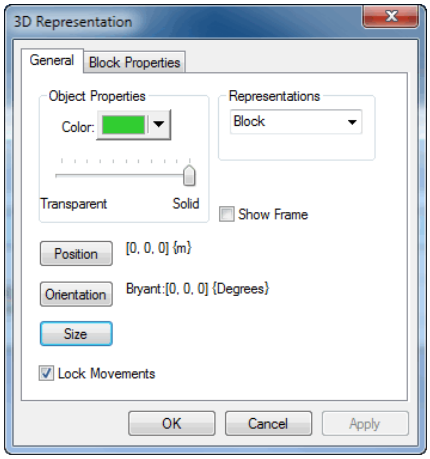

*Mass and inertia properties*

6. Press **OK** on this dialog, and then **OK** on the Body Properties Dialog.

Now the shape of the cube must be set that it represents indeed a cube of 10{cm}.

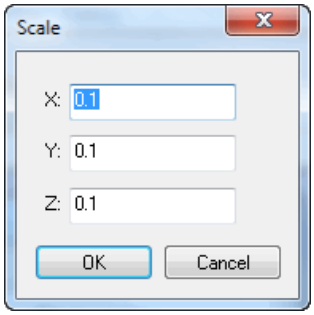
7. To change the representation, press the **Edit** Button in the *Representation* section at the left of the editor.

This opens a dialog shown below with which we can choose the color, shape and size of a body. It is important to know that this is indeed only a representation. Any change made here does **NOT** influence the dynamic behavior of the model, but only the appearance.




*Edit representation dialog*

8. Press the **Size** button, and for every direction choose 0.1{m} and click **OK**.

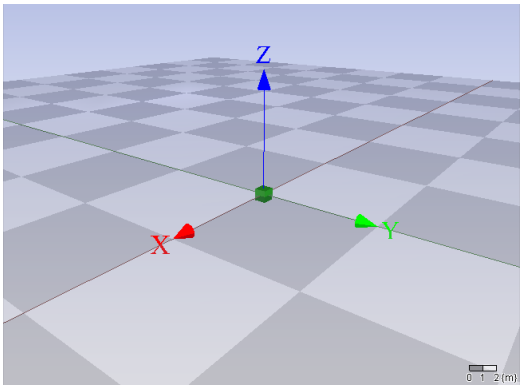



*Changing size*

The color of the body can also be changed with this dialog.

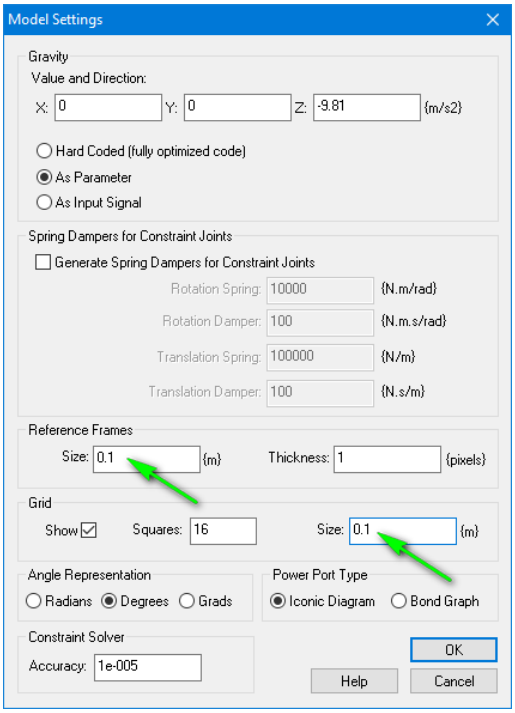
9. Choose a color you like and press OK to close.

If the body is not visible anymore, you can move the camera closer to the object by moving the mouse while pressing the Ctrl-key.
Now our worksheet will look like this:

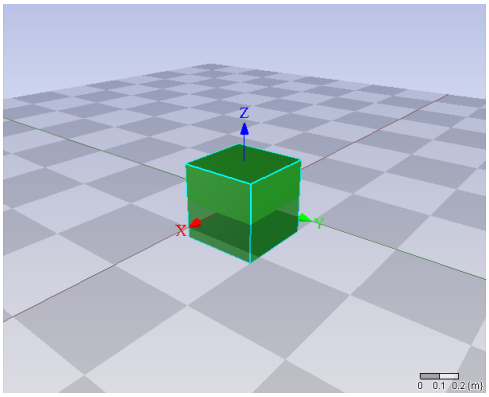



The grid is currently a square of 1 {m} which is a too big for our model.

10. To change the grid, choose from the **Tools** menu **Model Settings.** The dialog shown below opens.



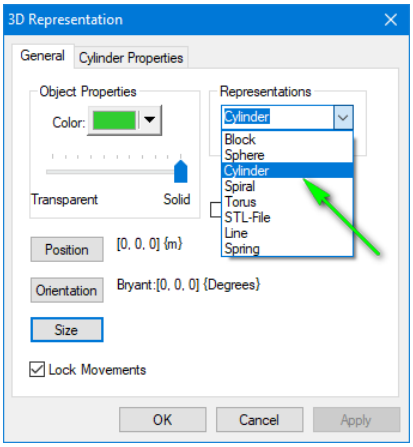

Here you can change the size of the reference frames, and the grid. Change both to 0.1 {m}. The scenery then looks like as shown below.

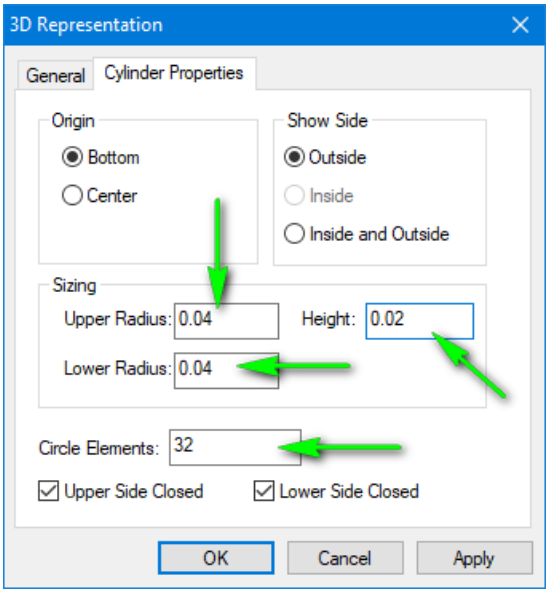



### Step 2: Inserting left wheel

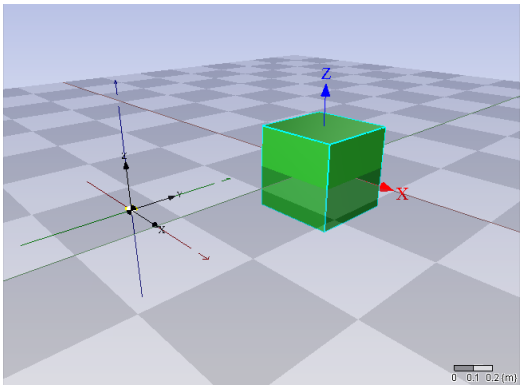
Using similar steps followed, we insert a body for the left wheel. The size of the newly added body is now already scaled by 0.1. This is because of the settings for the reference frame size we changed in the previous step.

11. Change the representation to be a cylinder with a radius of 4 {cm} and height of 2 {cm} as shown below.



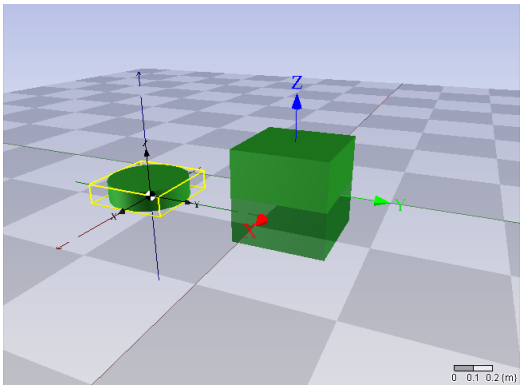

12. Press **Ok** to apply the settings. The scenery will now look like this:




*Cylinder representation with wrong scaling*

Not exactly what we expected, but it can be explained: Because we changed the size of the reference frames to be 0.1, all new bodies inserted will have a scaling of 0.1 in all directions. So by setting the exact dimension of the cylinder in the *Cylinder Properties*, all those are now scaled by 0.1.

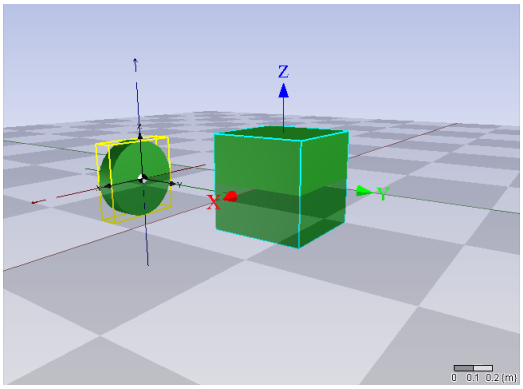
13. To adjust the setting, open the representation of the cylinder and change the *scale* back to 1. It will now look like this:




*Cylinder representation with correct scaling*

We see that the orientation of the cylinder is such that it is lying down.
Ultimately, we need the wheel to be connected to the left side of the body with y-coordinate -0.5 and rotate in y-axis. For that the representation should be adjusted accordingly.

14. Open the *cylinder representation* again and change the *orientation* in x-axis to 90-degrees.




*Orientation changed to 90-degrees in x-axis*

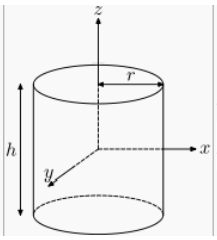
You see that it has rotated to the left (positive angle) and that the origin is at the bottom.

15. Change the color of the cylinder to purple (Red, Green, Blue = 153, 85, 187) to indicate Left Wheel.

We now have the correct representation, we will go ahead and change the body properties.

On Wikipedia we can find the following for calculating the inertia of a cylinder.

Solid cylinder of radius $r$, height $h$ and mass $m$




$$I_z = \frac{mr^2}{2} \quad [1]$$

$$I_x = I_y = \frac{1}{12} m \left(3r^2 + h^2\right)$$

We are rotating over the y-axis, so take that into account. Also in the picture the reference frame is at the center of the cylinder, but we have it at the bottom.

16. Open the Body Properties and change the name to **LeftWheel**.

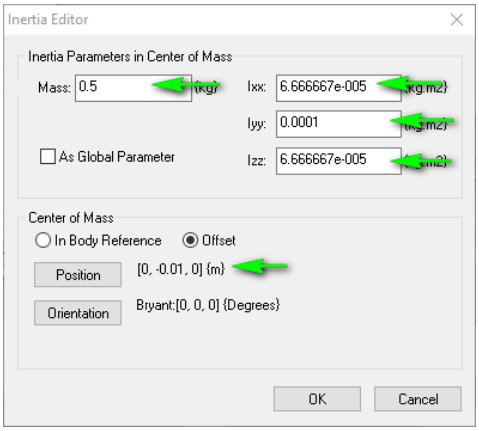
Substituting for mass (m) of 0.5 {kg}, height (h) of 0.02 {m} and radius (r) of 0.04 {m}, the inertia's we can be calculated as the following.

$$I_y = \frac{mr^2}{2} = \frac{(0.5 * 0.02^2)}{2} = 0.0001\ kg.m^2$$

$$I_x = I_z = \frac{1}{12} m(3r^2 + h^2) = \frac{1}{12} * 0.5 * (3 * 0.02^2 + 0.02^2) = 6.666667 * 10^{-5}\ kg.m^2$$

In the Inertia Properties, we can also specify the location of the center Of Mass w.r.t. the body reference. Since the center of mass is really at the center, and we have our body reference at the right side. The position offset is 1{cm} to the left.

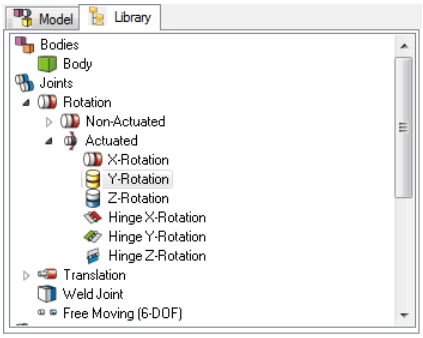
17. Fill in the calculated inertia values and position offset in the dialog as shown below.

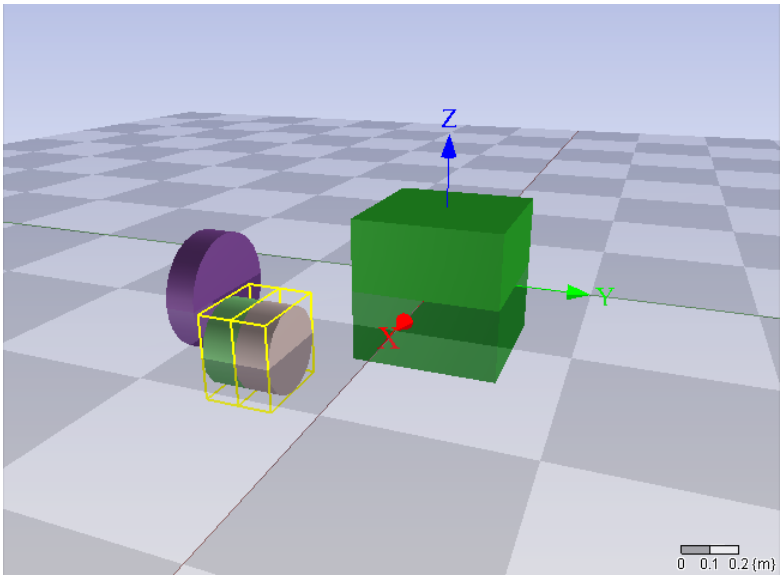



### Step 3: Connecting the left wheel

We're going to attach the left wheel to the car body using a joint that rotates over the y-axis. We want to be able to actuate this joint.

18. Choose the *Actuated Y-Rotation Joint* from the library and drag the joint to the workspace.

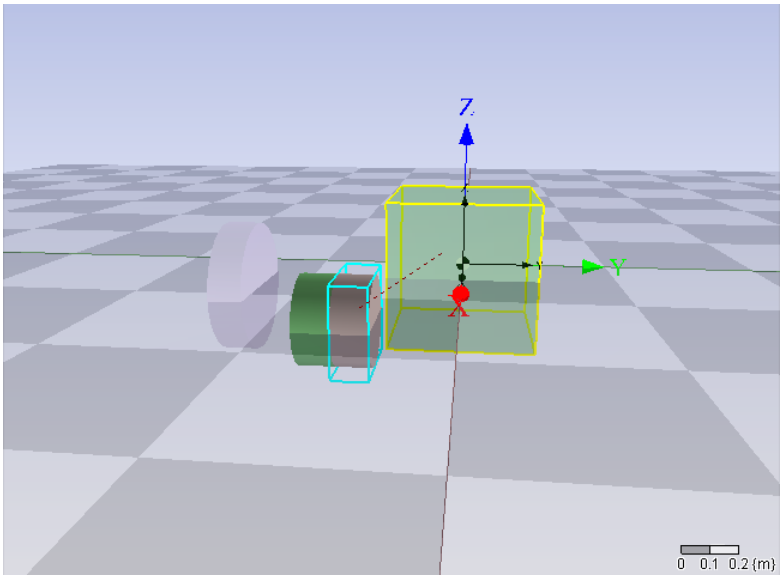




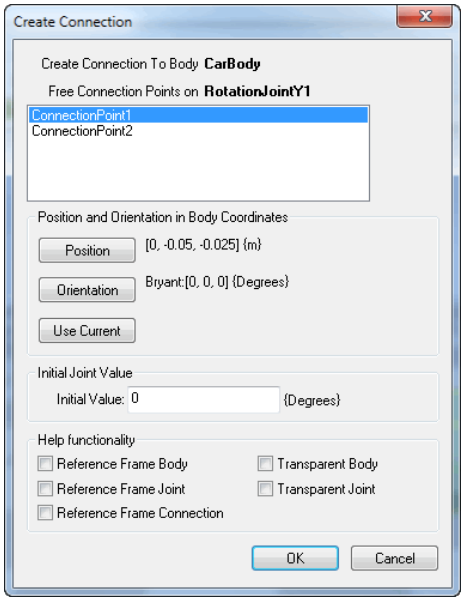

19. Now set the 3D Mechanics Editor in **Connection Mode** by pressing the Space Bar, or by selecting the *Connection Mode* icon in the right button bar.

Now we want to connect the wheel to the body. The joint has two connection points. We are going to connect the first connection point to the *CarBody* and the second connection point to the *LeftWheel*.

20. Start connecting by dragging from the CarBody to the Joint.

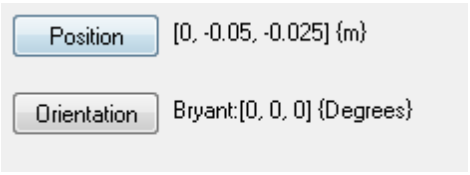



Once you release your mouse left button when you selected the joint the following dialog will appear:




At the top, it shows that a connection is made from *CarBody* to *RotationJointY1*. We're using *ConnectionPoint1* to make the connection.
The position and orientation must be made w.r.t. the body coordinates.

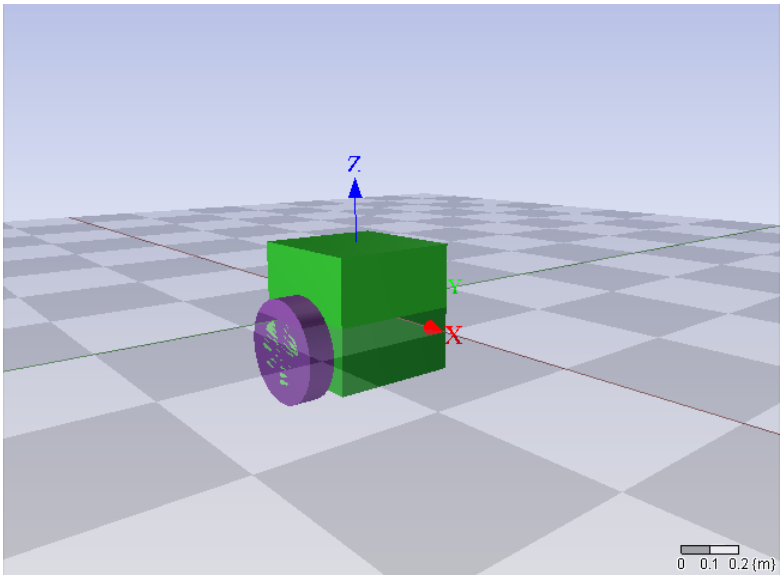
21. The orientation is the same as the body and is therefore set to zero. The position offset is [x,y,z] = [0, -0.05, -0.025]




In the scenery still nothing happens, since the joint is not yet connected to the wheel. So we will do the same action but now for the connection to the wheel.

22. Choose the position as [0, 0, 0] and the orientation also [0, 0, 0].

Now the model will look like this.



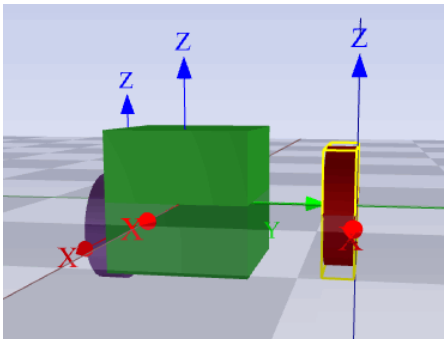

### Step 4: Inserting right wheel

The easiest way to create the second wheel is by copying the first wheel.

23. Select the left wheel and press Ctrl-C and Ctrl-V. Or choose *Copy and Paste* from the *Edit* menu.

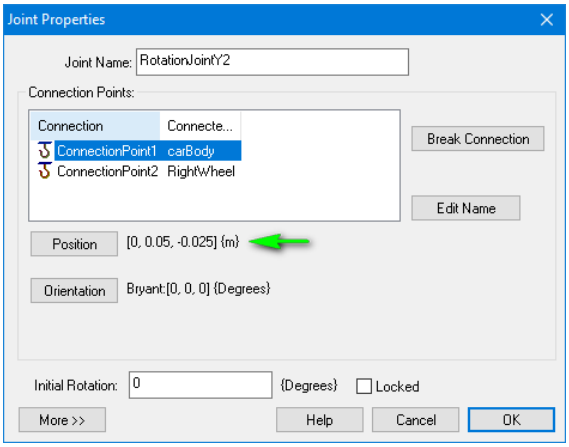
A new wheel is inserted.

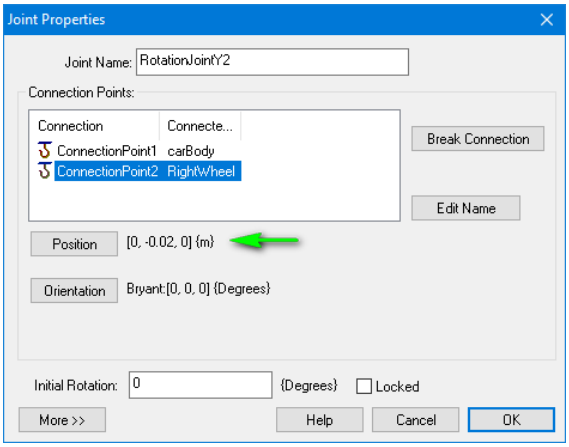
24. Change its name from *LeftWheel2* to **RightWheel** and it's color to **red**.



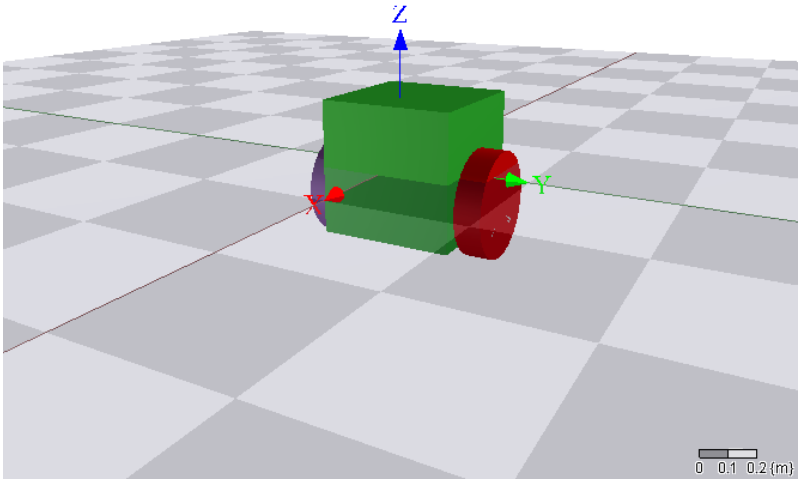

**Step 5: Connecting the right wheel**

24. Connect a second Actuated Y-Rotation Joint this wheel just like with the first wheel. Make sure the settings are set as shown below.



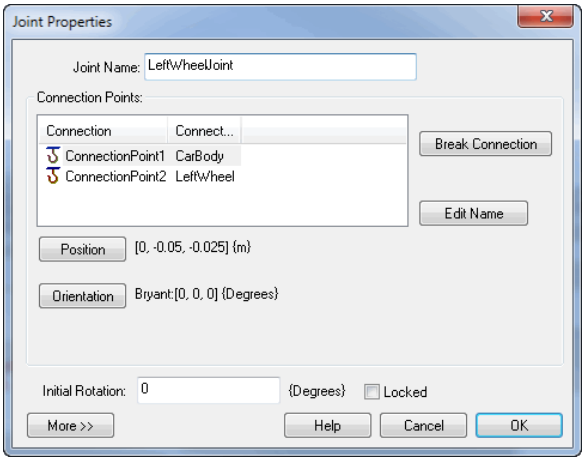



Now it will look like this.




At the left side of the editor, you can select the model tree by pressing the tab with **Model**. Here you can see that the joints have the name *RotationJointY1* and *RotationJointY2*.

25. With the right mouse button select **Joint Properties**. A dialog will open, here you can change the names of the joints to **LeftWheelJoint** and **RightWheelJoint**.

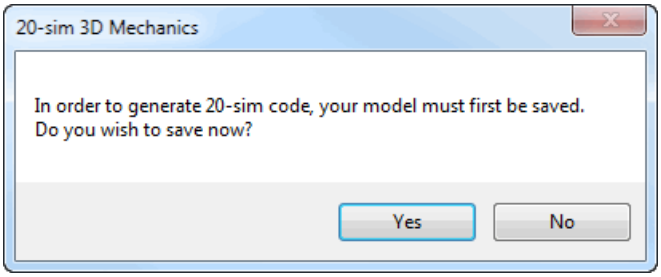



### Step 6: Simulating the model

At this point we have enough information to make a simulation.

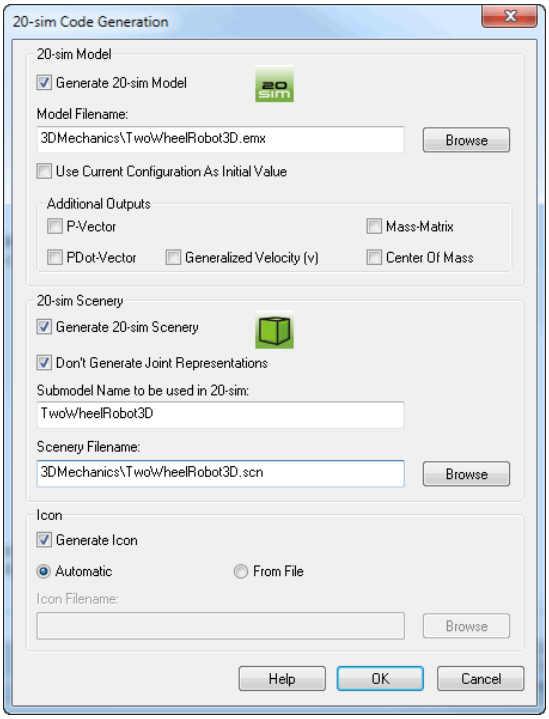
26. Press the 20-sim icon in the button bar (see figure below) in order to generate the 20-sim code.

First you will get the question:

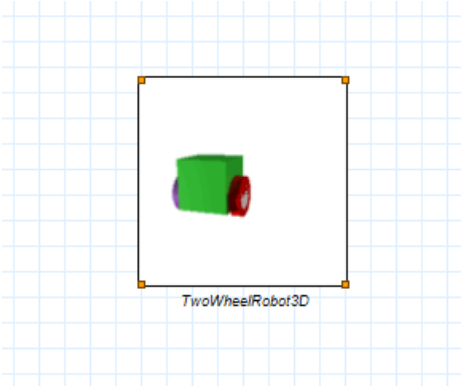



27. Click **Yes** and choose a location and name. e.g. TwoWheelRobot3D.3dm for your .3dm model.

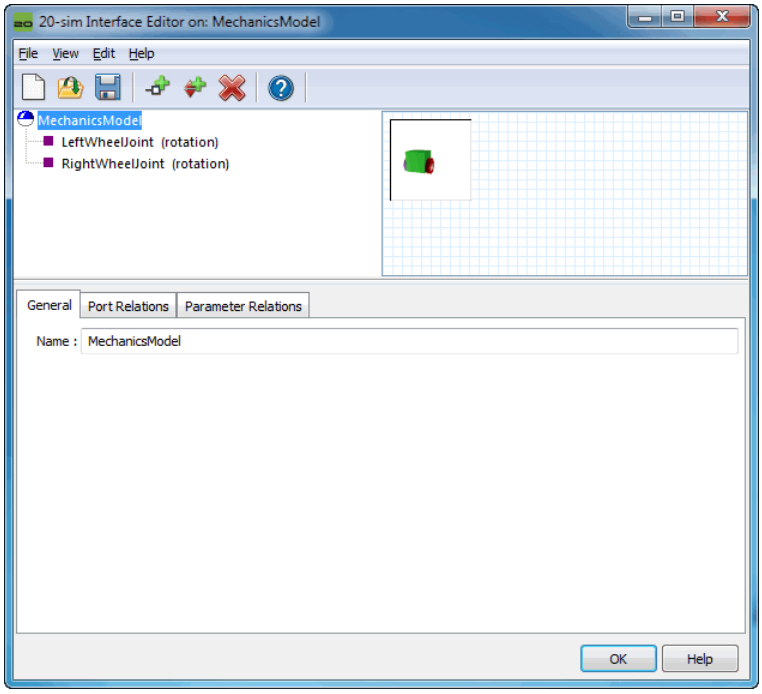
This name MUST be different from our 20-sim model name later on. Once you chose your name, the Code Generation Dialog will open. It will show the following:




28. Press **Ok**, and 20-sim will automatically update it's icon and pop to the front.

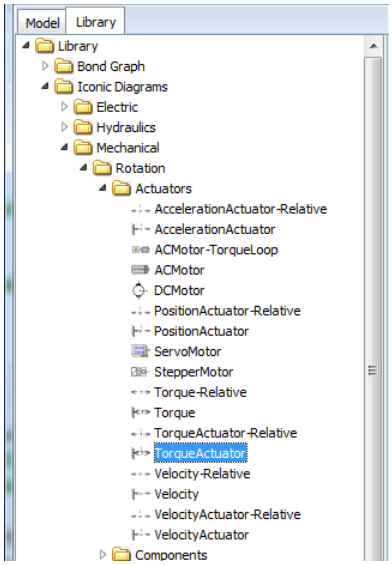



If we look at the interface of this model (right mouse click: select **Edit Interface**) we see that there are two ports:



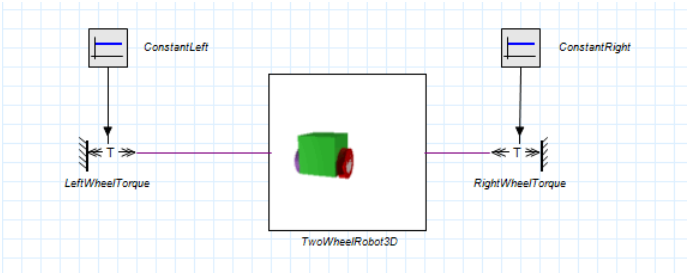

Just close the dialog, since we don't have to make modifications to it.
We are now going to connect simple torques to the Joints.

29. Select **Torque Actuator** from the iconic Diagram library and drag two of them on the worksheet.

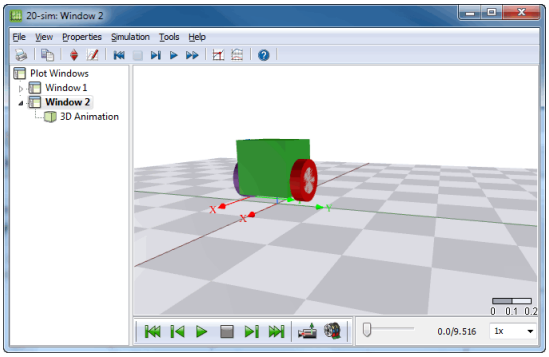



30. Change the names of the Torque Actuators to **LeftWheelTorque** and **RightWheelTorque**.

Now connect the actuators to the TwoWheelRobot3D sub-model. Two constant source sub-models are used as an input to the torque actuators.




31. Now press the **start simulator** button and the *simulator window* will open together with the *3D animation window*.




32. Now press the run button. And then the replay animation.

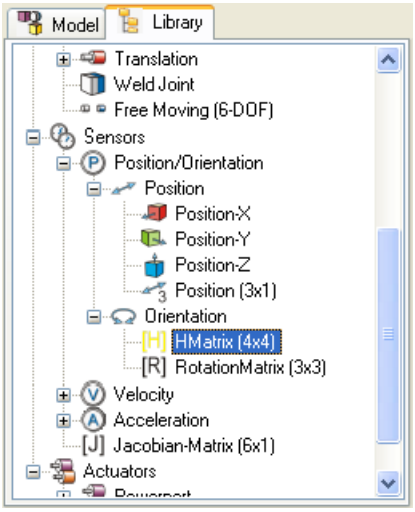
You will see that the car will just drop through the floor. Therefore, it is now time to add ground contact.

### Step 5: Adding ground contact

To add ground contact, go back to the 3D Mechanics Editor.

*Determining the lowest point of the wheel*

33.From the library choose H-Matrix from the Sensor\Position&Orientation\Orientation.




This will return a 4x4 matrix giving the position and orientation of a specific point on the car.
We're going to connect it to the center of each wheel. The idea is, if we know the center of the wheel, we can subtract the radius of the wheel in z-direction to obtain the lowest point of the wheel.

34. Drag the H-Matrix and change the name into **LeftWheelAbsH**. Then connect it to the left wheel just like you connected the joint. Since the body reference is at the side of the cylinder we want to place it really in the center, by giving it an offset of [0, -0.01, 0].

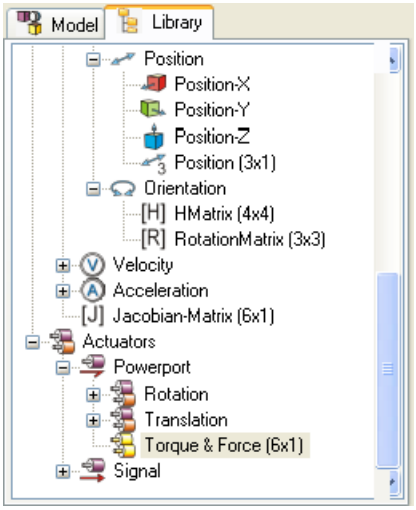
35. Repeat this action for the right wheel and change the name to **RightWheelAbsH.**

When we generate a 20-sim model again we now have two additional ports that give the H-Matrices for the left wheel and right wheel. By subtracting the radius of the wheel in z-direction we get the lowest point. This will be done in 20-sim.

*The ground contact*

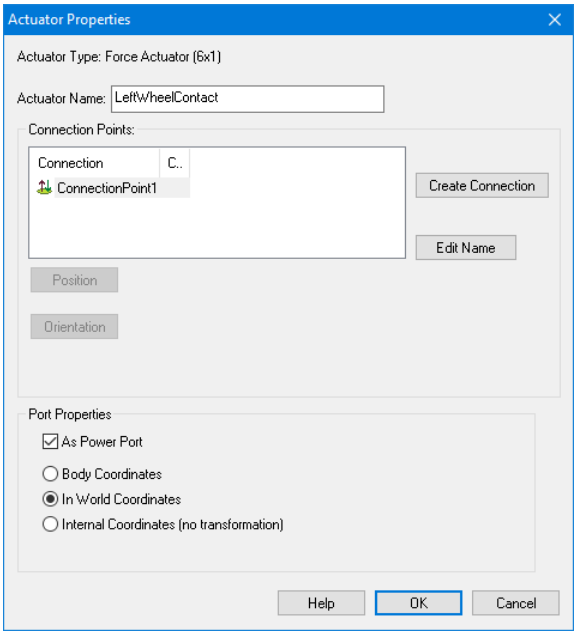
The ground contact is modeled by applying a force upwards which is dependent on position. The force must be perpendicular to the floor.
We do this by applying a *force Actuator* also at the same location as the H-Matrix sensor.

36. From 3D mechanics Select the **Torque & Force** actuator from the library.




**37.** Drag it to the workspace and rename it to **LeftWheelContact.**

Note: Make sure that the *Port Properties* are set to **World Coordinates**.



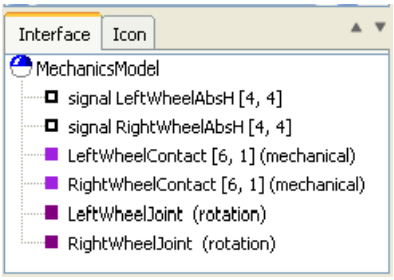

**38.** Connect the actuator to the *LeftWheel* at **position [0, -0.01, 0]**

**39.** Do the same for the *RightWheel* but connect it to **position [0, 0.01, 0]**

Now an external force can be applied to the center of the wheel, but always with the orientation of the inertial frame.

39. Generate 20-sim model again.

In 20-sim the model will be updated and 4 additional ports will appear, 2 for each wheel.



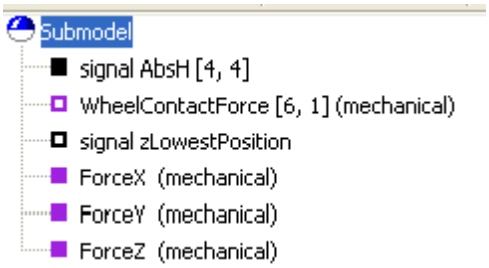

40. In 20-sim Insert a new empty submodel. (with the right-mouse button) and name it **LeftWheelContact**.

Open the interface for this model and add the following ports:

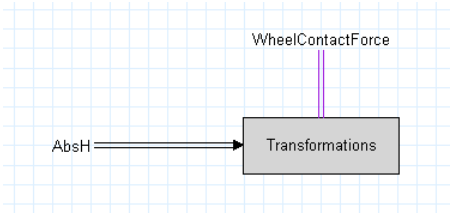
- signal port: AbsH[4,4]
- Iconic Diagram Port, mechanical: WheelContactForce[6,1]

41. Connect both ports it to the TwoWheelRobot3D model.
42. Now Double Click on the LeftWheelContact sub-model and choose a graph sub-model. A graph sub-model will show with the two ports *WheelContactForce* and *AbsH*
43. At this level, insert yet another new sub-model with name **Transformations**.
44. Open the Interface editor for this sub-model and add the following ports:

- signal port: AbsH[4,4]
- Iconic Diagram Port, mechanical, Orientation Output, Causality: Fixed Force Out: WheelContactForce[6,1]
- signal port, Orientation Output: zLowestPosition
- Iconic Diagram Port, mechanical, Causality: Fixed Force In (Fixed velocity out): ForceX
- Iconic Diagram Port, mechanical, Causality: Fixed Force In (Fixed velocity out): ForceY
- Iconic Diagram Port, mechanical, Causality: Fixed Force In (Fixed velocity out): ForceZ




45. Connect the AbsH port and the WheelContactForce.




46. Double Click on the *Transformations submodel*. Choose *equation sub-model* and enter the following.

```
parameters
       real global g_radius; //radius of the wheel
variables
```

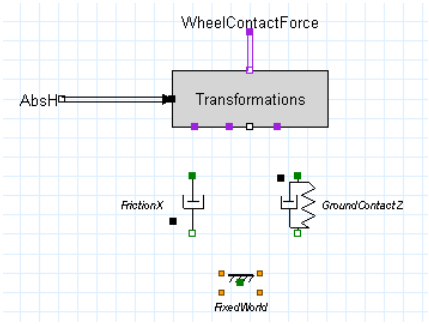
```
        real LowestPointAbsH[4,4];

        real AbsHinverse[4,4];
        real invAdjointH[6,6];
        real WheelWrench[1,6];
        real WheelTwist[6];

equations

        // copy all values from the AbsH matrix
        // but by subtracting the g_radius from the z-coordinate

        // the rotation part
        // now unity, so no rotation w.r.t. inertial frame
        LowestPointAbsH[1:3,1:3] = eye(3);

        // the position part
        LowestPointAbsH[1,4] = AbsH[1,4];                       // x-position
        LowestPointAbsH[2,4] = AbsH[2,4];                       // y-position
        LowestPointAbsH[3,4] = AbsH[3,4] - g_radius;            // z-position

        // and the last row, which is [0, 0, 0, 1];
        LowestPointAbsH[4,1:4] = AbsH[4,1:4];

        // and set the output port
        zLowestPosition = LowestPointAbsH[3,4];


        // make an adjoint matrix, so we can transform a force
        // acting on the lowest point, to the center of the body
        AbsHinverse = inverseH (LowestPointAbsH);
        invAdjointH = Adjoint (AbsHinverse);

        // make a 1x6 vector holding the translational forces
        WheelWrench = [0.0, 0.0, 0.0, ForceX.F, ForceY.F, ForceZ.F];

        // transform the incoming WheelContactForce vector
        WheelContactForce.F =  transpose (WheelWrench * invAdjointH);

        // and do the same for the twist (velocity) vector
        WheelTwist = invAdjointH * WheelContactForce.v;

        // and pass the velocity
        ForceX.v = WheelTwist[4];
        ForceY.v = WheelTwist[5];
        ForceZ.v = WheelTwist[6];
```

*Describing the ground contact*

At the three separate ports **ForceX**, **ForceY** and **ForceZ**, we're going to connect the ground contact and the friction in x and y-direction.

47. From the Translation library, choose the ***FixedWorld***, **Damper**, and **SpringDamper** sub-models.

48. Rename the *Damper* and *SpringDamper* to **FrictionX**, **GroundContactZ**.




49. Now open the interface of the *FrictionX* model, and add a port name *zPositionLowestPoint*.

50. Do the same with the *GroundContactZ* model.

51. Now go down in the FrictionX model and change the line at the equations like this

```
equations
       p.F = if zPositionLowestPoint < 0 then
                              d * p.v
                      else
                              0.0
                      end;
```

This means that the friction only is active if the lowest part of the wheel is lower that the ground.

52. For the *GroundContactZ* model rewrite the equations to be like this

```
equations
       p.F = if zPositionLowestPoint < 0 then
                              k * zPositionLowestPoint + d*p.v
                      else
                              0.0
                      end;
```

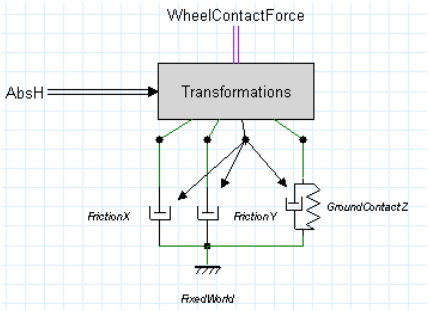
This means there is a spring/damper combination active only when the wheel goes through the ground.
We will determine them later when we make a simulation.

*Connecting the sub-models*

It is now time to connect the sub-models.

53. First copy *FrictionX* and paste it to the model *FrictionY*. They will be both the same.

54. Now connect *FrictionX*, *FrictionY* and *GroundContactZ* to the *Transformations* sub-model at the top. Some small nodes will be inserted, because of the port orientation. Also connect the bottom of the models to the *FixedWorld* and the *zLowestPosition* to all three models.

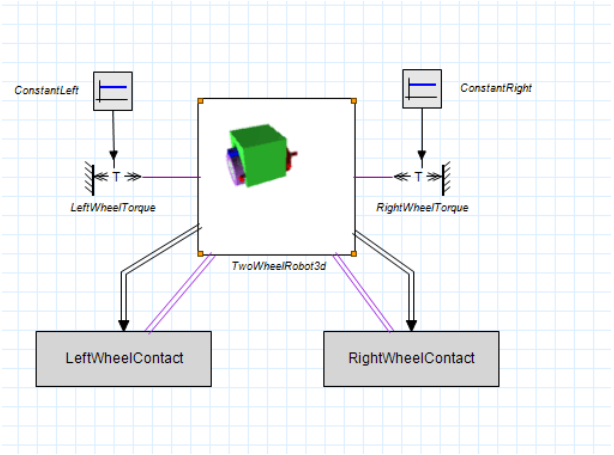



55. Now go up one level, so you see the *TwoWheelRobot3D* model again.

56. Copy the *LeftWheelContact* model, and paste this model, and rename it to **RightWheelContact**.

57. Connect it to the *TwoWheelRobot3D* model, in the same way you connected the *LeftWheelContact* model.

Note: The global parameter *g_radius* is defined twice since we copied the implementation of the *LeftWheelContact* to the *RightWheelContact*. Therefore, remove the value of the parameter at one of the sub-models.




## Simulating the model

Before simulating the model, the parameters should be set properly.

58. Open the parameters editor and select the sub-model **LeftWheelContact \GroundContactZ** and change the parameters to:

    d = 1k {N.s/m}
    k = 100k {N/m}

**59.** Do the same with the **RightWheelContact\GroundContactZ.**

**60.** Open the *Parameters Editor* again, and change the **ConstantLeft** and **ConstantRight** models the values to **C = 0.01**

**61.** Change the friction in X -and Y-direction of both left wheel as right wheel to **d = 10 {k.N.s/m}**

The contact model should now be working as expected. However, the car is rotating. This is because the wheels are actuated with a constant torque. The torque is applied between the wheels and the body, so they will rotate in opposite directions.

### Step 6: Balancing the mass

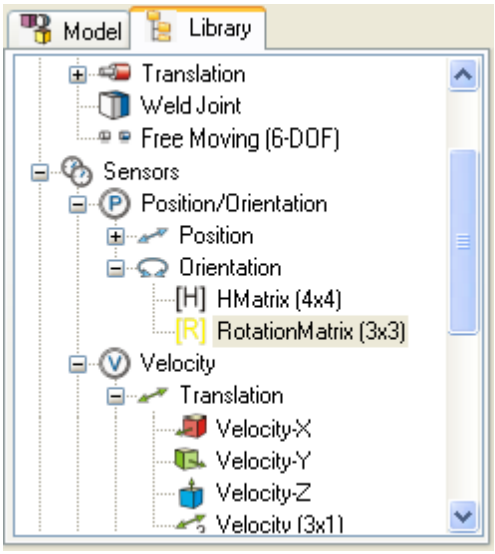
We want to be able to control two things:

The mass of the robot should remain upright.
The mass of the robot should be under a certain angle such that it moves with a constant velocity.

Because we rotate over the y-axis a positive rotation will tilt the mass in the forward x-direction. In order to do this, we must move the robot forward to balance this mass. So by controlling the angle, we control the velocity of the vehicle.

*Measuring the angle of the robot*

**62.** Go to the 3D Mechanics model and from Sensors in the library choose **RotationMatrix(3x3)**




63. Rename it to **Rbody** and connect it to the *CarBody* at the location of the axis: [0, 0, -0.025].

Generate 20-sim code again. An additional port is now available holding the Rotation matrix of the CarBody.

From this rotation matrix we can extract the tilting-angle.

64. In 20-sim create a new sub-model with the name *CarAngle*. At the interface define two ports:

- input: signal R[3,3]
- output signal angle

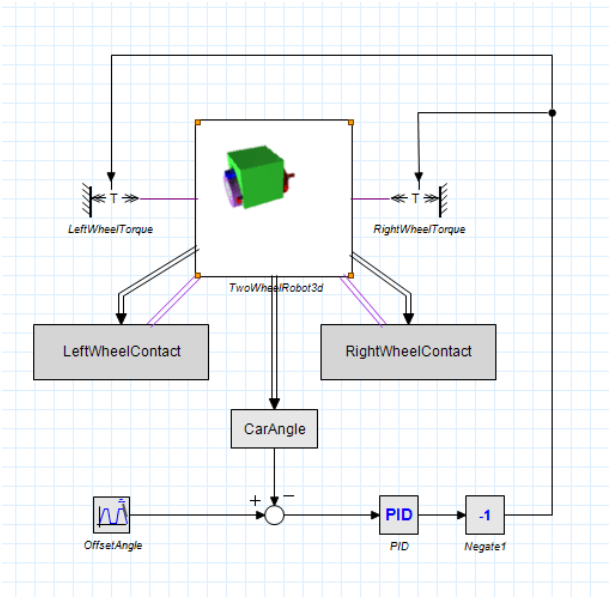
For the implementation fill in the following:

```
variables
       real angles[3];
equations
       angles = dll('EulerAngles.dll', 'EulZYXrFromRotationMatrix', R);
       angle = angles[2];
```

What this does is, it gives the three angle rotations that are applied next to each other in the order: Z-axis rotation, Y-axis rotation and X-axis rotation.

If we're moving in the x-direction, the rotation of the x-axis and z-axis are zero, so this will only have value for the y-axis.
If we drive in a circle, we first have to rotate over the z-axis, and then the y-axis. The x-axis will remain zero in that case as well.

Now connect the complete model in the following way:




What we see here is that the angle of the car is compared to a reference angle. The difference is fed into a PID controller. The output of the controller is given to both wheel motors.
There is an additional multiplication of -1, because a positive torque on the wheels, will make the wheel move backward (positive angle is counter-clockwise). We want it to move forward with a positive measured angle.

The OffsetAngle can be created by a simple MotionProfile submodel. Choose the amplitude not larger than 0.5 {rad}. You can give any type of movement you want. The PID works fine with the default parameters. Simulating the model should drive the car in a straight line.

# 10 Animation Toolbox

## 10.1 Animation Toolbox

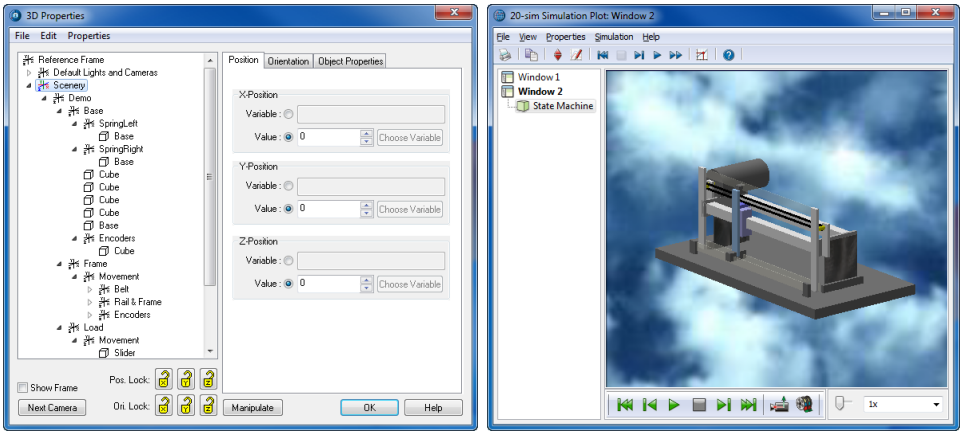
Animation is a powerful method to inspect and verify your simulation results, especially when working in three dimensions. The 20-sim *Animation Toolbox* offers you an easy way to create 3D Animations and view graph animations.

### 3D Animation

Simulation results in 20-sim can be shown as animations using a *3D Animation Editor*. Animations are composed of predefined objects like cubes, spheres, lines, squares, cameras and lights. Complex objects can be imported from CAD packages using common exchange formats.

### Objects

Any variable of a 20-sim model can be connected to an object. Various attributes of the object can be controlled this way: position, orientation, size, etc. Thermal graphs can be created by controlling the color of the objects.




*In the 3D Animation Editor you can connect every object property to a variable in the model.*

Reference frame objects can be used to group animation objects and inherit object attributes. Objects can be duplicated, resulting in complex animation with a few mouse clicks.

### Connected to Simulation

Animations are fully linked to the 20-sim *Simulator*. While a plot is being drawn, simultaneously the animation will be shown! This link is kept after a simulation has finished. While inspecting the numerical values, you will notice that the 3D Animation changes simultaneously!

### Movies

Every animation can be exported to movies in various formats (mpg, wmv, avi. flash) for the use in presentations and external programs.

### Graph Animation

With 20-sim Graph Animation you can display the results of a simulation in your graphical model. During simulation, the thickness and color of the bonds, connections and signals will correspond with the values they carry.

## 10.2 3D Animation Basics

This lesson demonstrates how to build a simple 3D-animation. It helps you to understand the basics of reference frames, define objects and couple them to variables in your model.

### Model

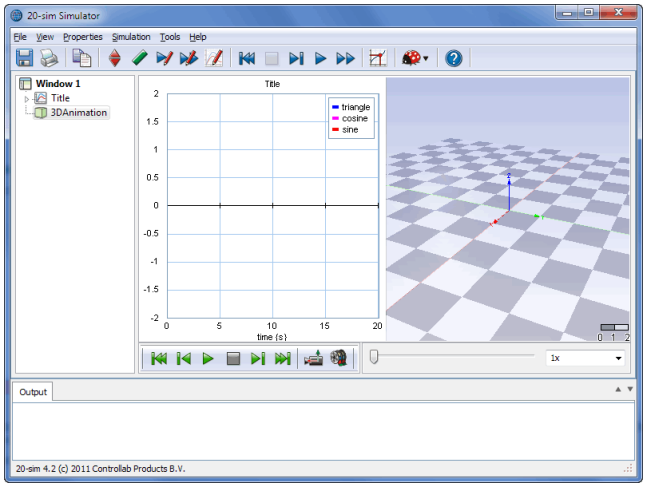
1. **Open** 20-sim and **load** the model *3DAnimation* from the *Getting Started\Animation Toolbox* section of the library.

This model consists of three signal generators. We will use the outputs of these generators to show moving objects in a 3D Animation window.

2. Open the *Simulator*.

3. In the tree at the left select **Window1**. From the **right mouse menu** select **Add Plot** - **3D Animation**.

This will open a *3D Animation plot* next to the standard plot:

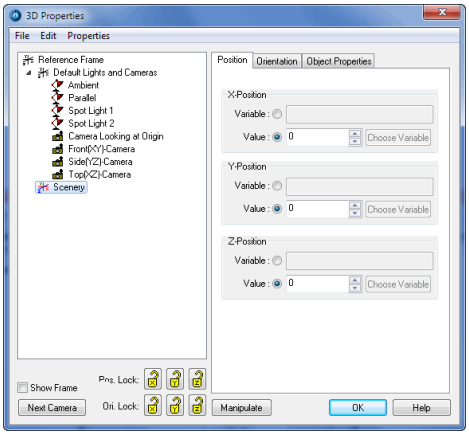



*The 3D Animation window at start-up.*

The first time the 3D Animation window is opened, the **top reference frame** is shown (red = x, green = y, blue = z) in a semi transparent grid (the checkerboard) that covers the xy-plane. All objects you want to show in the 3D Animation window, must be defined with respect to this frame.

4. In the tree at the left select **3D Animation**. From the **right mouse menu** select **Plot Properties**.

This opens the *3D Animation Properties window*:




*The 3D Animation Properties window at start-up.*

At the left of the *3D Animation Properties window* an object tree is shown with several reference frames and objects:

- Reference Frame: This is the top reference frame. All other reference frames and objects are defined with respect to this frame.
- Default Lights and Cameras: This reference frame contains the default lights and cameras.
- Scenery: In this reference frame you can add your own objects.

5. **Close** the *3D Animation Properties* and re-open it by **double clicking** in the 3D Animation plot.

## Cameras

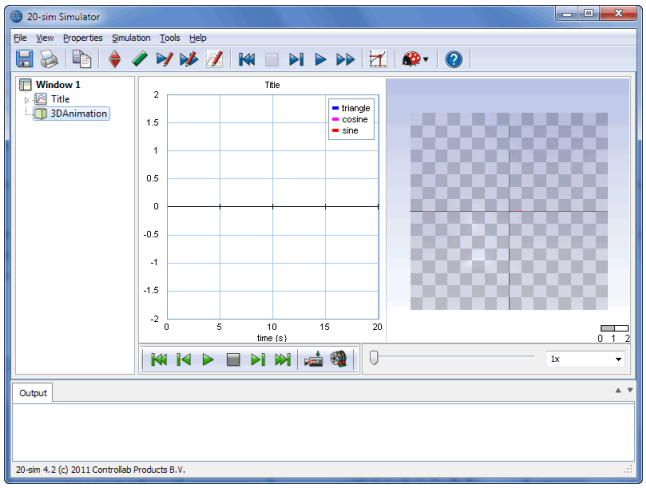
The 3D Animation window comes with five default cameras. All cameras are defined with respect to the *Default Lights and Cameras* frame, which has the same orientation as the top reference frame.

### Front Cameras

There are three cameras which are positioned on top of the principal axes of the *Default Lights and Cameras* frame. The front cameras do not show in perspective.

6. In the **3D Properties** window, from the objects tree select the **Front(XY)-Camera**.

As you will notice the view in the 3D Animation window changes. It shows the x-axis (red) and y-axis (green) of the top reference frame:




*The view with the Front (XY) Camera selected.*

7. Try out the other cameras.

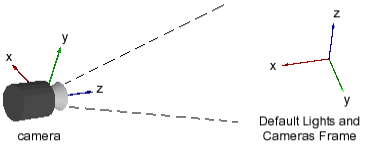
### Camera Looking at Origin

The *Camera Looking at Origin* can be positioned at every desired place but will always look at the origin of the *Default Lights and Cameras* frame.

8. In the *3D Animation Properties window*, from the objects tree select the **Camera Looking At Origin** and inspect the *position tab* on the right of the window.

9. Change the Y-position of the **Camera Looking At Origin** and inspect the results in the 3D Animation window.

As you will see the position of the camera can be changed, but its orientation will always be chosen to make the camera look at the origin of the *Default Lights and Cameras* frame.




*Position and orientation of the Camera Looking at Origin.*

## Adding Objects

With the camera in a good position, it is easy to add objects. We will remove the *3D Animation window* and open it again to get a fresh set of cameras.

10. Close the **3D Properties** and the **3D Animation window**.
11. In the tree at the left select **3D Animation**. From the **right mouse menu** select **Delete Plot**.
12. In the tree at the left select **Window1**. From the **right mouse menu** select **Add Plot** - **3D Animation**.
13. Open the **3D Properties** window.
12. In the **3D Properties window**, from the objects tree select the **Scenery** frame.
13. From the right mouse menu select **Edit**, **Insert Object** and **Sphere**.

Now a sphere is added to the objects tree. A dialog is popped-up showing the ***Object Properties*** of the *Sphere*.

14. Change the properties of the Sphere into:

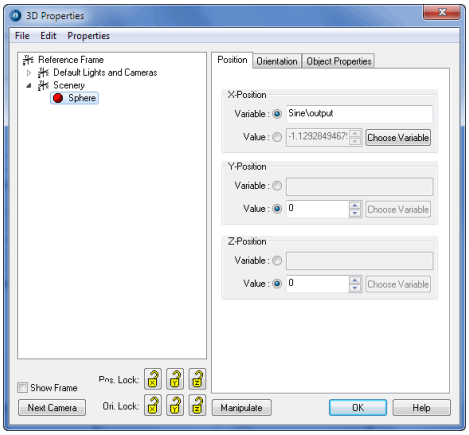
| **Color** | **Scaling Values** |
|---|---|
| Red, Value = 1.0 | Scale X, Value = = 0.5 |
| Green, Value= 0.0 | Scale Y, Value = = 0.5 |
| Blue, Value = 0.0 | Scale Z, Value = = 0.5 |

15. **Close** the *Object Properties* dialog and set the position and orientation of the sphere equal to:

| **Position** | **Orientation (Bryant)** |
|---|---|
| X-position, Variable = *Sine**output* | X, Value = 0.0 |
| Y-position, Value = 0.0 | Y, Value = 0.0 |
| Z-position, Value = 0.0 | Z, Value = 0.0 |

16. For the X-position you have to choose the correct *variable* using the **Choose Variable** button.

Your *3D Properties window* should now look like:

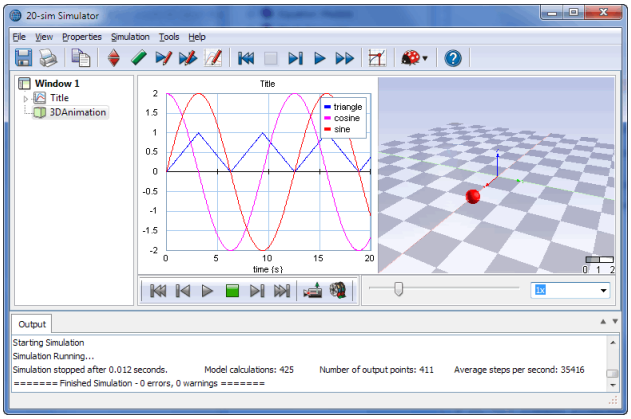



*The X-position of the sphere is coupled to a variable of the 20-sim model.*

17. **Close** the *3D Properties window* and return to the 20-sim *Simulator*.

18. From the **Simulation** menu select **Run** to calculate the plot (or click the blue *Run Simulation* button ).

19. From the **Simulation** menu select **Replay** and **3D Animation** (or click the green *Run 3D Animation* button ) and you will see the *Sphere* move.




*The sphere moving along the X-axis.*

If you have problems producing a good animation, close all windows. Open 20-sim and **load** the model *3DAnimation1* from the *Getting Started\Animation Toolbox* section of the library. This model will show the proper animation of the moving sphere.

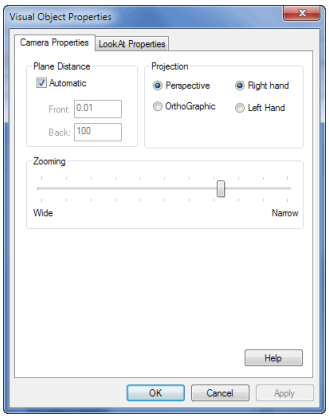
## Zooming

The object is quite far away. We will use the camera zoom to get a close look.

20. Open the **3D Properties** window and from the objects tree select the **Default Lights and Cameras** and **Camera Looking at Origin**.

21. Select the **Object Properties** (tab at the right) and click the **Set Object Properties** button.

22. Pull the **Zooming slider** until you have a closer look at the object and close the Properties window.




*Use the Camera properties to change the zooming level.*

## Other Objects

23. In the tree at the left of the 3D Properties window select the **Scenery** frame.
24. From the menu select **Edit**, **Insert Object** and **Cube**.
25. Open the 3D Properties window and from the objects tree select the **Scenery** frame.
26. From the menu select **Edit**, **Insert Object** and **Cube**.

We would like to see the *Cube* rotate.

27. Select the *Cube* from the objects tree and change the ***Object Properties*** to:

| **Color** | **Scaling Values** | **Position** | **Orientation (Bryant)** |
|---|---|---|---|
| Red = 0.0 | X = 0.5 | X-position = 0.0 | X = 0.0 |
| Green = 1.0 | Y = 0.5 | Y-position = 2.0 | Y = 0.0 |
| Blue = 0.0 | Z = 0.5 | Z-position = -0.25 | Z = *Triangle\output* |

28. Like the *Cube* now **add** a ***Line*** object.

We want the line to rotate along the Y-axis.

29. Select the ***Object Properties*** of the ***Line*** and change the properties to:

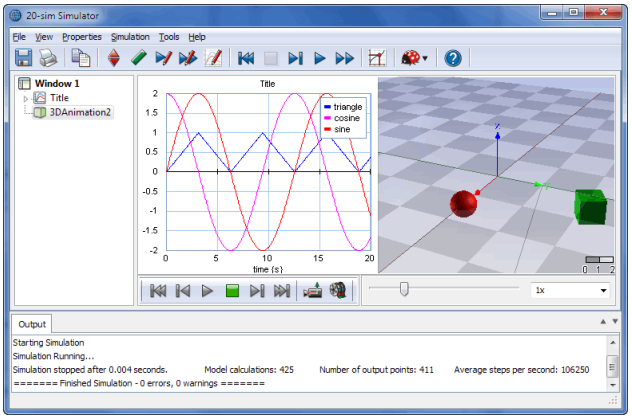
| **Color** | **Start Position** | **End Position** |
|---|---|---|
| Red = 1.0 | X-position = 0.0 | X = *Sine\output* |
| Green = 1.0 | Y-position = 1.0 | Y = 1.0 |
| Blue = 1.0 | Z-position = 0.0 | Z = *Cosine\output* |

30. **Close** the ***3D Properties window*** and return to the 20-sim ***Simulator***.

31. From the **Simulation** menu select **Run** to calculate the plot (or click the blue *Run Simulation* button ).

32. From the **Simulation** menu select **Replay** and **3D Animation** (or click the green *Run 3D Animation* button ).

You will see the sphere move, the cube rotate and the line rotate along the Y-axis. It should look like the figure below:




*Three moving objects.*

If you have problems producing a good animation, close all windows. Open 20-sim and **load** the model *3DAnimation2* from the *Getting Started\Animation Toolbox* section of the library. This model will show the proper animation of the moving objects.

## 10.3 Planetary System

All objects in a 3D Animation can be defined with respect to the **top reference frame**. If you have multiple objects that move relative to the top reference frame but have a fixed position and orientation to with respect to each other, it is useful to introduce additional reference frames. We will demonstrate this by creating a 3D Animation of a planetary system.

### Remove Default Elements

1. **Open** 20-sim and **load** the model **Planetary System** from the *Getting Started \Animation Toolbox* section of the library.

2. Open the ***Simulator***.

3. In the tree at the left select **Window1**.

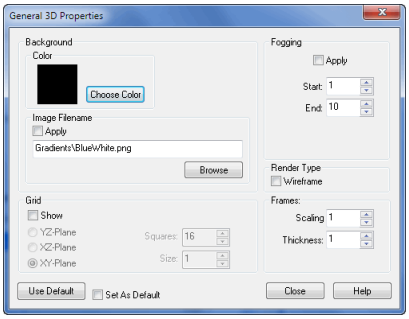
4. From the **right mouse menu** select **Add Plot** - **3D Animation**.

5. In the tree at the left select **3D Animation**. From the right mouse menu select **Plot Properties**.

### Change background and remove the grid

We will remove the grid and change the background.

6. From the **Properties Menu** select **General Properties**.
7. Select the **Choose Color** button and **change the background color to black**.
8. **De-select** the **Apply Image** check box and the **Show Grid** check box.




*In the General Properties window you can select the background color and the grid.*

9. **Close** the **General 3D Properties window.**

## Sun

10. Select the **Scenery *Reference Frame*** and rename it (**Edit** menu) to: ***Sun Reference Frame***.
11. Select the ***Sun Reference Frame***.
12. Insert a **Spot Light**.
13. **Rename** it to ***Sun Light*** and change the **properties** to:

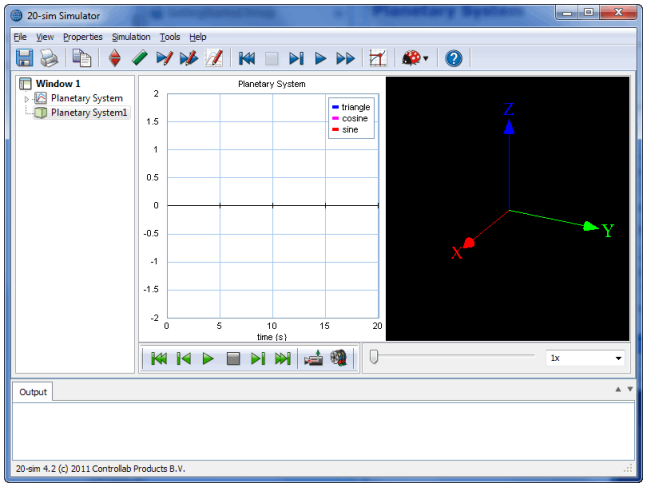
| **Ambient Color** | **Diffuse Color** | **Specular Color** | **Position** |
|---|---|---|---|
| Off | On | On | |
| Red = 0.0 | Red = 1.0 | Red = 1.0 | X-position = 0.0 |
| Green = 0.0 | Green = 1.0 | Green = 1.0 | Y-position = 0.0 |
| Blue = 0.0 | Blue = 1.0 | Blue = 1.0 | Z-position = 0.0 |

14. **Select** the ***Sun Reference Frame*** and insert an **Ambient Light**. Change its **properties** to:

| **Ambient Color** | **Diffuse Color** | **Specular Color** | **Position** |
|---|---|---|---|
| On | Off | Off | |
| Red = 0.5 | Red = 0 | Red = 0 | X-position = 0.0 |
| Green = 0.5 | Green = 0 | Green = 0 | Y-position = 0.0 |
| Blue = 0.5 | Blue = 0 | Blue = 0 | Z-position = 0.0 |

15. **Select** the ***Sun Reference Frame*** and insert a **Camera**. Change the **zooming** until you have a good view.

Now your 3D Animation should look like:




*With an additional camera you will have a new view.*

16. **Select** the ***Sun Reference Frame*** and insert a **Sphere**.
17. **Rename** it to ***Sun*** and change the **properties** to:

| Color | Scaling Values | Position | Orientation (Bryant) |
|---|---|---|---|
| Red = 1.0 | X = 1.0 | X-position = 0.0 | X = 0.0 |
| Green = 1.0 | Y = 1.0 | Y-position = 0.0 | Y = 0.0 |
| Blue = 0.0 | Z = 1.0 | Z-position = 0.0 | Z = 0.0 |

18. **Select** the ***Sun Reference Frame*** and insert a **Circle**.
19. **Rename** it to ***Hot Spot*** and change the **properties** to:

| Color | Scaling Values | Position | Orientation (Bryant) |
|---|---|---|---|
| Red = 0.0 | X = 0.2 | X-position = 0.5 | X = 0.0 |
| Green = 0.0 | Y = 0.2 | Y-position = 0.0 | Y = 1.57 |
| Blue = 0.0 | Z = 0.2 | Z-position = 0.0 | Z = 0.0 |

## Earth

As you might have guessed we are creating the sun. For the earth motion we will use a second reference frame.

20. **Select** the ***Sun Reference Frame*** and from the **Edit** menu select **Copy** and **Paste**.

As you can see a second reference frame is added including objects.

21. From the second reference frame delete the **Sun Light** and **Camera** objects.
22. Click on the **Camera** of the **Sun Reference Frame** to get a correct view again.

23. **Rename** the second reference frame to ***Earth Reference Frame*** and change the properties to:

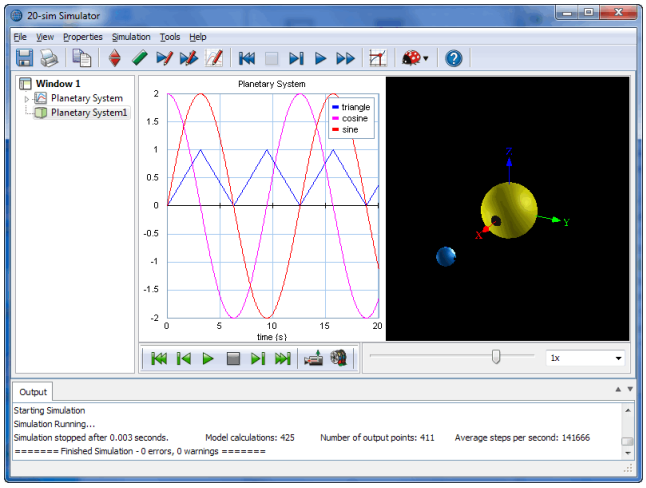
| Scaling Values | Position | Orientation (Bryant) |
|---|---|---|
| X = 0.3 | X-position = *Sine\output* | X = 0.0 |
| Y = 0.3 | Y-position = *Cosine\output* | Y = 0.0 |
| Z = 0.3 | Z-position = 0.0 | Z = *Sine\arg* |

24. From the *Earth Reference Frame* **select** the **Hot Spot** object and change the **name** to ***Launch Site***.
25. **Select** the **Sun** object and change its name to ***Earth***.
26. Change its color to red = 0, green = 0.5 and blue = 1.
27. Select the ***Camera*** from the ***Sun Reference Frame***.

We have added objects and variables which haven't been simulated before. To see a good animation, we first have to run a simulation.

28. **Close** the ***3D Animation Properties window*** and return to the 20-sim ***Simulator***.
29. From the **Simulation** menu select **Run** to calculate the plot (or click the blue *Run Simulation* button ).
30. From the **Simulation** menu select **Replay** and **3D Animation** (or click the green *Run 3D Animation* button ).

Now you will see the earth orbiting the sun. It should look like the figure below:




*The earth orbiting the sun.*

If your objects seem far away, change the zooming of the camera.

## Moon

The same trick as we did with the earth will now be repeated to create a moon.

31. Open the ***3D Properties window***.

32. Select the **Earth Reference Frame** and from the **Edit** menu select **Copy** and **Paste**.

33. **Rename** the objects to ***Moon Reference Frame***, ***Moon*** and ***Lunar Landing Site***.

34. Select the ***Moon Reference Frame*** and change the **properties** to:

| Scaling Values | Position | Orientation (Bryant) |
|---|---|---|
| X = 0.5 | X-position = *Cosine* \output | X = 0.0 |
| Y = 0.5 | Y-position = *Sine* \output | Y = 0.0 |
| Z = 0.5 | Z-position = 0.0 | Z = *Sine*\arg |

35. Select the ***Moon*** and change its color to red = 1, green = 1 and blue = 0.5.

36. **Select** every reference frame and de-select the option **Show Frame**.

You can also press the **Ctrl-key** and de-select the *Show Frame* option of the top frame. This will make that all underlying frames will have the *Show Frame* option de-selected.

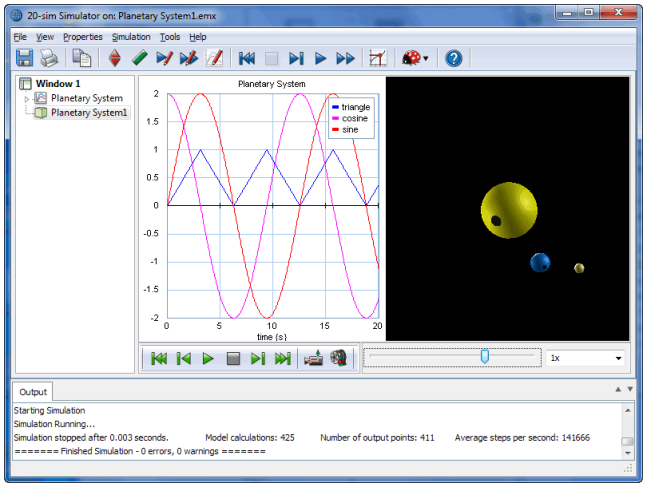
37. Select the ***Camera*** from the ***Sun Reference Frame***.

## Viewing the Animation

38. **Close** the ***3D Properties window*** and return to the 20-sim ***Simulator***.

39. From the **Simulation** menu select **Replay** and **3D Animation** (or click the green *Run 3D Animation* button ).

Now you should see the earth orbiting the sun:




*The earth orbiting the sun and the moon orbiting the earth.*

40. In the *Simulator,* from the **View** menu select **Numerical Values**.

With the *Numerical Values window* you can inspect the values of the simulation plot. At the same time you will see the corresponding view in the *3D Animation window*. If you use the slider of the *Numerical Values window* you can seen the earth and the moon move.

41. Use the slider of the *Numerical Values window* to move the *earth* and the moon.

## Switching Cameras

42. In the ***3D Properties window*** select the **Earth Reference Frame** and insert a **Camera** object. Change the **zooming** until you have a good view.
43. **Change** the name of the Camera into ***Earth Camera.***
44. In the 3D Animation window click the **Play** button to see the sun and moon moving around the earth.
45. Switch between cameras by clicking the **Camera** button.
46. If you have problems producing a good animation, close all windows. Open 20-sim and **load** the model *Planetary System1* from the *Getting Started\Animation Toolbox* section of the library. This model will show the proper animation of the planetary system.

# 11 Control Toolbox

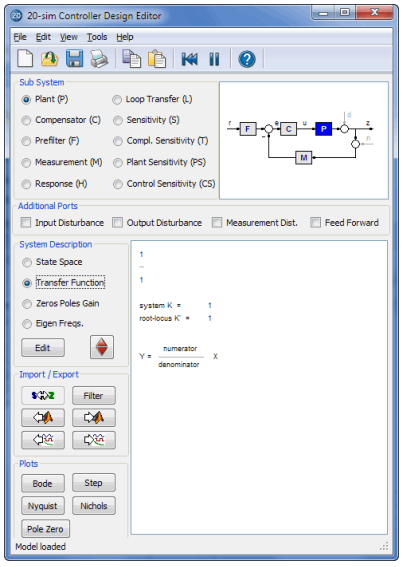
## 11.1 Control Toolbox

The Control Toolbox of 20-sim contains several tools that can aid you in developing controllers for your modeled machines, the *Controller Design Editor*, the *Filter Editor* and the *Neural Network Editors*.

### Controller Design Editor

1. In the Editor from the **Tools** menu select **Control Toolbox** - **Controller Design Editor**.

The *Controller Design Editor* is a specialized tool for the design of feedback control systems. A feedback structure of subsystems is presented with a linear plant, controller, measurement and pre-filter. Also the open-loop and closed-loop gains and the sensitivities are available.



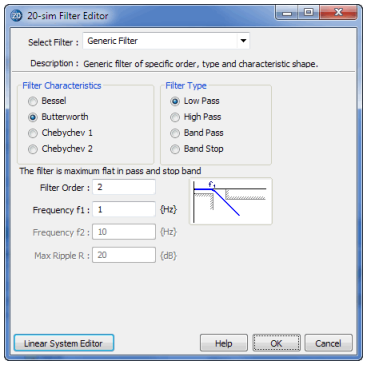

*The Controller Design Editor.*

You can edit your controller as an ABCD State Space system, a Transfer Function or in a Zero Pole Gain form. Changes in one of the subsystems directly update all open plots and dialogs. For instance, adapting the controller gain immediately changes poles and zeros of the closed-loop system and the overall step response. The integration within 20-sim and linear system exchange with MATLAB makes this editor a powerful tool for designing feedback control systems!

### Filter Editor

2. In the Editor from the **Tools** menu select **Control Toolbox** - **Filter Editor**.

With the *Filter Editor* you can create your own linear filters according to your specifications. Available filters are Bessel, Butterworth and ChebyChev filters where you can specify the order and characteristic frequencies. A choice can also be made from PID, lead/lag, or notch filters.

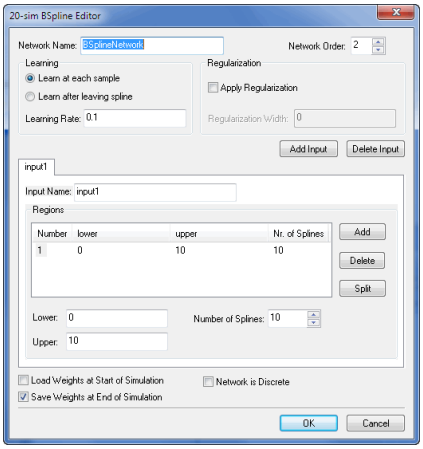



*The Filter Editor.*

## Neural Network Editors

3. In the Editor from the **Tools** menu select **Control Toolbox** - **B-Spline Network Editor**.

The 20-sim Control toolbox supports two well-known networks: Adaptive B-Spline Networks and Multi-Layer Perceptron Networks.




*The B-Spline Editor.*

These neural networks must be trained by repeatedly presenting examples to the network. Each example includes both inputs and desired outputs. Based on the error between desired outputs and the real network output, the neural network adapts the weight of each neuron according to a user-defined learning rate. If the response is accurate enough, you can save the weights of the neurons, to use the neural network in your controller. An unlimited number of neurons may be used.

# 12 Frequency Domain Toolbox

## 12.1 Frequency Domain Toolbox

The 20-sim *Frequency Domain Toolbox* consists of the *Linear System Editor*, *FFT Analysis, Model Linearization* and *Dynamic Error Budgeting* functionality.

### Linear System Editor

The *Linear System Editor* is a specialized tool for the design and analysis of linear systems. The editor supports continuous-time and discrete-time SISO systems using various representations.

The graphical interface allows you to edit a linear system in any desired form: ABCD state space, Transfer Function or Zero Pole Gain. Analyzing the Step response, Bode plot, Nyquist diagram, Nichols chart and Pole-Zero plots allows you to quickly evaluate system behavior. Phase, gain and modulus margins are calculated, as well as rise time, overshoot and steady state value.

Input can originate from a 20-sim linear system model, 20-sim filter or control editor, MATLAB workspace, or user input. You are able to generate output for all 20-sim editors, the clipboard and the MATLAB workspace.

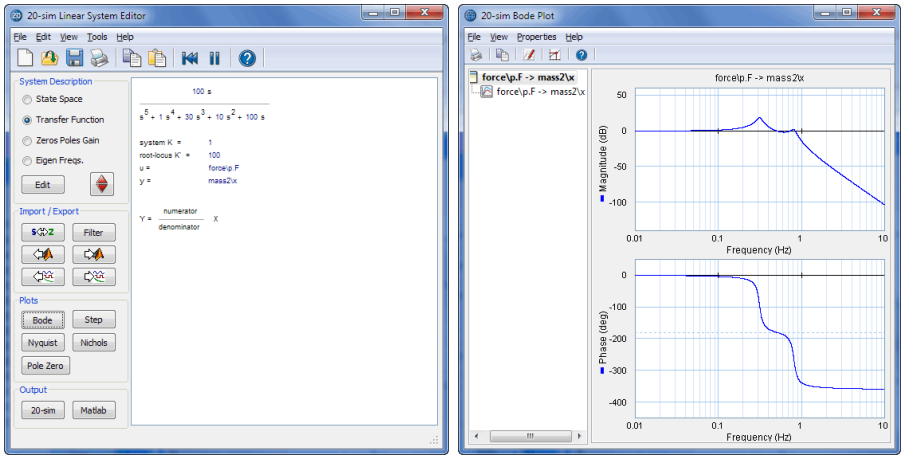
### Fast Fourier Transform

Fast Fourier Transforms (FFT) can be applied to any time-domain plot in 20-sim. Either simulation results or measurement data will do. When the data is not equally spaced, linear interpolation is first applied after which the Fast Fourier Transform is used to calculate the frequency contents. Three representations are supported: Amplitude and Phase plot, Frequency plot and Power Spectral Density plot.

### Model Linearization

Any 20-sim model can be linearized to state space form. If possible, linearization will be performed symbolically. Otherwise linearization will be performed numerically.

The resulting state space model is shown in the 20-sim *Linear System Editor*. The *Linear System Editor* is a specialized tool for the design and analysis of linear systems. The editor supports continuous-time and discrete-time SISO systems using various representations. Standardized plots enable you to quickly evaluate system behavior.



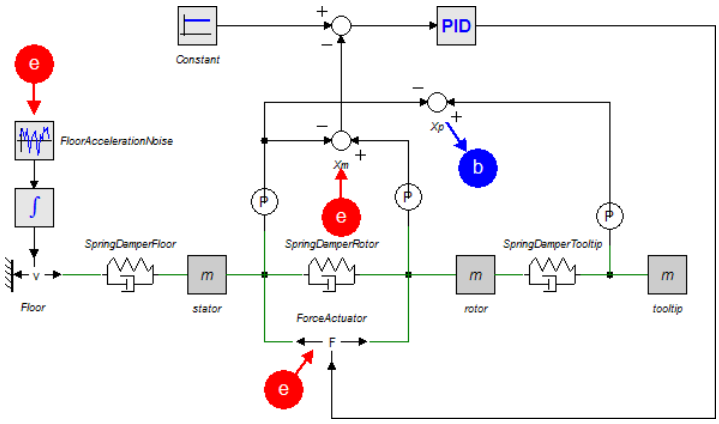

*In the Linear System Editor you can generate bode plots.*

## Features

- Editing as ABCD state space, Transfer Function or Zero Pole Gain with automatic transformation between these forms.
- Transfer between continuous-time and discrete-time representation.
- View characteristic properties like eigenfrequencies and damping.
- Handles numeric and symbolic models.
- Various plot options: Step Response, Bode Plot, Nyquist Diagram, Nichols Chart and Pole-Zero Plot.

## Dynamic Error Budgeting

The performance of precision machines is mostly limited by the disturbances that are injected in these machines. These disturbances are often stochastic in nature. Dynamic Error Budgeting is a method whereby the effect of these disturbances on the final performance can be calculated. The advantage of this method is that it enables the designer to enter the contributions of the individual disturbances and view and optimize the overall machine performance.



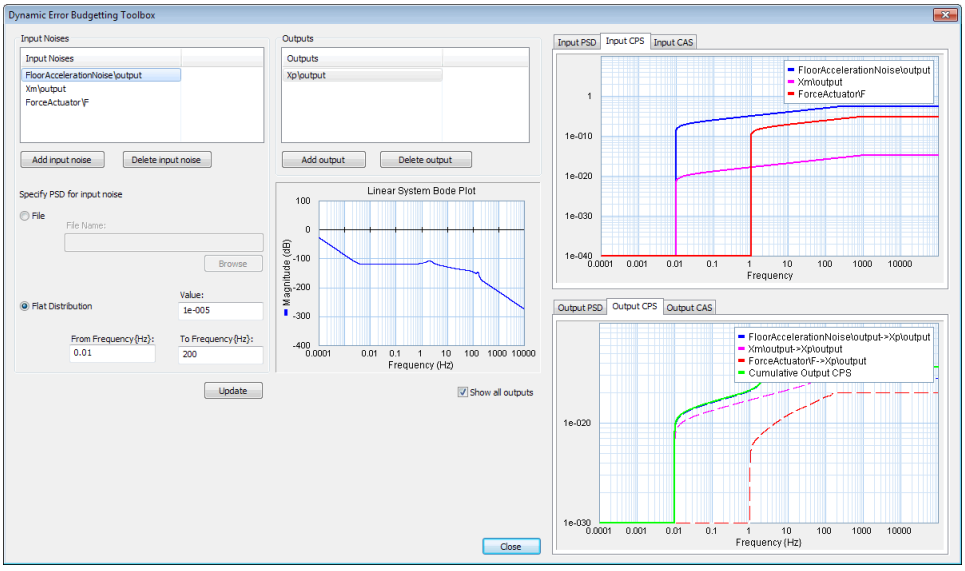

*The Dynamic Error Budgeting toolbox shows the total error as a result of injected disturbances.*

## Running the Toolbox

You can open the Dynamic Error Budgeting tool in the *Simulator*:

1. **Open** 20-sim and **load** the model *Dynamic Error Budgeting* from the *Getting Started \Frequency Domain Toolbox* section of the library.

2. In the *Simulator* from the **Tools** menu select **Frequency Domain Toolbox** and **Dynamic Error Budgeting**.




*The Dynamic Error Budgeting tool.*

The tool allows you to enter disturbances (as power spectral density) in the *Input Noises* section.

3. For each disturbance you have to select a corresponding variable by clicking the **Add input noise** button.

Each disturbance is effectively a summation to the chosen variable, just like closed loop linearization. You can inspect each disturbance in the graph on the top right.

Next you have to select an output variable, where the result of the disturbances is calculated.

4. Select the output by clicking the **Add Output** button.

In the graph on the bottom right you can see the resulting error at the selected output as a result of the disturbances. The error is given in the form of a power spectral density (PSD) and cumulative power spectral density (CPS). The square root of the final value of the CPS is equal to the standard deviation of the output error.

# 13 Mechatronics Toolbox

## 13.1 Mechatronics Toolbox

The *Mechatronics Toolbox* includes the *Motion Profile Wizard*, the *CAM Wizard* and the *Servo Motor Editor*.

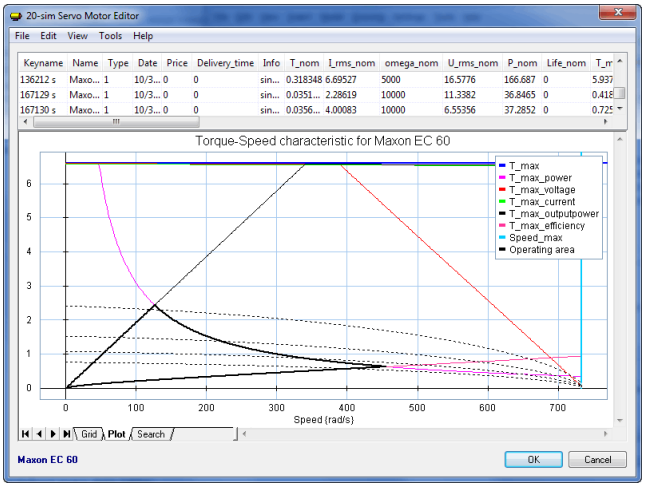
### Servo Motor Editor

The 20-sim Servo Motor Editor is a program that helps engineers to choose the proper servo motor for any electromechanical system:

- Brush DC (Iron Armature Motor, Hollow Rotor Motor, Disc Armature Motor)
- Brushless DC
- AC synchronous
- AC synchronous linear

In cooperation with motor manufacturers, motor data tables have been created for the *Servo Motor Editor*. The performance of every motor can be shown by the torque speed curve. The *Servo Motor Editor* can generate dynamic models for the simulation program 20-sim. In this program you can simulate the thermal and dynamic behaviour of the servo motor in combination with control loops and dynamic loads.

Any engineer involved in the design of electromechanical machines can benefit from the *Servo Motor Editor*. Precious time and money can be saved by finding the optimal servo motor in a few minutes, without risking overheating or under-powering.




*The 20-sim Servo Motor Editor.*

### Features

- Support of predefined motor data tables.
- Add your own motors to the data table.
- Quick search by multiple parameters.

- Torque-Speed curves with: line of maximum current, maximum torque, maximum voltage, maximum speed, maximum power, maximum efficiency and maximum power output.
- Safe Operating Area, Desired Operating Area.
- Generate dynamic models for the simulation program 20-sim. These models include thermal effects of the coils and housing, electrical losses through dissipation, magnetic losses through hysteresis, eddy currents and cogging.
- Show torque speed curve with dynamic load curve.
- Template models, covering most commercial control schemes and many mechanical loads.

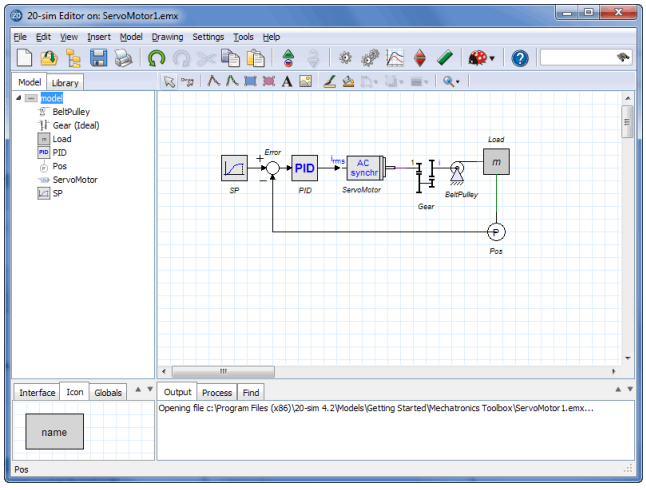
### Application Areas

- Control Design
- Industrial Equipment and Machinery
- Precision Engineering
- Simulation
- Utilities and Energy
- Vibration Analysis and Control

## 13.2 Servo Motor Editor

This section demonstrates how to use the Servo Motor Editor to select the correct motor for a drive train. The Servo Motor Editor is part of the Mechatronics Toolbox.

You can use the Servo Motor Editor to create models that include permanent magnet servo motors. In the figure below you see a PID controlled AC synchronous motor driving a load through a gearbox and beltdrive. In this example we will show how to create and use such a model.

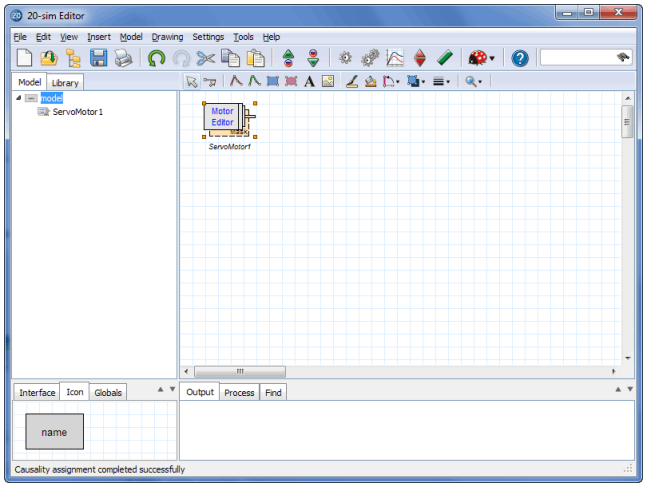



*PD controlled servo motor with gearbox and belt.*

### Model

1. Open 20-sim and select **File**, **New** and **Graphical Model**.
2. From the **Tools** menu select **Mechatronics Toolbox** and **Servo Motor Editor.**

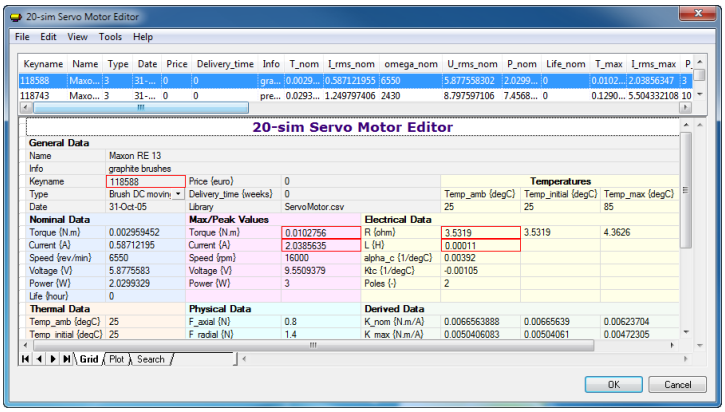
A servo motor model will be inserted:




*A servo motor inserted.*

3. The **Servo Motor Editor** will be opened automatically. If this does not happen, force the editor to open by clicking the **Go Down** button.

The editor will show the default motor list and the parameters of the selected motor.



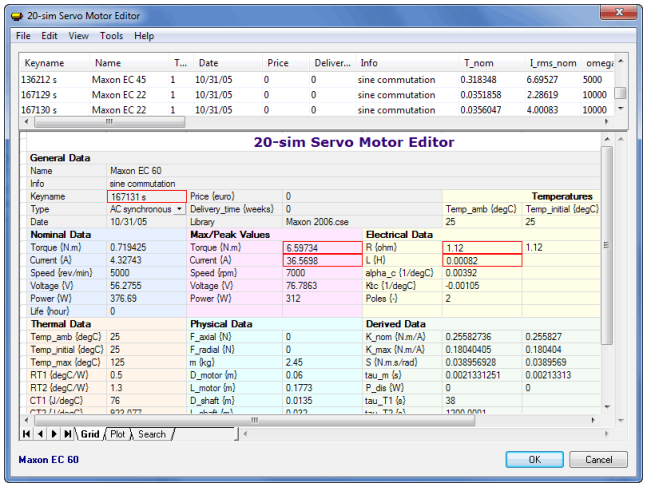

*You can choose from a list of existing servo motors in the Servo Motor Editor.*

4. From the **File** menu choose **Open Database**.
5. Load the file *Maxon 2006.cse*. You can find this file in the folder *C:\Program Files \20-sim 4.7\Tools\Servo Motor Dynamics*

Use *C:\Program Files (x86)\20-sim 4.7\Tools\Servo Motor Dynamics* on 64-bit versions of Windows

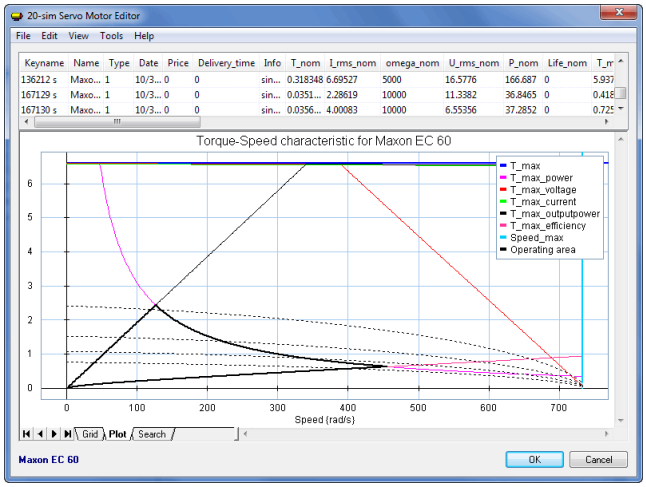
Now the Servo Motor Editor will show a list of Maxon motors.

6. From the list choose the Maxon EC 60 motor with keyname **167131 s**.




*Display the characteristic parameters of a selected motor.*

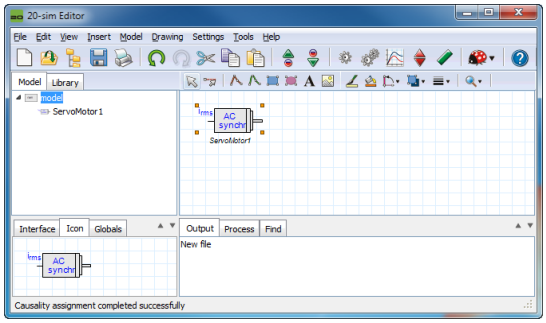
7. Click the **Plot tab** to inspect the torque speed curve.




*Show a torque speed plot of the selected motor.*

8. Click the **OK** button to close the editor and generate the dynamic model.

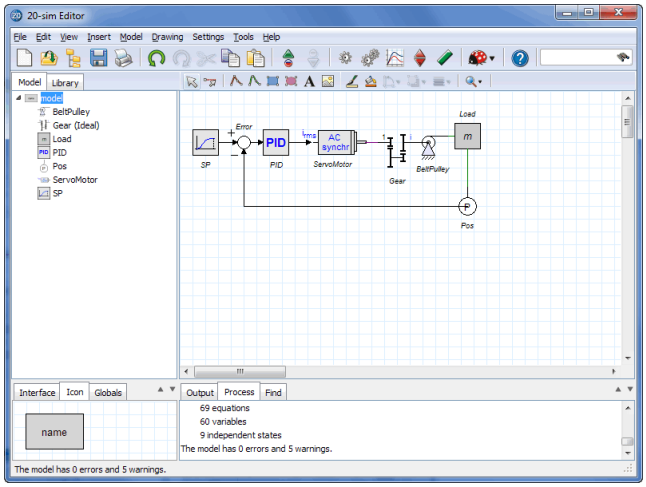
The 20-sim editor will now look like:




*The selected motor is now inserted in 20-sim.*

9. **Add** the following models and **connect** them until your model looks like the figure below. Try to give the same **submodel names** as indicated in the figure.

| **model library** | **model** |
|---|---|
| *Iconic Diagrams\Mechanical\Translation\Sensors* | *PositionSensor-Absolute* |
| *Iconic Diagrams\Mechanical\Translation\Components* | *Mass* |
| *Iconic Diagrams\Mechanical\Translation\Transmission* | *BeltPulley* |
| *Iconic Diagrams\Mechanical\Rotation\Gears* | *Transmission (Ideal)* |
| *Signal\Block Diagram* | *PlusMinus* |
| *Signal\Sources* | *SignalGenerator-Cycloid* |
| *Signal\Control\PID Control\Continuous* | *PID* |




*The complete model of the belt driven load.*

10. From the **File** menu click **Save as**. Store the model in a temporary directory (e.g. *C:\temp*) using the name *ServoMotor.emx*.

If you have problems creating the model, load the model *ServoMotor1* from the *Getting Started\Mechatonic Toolbox* section of the library.

## Simulation

11. In the *Editor* toolbar from the **Model** menu select the **Start Simulator** command.
12. In the Simulator, click the **Parameters** command from the Properties menu.
13. As you will see the parameter values for the AC motor have been filled in automatically by the Servo Motor Editor. **Enter** the other **parameters** as:

| **Parameter** | **Value** |
|---|---|
| *Load\m* | 20 |
| *PID\kp* | 5000 |
| *PID\tauD* | 0.05 |
| *PID\beta* | 0.1 |
| *PID\tauI* | 20 |
| *BeltPulley\radius* | 0.1 |
| *Gear\i* | 20 |
| *SP\start_time* | 1 |
| *SP\amplitude* | 1 |
| *SP\stop_time* | 2 |

14. From the **Properties** menu select the **Run** command and change the default values to:

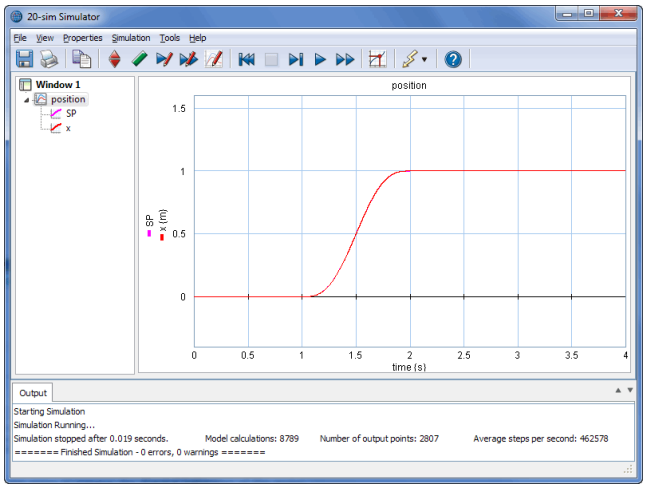
| | |
|---|---|
| *Start* | 0 |
| *Finish* | 4 |
| *Method* | Vode Adams |

15. In the tree, select the plot (**model**). From the **right mouse menu**, select **Rename** and rename it to *position*.
16. From the **right mouse menu**, select **Plot Properties**.
17. From the **Properties** menu select the **Plot** command. Add the following plot variables to the **Y-tab**.

| **Variable** | **Label** |
|---|---|
| *SP\output* | SP |
| *Pos\x* | x |

18. Click on the **Shared Axis** option to select it.

19. **Close** the Plot *Properties Editor* and **run** a Simulation. It should look like the next figure.




*Simulation of the drive system.*

As you can see the motor is able to drive a load of 20 kg. Now we are going to inspect the thermal behaviour of the motor.

19. In the tree at the left click on ***Window 1*** to select it.

20. From the right mouse menu select **Add Plot** - **Plot**.

Now a second plot will be visible.

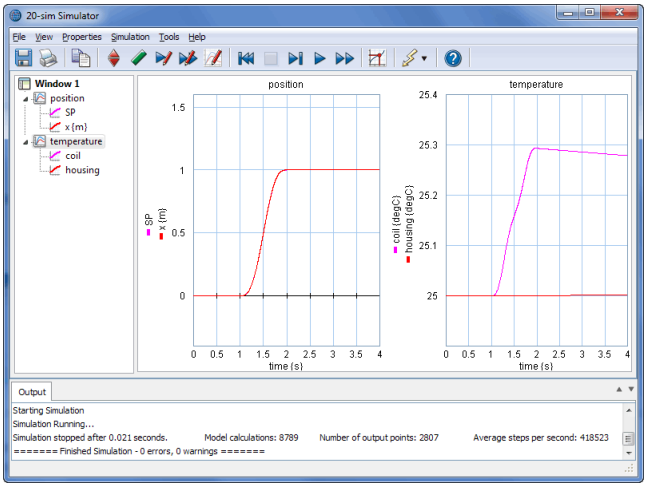
20. **Rename** the plot to *temperature*.

21. Open the **Plot Properties Editor** (Properties - Plot). Add the following plot variables to the **Y-tab**.

| **Variable** | **Label** |
|---|---|
| *ServoMotor\Temp_coil* | coil |
| *ServoMotor\Temp_housing* | housing |

22. Click on the **Shared Axis** option to select it.

23. **Run** a new **simulation.** The *Simulator* will look like the next figure.




*In the Simulator you can open additional plots.*

The coils heat up immediately and the housing heats up gradually. We have to run a simulation for a longer time, to see if the motor will overheat. We can also inspect the torque speed curve to inspect the thermal behaviour.

24. From the **View** menu choose **New Plot Window**. This will open a new plot.

25. Open the **Plot Properties Editor** (Properties - Plot). Add the following plot variables.

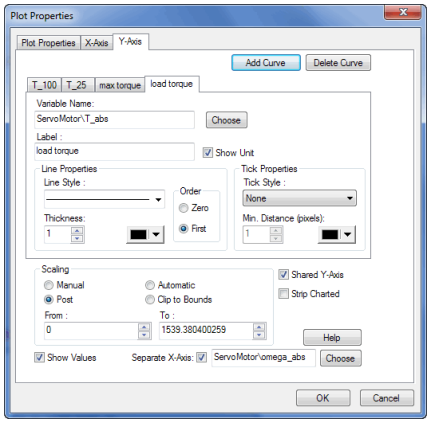
| **x-axis** | **Variable** | **Label** |
|---|---|---|
| | *ServoMotor\omega_range* | speed |
| **y-axis** | **Variable** | **Label** |
| | *ServoMotor\T_100* | T_100 |
| | *ServoMotor\T_25* | T_25 |
| | *ServoMotor\Torquemax* | max torque |
| | *ServoMotor\T_abs* | load torque |

For the variable *ServoMotor\T_abs* we will choose a separate x-axis.

26. Select the **Separate X-Axis** option at the bottom of the Plot Properties Editor.

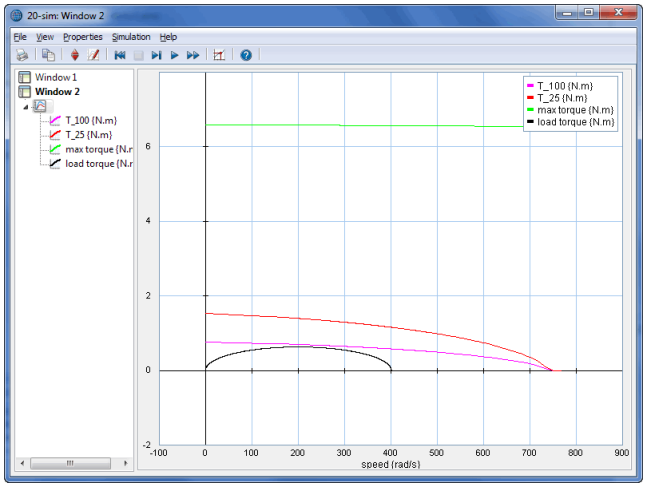
27. Click the **Choose** button next to the Separate X-Axis option and select the variable *ServoMotor\omega_abs*.

Your Plot Properties Editor should now look like:




*You can choose a separate x-axis for every plotted variable.*

28. **Run** a new simulation. The extra plot will look like the next figure.




*Torque speed plot with the duty cycle lines and the load torque.*

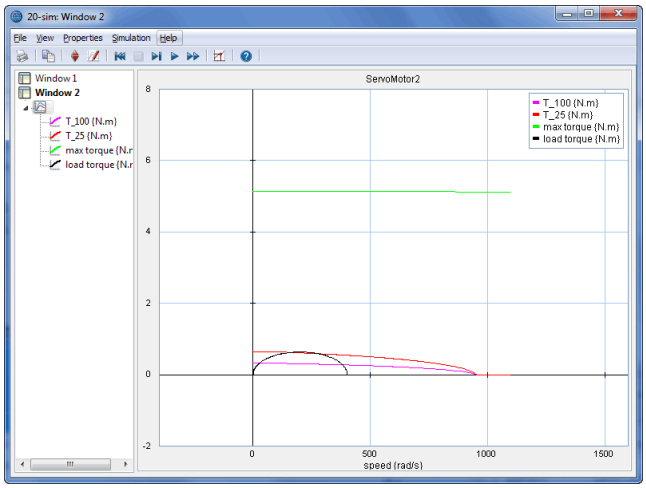
The load torque, i.e. the torque that is applied by the motor to drive the load, is completely under the 25% duty cycle curve. This means the motor will never overheat when once every 4 sec there is a step change of 1 s. The motor curve is even under the 100% duty cycle. This means it wouldn't overheat if we would apply 4 steps in 4 s. If we stick to the 25% duty cycle, we might choose a lighter motor.

29. Go to the Editor. Select the **ServoMotor model** and click **Go Down**.

30. In the **Servo Motor Editor** choose the motor with keyname **252463 s**.

This is a motor with less power (156 W vs. 312W), a smaller maximum torque (5.14 Nm vs. 6.59 Nm) and a higher maximum speed (10000 rpm vs. 7000 rpm)

31. Click **OK** to close the Servo Motor Editor.
32. In the Editor from the **Mode**l menu click the **Check Complete Model** command. This will ensure that the new motor will be used in the simulation.
33. In the Simulator from the **Simulation menu** choose **Clear** - **All Runs.**
34. Now **run** a new simulation. The torque speed plot will now look like:

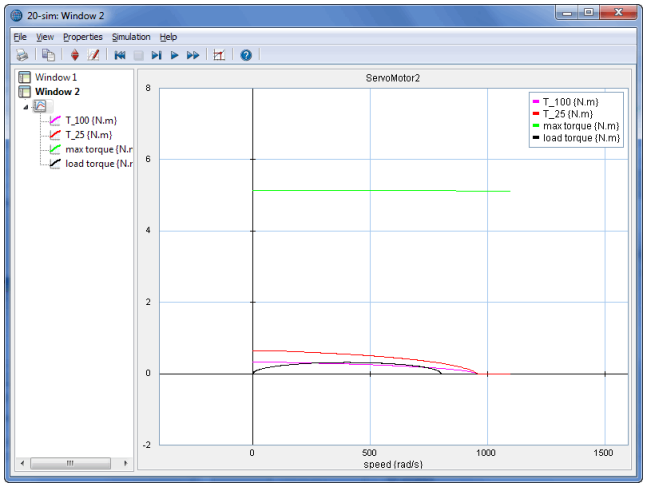



*The motor torque crosses the 25% duty cycle line. This means the motor will overheat.*

The load curve now crosses the 25% duty cycle line. This means the motor will heat up higher than the allowed maximum. To prevent this from happening we will use the higher maximum speed of the motor. This can be done by changing the gearbox ratio.

35. In the **Simulator**, from the **Properties** menu click the **Parameters** command. **Change** the gearbox ratio *Gear\i* to 40.

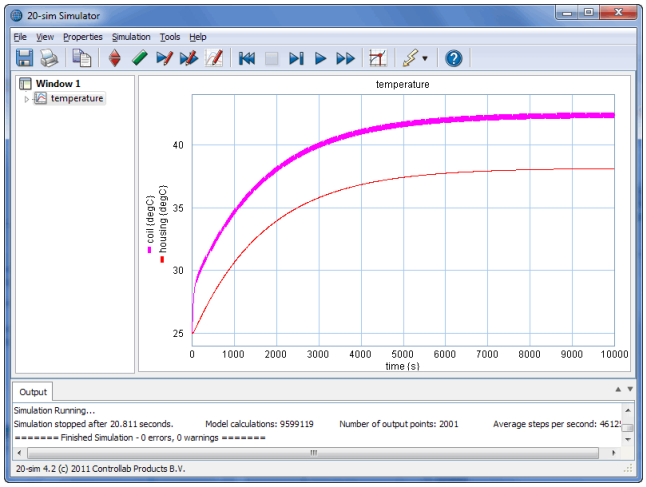
36. **Clear** the previous simulation and **run** a new simulation. The torque speed plot will now look like:




*Changing the gearbox ratio will keep the motor torque under the 25% duty cycle line.*

Now the load curve does not cross the 25% duty cycle line anymore. You can also load the model ServoMotor2 from the Getting Started\Mechatonic Toolbox section of the library to show the final result.

You can load the model *ServoMotor3.emx* from the Getting Started\Mechatonic Toolbox section of the library, to see a long simulation of the motor temperature. In this model the setpoint generation is performed by a motion profile block which outputs a repeating signal with a duty cycle of 25%. As you can see from the plot the coil temperature increases to 43 °C which is far below the maximum temperature of 125 °C.




*A simulation of the motor temperature.*

# 14 Real Time Toolbox

## 14.1 Real Time Toolbox

The *Real Time Toolbox* provides you with C-code generation tools and templates for all kinds of different targets and platforms.

With 20-sim you are able to generate C-code as well as MATLAB code for every 20-sim model that you have created, whether it is a continuous or discrete-time model. The C-code can be created for several targets, like MATLAB Simulink, but also for Rapid Prototyping (RP) systems and Hardware-In-the-Loop (HIL) simulations.

### Matlab Simulink

Generating C-code for use in MATLAB Simulink also includes a submodel block with input and output terminals. 20-sim uses the MEX-compiler, to compile this code directly into an S-function. These S-functions can also be used in the Real Time Workshop in order to generate code for a specific platform, for instance xPC Target.

### C-code

20-sim can generate standalone C-code for use in C and C++ programs. The generated C-code is supplied with several fixed step simulation algorithms to assure that it will run in real-time. The Euler and RungeKutta 4 method are supported by default.

All C-code templates are open and can be adapted by the user to assign compilers, run ftp-sessions and automate almost everything between the 20-sim code generation and the actual running of the code on a (remote) machine. During C-code generation support code is generated for 20-sim operators like matrix calculations and trigonometric functions.

### 20-sim 4C

With the Real-Time Toolbox you can generate code for 20-sim 4C. 20-sim 4C is a program that is sold separate from 20-sim. 20-sim 4C is a prototyping environment that enables you to connect 20-sim models to physical systems. The models can be executed as real-time C-code on hardware like PC's or ARM-9 based processor boards. This enables you to perform various tasks:

- **Measurement and Calibration:** From 20-sim 4C you can export C-code that will operate and read sensors.
- **Machine Control:** With 20-sim 4C you can export code to external targets to control the operation of machines. In 20-sim 4C you can start and stop the controller and change parameters during run-time.

- **Rapid Prototyping:** 20-sim models can be exported to 20-sim 4C with the click of a button and executed on a target with a second click. This makes 20-sim 4C a valuable tool for rapid prototyping.


*20-sim 4C allows you to run C-code, generated in 20-sim, on real machines.*

More information on 20-sim 4C can be found on the 20-sim website www.20sim.com.

# 15 Time Domain Toolbox

## 15.1 Time Domain Toolbox

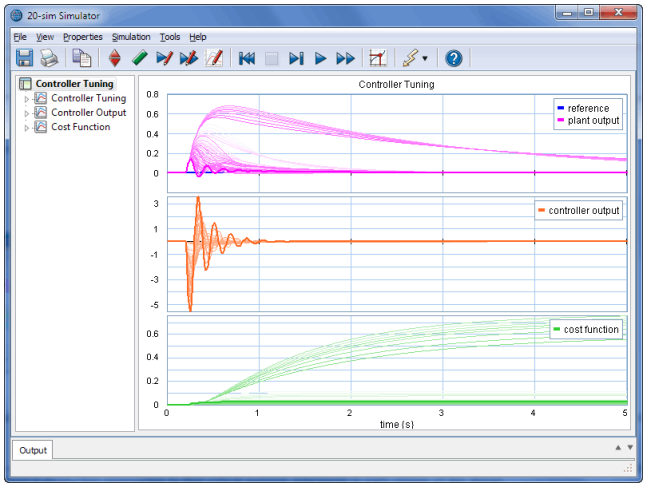
During simulation, the time domain behavior of a model is calculated. Based on this time-domain behavior, the model can be analyzed. A set of powerful methods for time domain analysis is available in 20-sim.

### Parameter Sweeps

Parameter sweeps are multiple simulations with a variation of parameter values. It is a quick method to see how your model behavior depends on the parameter values.

### Parameter Optimization

Using Optimization, you can maximize the performance of your model by varying specified model parameters. Predefined or user defined cost functions can be used as a measure of model performance. A number of well known optimization methods can be used to minimize or maximize these cost functions.




*Using optimization to find controller parameters with optimal disturbance rejection.*

### Curve Fitting

With Curve Fitting you can fit model performance to a given result by variation of parameters. It is a very useful method to optimize model parameters when measurement data is available.

### Sensitivity Analysis

Sensitivity Analysis is used to investigate the effect of parameter variation on model performance. The change in performance divided by the parameter change is plotted in a table. Sensitivity Analysis is a well-known method in production engineering to find critical systems tolerances in early stages of the design

## Monte-Carlo Analysis

With Monte-Carlo Analysis, you can perform a predefined number of simulation runs with statistical variation of parameter values. Results can be shown as histograms or inspected numerically. Monte Carlo Analysis is a powerful method to find out worst case model behavior.

## Variation Analysis

With Variation Analysis you can find the statistical range of parameter values that yield a model with a certain performance level. Variation Analysis is a powerful method to find a measure for system tolerances in the early stage of the design.

# 16 Scripting Toolbox

## 16.1 Introduction

20-sim scripting allows you to run tasks in 20-sim automatically using specialized scripting functions. With these functions you can open models, run simulations, change parameters, store results and much more. Scripting support was introduced in 20-sim 4.4.


*20-sim session automated by a script in Octave, Matlab or Python.*

These scripting functions are not very useful stand-alone, but you can use them to write scripts to automate various tasks in 20-sim.
20-sim provides a set of script functions for numerical computation environments like Octave and Matlab and for the Python programming language (since 20-sim 4.6).
The 20-sim scripting functions are based on XML-RPC calls, so any other programming language with support for XML-RPC can be used to automate various 20-sim steps.

In this chapter you will learn how to run basic scripts and make scripts on your own.

The next sections explain:

- **Installation for scripting**:
  - *20-sim:* enabling the XML-RPC scripting interface in 20-sim
  - *Octave:* installing Octave as scripting environment
  - *Matlab:* installing Matlab as scripting environment
  - *Python:* installing Python as scripting environment
- **Prepare Scripting Folder:** extract the 20-sim scripting functions and documentation to your work directory.
- **Basic Script:** run your first script and see how a basic script is made in Octave/ Matlab or Python.
- **Advanced Scripts:** see how you can expand the basic script to perform more advanced tasks in Octave/Matlab.
- **Writing your own Scripts:** How to write your own scripts in Octave/Matlab or Python.

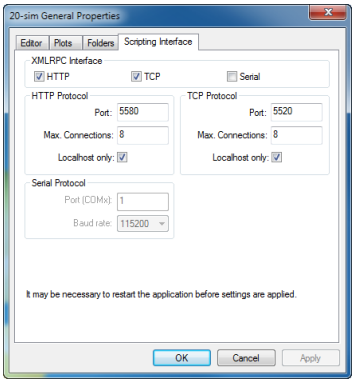
**Note:** Scripting is not supported in the 20-sim Viewer/Demonstration Version. If you would like to try the scripting functionality, you will need licensed 20-sim version or a trial license.

## 16.2 Installation for Scripting: 20-sim

20-sim uses XML-RPC as a protocol to communicate scripting functions with external packages. By default the XML-RPC interface is turned on only for your local computer (listening on the TCP ports **5580** and **5520)**.

To enable/disable and configure the 20-sim scripting support:

1. Open **20-sim**.
2. Go to **Tools/Options** and select the **Scripting Interface** tab.




*Scripting Interface settings tab.*

3. To enable the 20-sim scripting support, make sure that the **HTTP** and **TCP** checkboxes under *XMLRPC Interface* are enabled.

By default, 20-sim will only accept scripting connections from your local computer (**Localhost only** option is enabled).

Your firewall may generate a warning message and ask you to allow network communication for 20-sim.

4. Set the **firewall** to **allow communication**.

## 16.3 Scripting in Octave/Matlab

### Installation for Scripting: Octave

### What is Octave?

GNU Octave is a high-level language, primarily intended for numerical computations. The package is open source and can be freely distributed. GNU Octave offers functionality similar to Matlab users. If you have experience with Matlab, running Octave will be very easy. Users with no experience with Octave nor Matlab are advised to read a proper introduction to GNU Octave first. You will find lot of pages and videos on the Internet.

## Installation

The Windows versions of Octave 4.4.0, 4.2.x, 4.0.x, 3.8.x and 3.6.x have been tested with 20-sim scripting at the time of this release.
Note that for most versions of Octave only the 32-bit Octave is supported. Only the 64-bit version of Octave 4.2.1 is currently supported.

1. Go to: https://www.20sim.com/product/octave.html and follow the latest instructions.

If this web page is unavailable, you can follow the instructions below:

First choose the Octave version you wish to install and go to the corresponding section below:

### Octave 4.4.x / 4.2.x / Octave 4.0.x

1. Go to: https://www.gnu.org/software/octave/
2. Go to the download page and download the Windows installer (direct link: https://ftp.gnu.org/gnu/octave/windows/octave-4.4.0-w32-installer.exe)
3. **Run** the installer and follow the wizard. The steps below assume default installation settings.
4. **Octave 4.0.x only:** Unfortunately Octave 4.0.x has a Windows specific bug in its internal run() function. This bug is resolved in Octave 4.2.
   For Octave 4.0.x you will need to manually replace the default run() implementation with a corrected version. **Copy** the file:

   ```
   C:\Program Files (x86)\20-sim 4.7\Scripting\Octave-patch\4.0.0\run.m
   ```

   or on 32-bit versions of Windows:

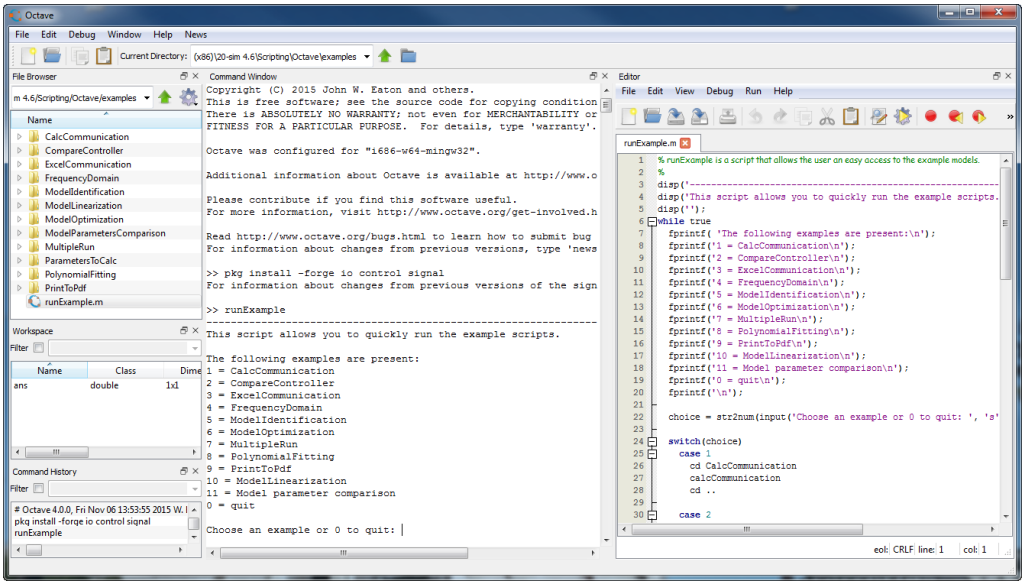
   ```
   C:\Program Files\20-sim 4.7\Scripting\Octave-patch\4.0.0\run.m
   ```

   to:

   ```
   C:\Octave\Octave-4.0.0\share\octave\4.0.0\m\miscellaneous\run.m
   ```

5. **Launch Octave** from the Start menu or using the script: `C:\Octave\Octave-4.x.x\octave.bat`
6. **Execute** the following commands to install packages io, control and signal:

   ```
   pkg install -forge io
   pkg install -forge control
   pkg install -forge signal
   ```




*The Octave GUI.*

Your Octave installation is now ready to use.

## Octave 3.8.x

1. Go to: http://www.20sim.com/downloads/files/ThirdParty/octave-3.8.2-2-installer.exe
2. **Run** the installer and follow the wizard.
3. **Launch Octave** using the script: C:\Octave\Octave-3.8.2\octave.bat
4. **Execute** the following commands to install packages io, control and signal:

```
pkg install -forge io
pkg install -forge control
pkg install -forge signal
```

Your Octave installation is now ready to use.

## Octave 3.6.x

1. Go to the Octave download site (http://sourceforge.net/projects/octave/.).
2. Click on the **Files** tab and click on **Octave Windows Binaries**.
3. Select the **Octave 3.6.4 for Windows MinGW installer**.
4. Now you can **download** the files **Octave3.6.4_gcc4.6.2_yyyyxxxx.7z** (Octave Installation) and **Octave3.6.4_gcc4.6.2_pkgs_yyyyxxxx.7z** (Octaveforge Packages).

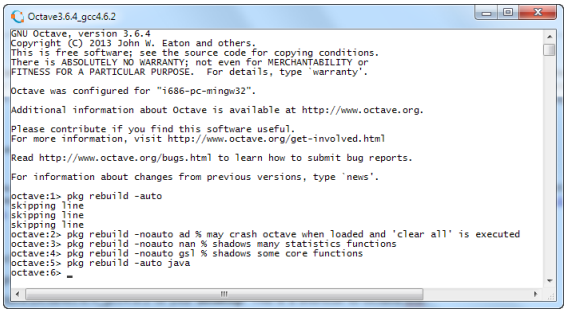
5. Create an installation directory which doesn't have space chars (i.e. C:\Octave).
6. **Unzip** the file **Octave3.6.4_gcc4.6.2_yyyyxxxx.7z** and **copy** it to the installation directory.
7. Copy the **shortcut link** `C:\Octave\Octave3.6.4_gcc4.6.2.lnk` to your **desktop**. This is a shortcut to start Octave.exe.

*Note:* Unzipping can be done with programs like 7-zip (http://www.7-zip.org/)
*Note:* There is a bug with Windows 8 and running Octave. In order to use Octave start Octave with `octave.exe -i --line-editing`. See the Octave wiki webpage for more information.

8. **Unzip** the file **Octave3.6.4_gcc4.6.2_pkgs_yyyyxxxx.7z** and **copy** it to the installation directory.
9. **Launch Octave** (e.g. the link to Octave.exe).
10. **Execute** the following five rebuild commands from the Octave console (e.g. re-type every line followed by ENTER):

pkg rebuild -auto

```
pkg rebuild -noauto ad
pkg rebuild -noauto nan % shadows many statistics functions
pkg rebuild -noauto gsl % shadows some core functions
pkg rebuild -auto java
```




*The Octave command window.*

11. Close and **restart Octave**.

## Installation for Scripting: Matlab

## What is Matlab?

Matlab is a high-level language, primarily intended for numerical computations. The package is commercially distributed by the Mathworks. If you don't have the resources to purchase Matlab, you can run use Octave, which offers similar functionality.

## Versions

Matlab R2011, R2012, R2013, R2014, R2015, R2016 and R2017 have been tested with 20-sim scripting but older and newer versions may also work fine.

## Installation

See the Matlab documentation from the Mathworks for information on installing Matlab. No special (additional) installation is needed to use 20-sim scripting from Matlab.

**Note**: Scripting in Matlab is similar to Octave. You can type exactly the same commands as given for Octave in the next sections.

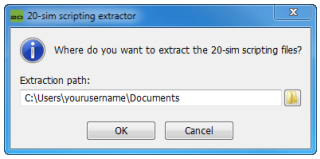
## Prepare Scripting Folder

20-sim comes with a *Scripting Folder* that contains documentation of all scripting functions, the function library and example scripts. You have to install this folder to use scripting.

## Installation

1. Open the **Install Scripting** program from the Windows *Start* menu (located under ***20-sim 4.7***)
   *- or -*
   Go to the folder where 20-sim is installed (e.g. ***C:\Program Files\20-sim 4.7\Scripting*** or ***C:\Program Files (x86)\20-sim 4.7\Scripting***) and open **20simScripting.exe**

2. This will open a dialog where you can choose where to extract the 20-sim scripting files. **Change** the path to a local working folder of your choice (for example: ***C:\Users\yourusername\Documents\20simscripting***)




*20-sim Scripts extraction*

Note: To write/modify scripts, the scripting folder should be accessible and writable by the user. Do not install the scripting folder on *C:\Program Files (x86)* or *C:\Program Files.*

For the remainder of this chapter, we use the name *scripting working folder* when we refer to the folder where you just extracted the 20-sim scripting files.

## Contents

Your newly created *scripting working folder* contains a number of subfolders:

1. **Models:** This folder contains the 20-sim models and data files that are used for the example and tutorial scripts .

2. **Octave:** This folder contains all Octave/Matlab scripting functionality and documentation

a. **documentation**: This folder contains the scripting API documentation: a list of supported functions and their syntax. It is a copy of the help file that you can open in the 20-sim **Editor** by selecting **Help** - **Octave Scripting API**.

   Note: the API documentation is also accessible from the Windows *Start* menu under *20-sim 4.7\Scripting API documentation*

b. **library**: This folder contains the core scripting functions.

c. **tutorials**: This folder contains basis scripts with a step by step explanation. You can use these scripts as a base for your own scripts.

d. **examples**: This folder contains some more advanced scripts.

3. **Octave-patch:** This folder contains modified Octave scripts for certain Octave versions to fix bugs in the core Octave scripts that are not yet fixed in the latest release (currently 4.0.0)

4. **Python:** This folder contains all Python scripting functionality and documentation (see the Scripting in Python section for more information).

## Basic Script

When the scripting files are properly installed in your *scripting working folder*, we can run some tutorial scripts. Tutorial scripts are a step by step demonstrations of usage of the scripting functionality in 20-sim. These scripts are found in the *tutorials* subfolder of the *scripting working folder*. We will start with a basic script that opens and runs a 20-sim model.

1. Open **20-sim**.

2. Open **Octave** (or **Matlab**).

3. In Octave/Matlab, change the local working directory to the tutorial folder inside your *scripting working folder* . E.g . type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\Octave\tutorials'
```

**Note:** 20-sim should be open before running the script!

4. In Octave/Matlab, execute the following command (e.g. type the following **case sensitive** command followed by ENTER):

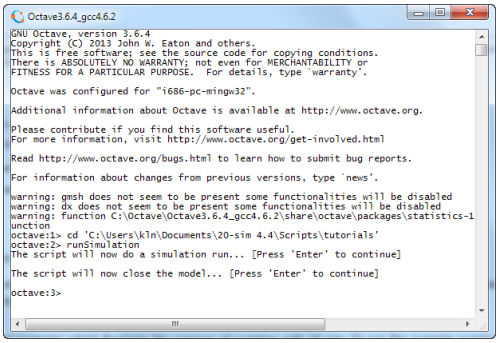
```
runSimulation
```

Note: Octave may give a cryptic "*undefined near line x column 1*" message, if you type the command as `runsimulation` instead of `runSimulation`!

5. Now Octave / Matlab will give a message and ask you to press **ENTER** to continue.

The model *ControlledSystem.emx* is loaded into 20-sim and simulated.

6. Again Octave/Matlab will give a message and ask you to press **ENTER** to continue.




Now the simulation and model will be unloaded.

## Inspecting the script

To see how the script is made, you can inspect it with a text editor.

1. Open a file browser and go to the tutorials folder (e.g. C:\Users\yourusername \Documents\20simscripting\Octave\tutorials)
2. Open the file *runSimulation.m* with a text editor like **Notepad**.

## Core Functions

The core functions of the *runSimulation* script are:

- **addpath**: The script starts with the command addpath('../library/xxsim'); This will enable Octave / Matlab to use the 20-sim scripting functions that are stored in the library subfolder of your scripting working folder.
- **xxsimConnect**: This command opens a connection to 20-sim.
- **xxSimOpenModel**: This command opens a model in 20-sim by giving the filename including the full path.
- **xxsimProcessModel**: This command will process the model.
- **xxsimRun**: This command will run a simulation.
- **xxsimCloseModel**: This command will remove the simulation model from 20-sim.

These functions are the basis of scripting in 20-sim and will be present in this order in most scripts. Therefore you can use the script *runSimulation.m* as a template for any new script that you create.

## Advanced Scripts

Now that we have seen the core functions of a script we will run and check some more advanced scripts.

1. Open **20-sim**.
2. Open **Octave** (or **Matlab**).
3. In Octave/Matlab, change the local working directory to the tutorial folder inside your scripting working folder. E.g . type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\Octave\tutorials'
```

## Set Parameter Values

4. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
SetParameterAndRun
```

This script will open the model *ControlledSystem.emx* and run a simulation. Then a model parameter is changed and a second simulation run is performed. As explained in the previous topic, you can inspect the script in a text editor.

Compared to the basic script you will find a new function:

- **xxsimSetParameters**: This function is used to set the parameter in the model with the new value.

## Multiple Runs

5. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
multipleRun
```

This script will open the model *ControlledSystem.emx* and run a simulation multiple times while changing a parameter. Then a model parameter is changed and a second simulation run is performed.

## Read Parameter Values

6. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
readAndSetParameters
```

This script will open the model *ControlledSystem.emx* and run a simulation. Then a model parameter is read from file and changed accordingly in the model, followed by a second simulation run.

You will find these new functions:

- **addpath**: An additional path is given (../library/xxlib) to allow addtional (user defined) functions.

- **xxlibReadCsv**: This function is used to read a parameter name and value from a spreadsheet file.

## Store Simulation Results

7. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
modelVerification
```

This script will open the model *ControlledSystem.emx* and run a simulation. After the run the simulation results are stored and plotted in Octave/Matlab. You will find these new functions:

- **xxsimSetLogVariables**: Define which variables are going to be logged during the simulation run.
- **xxsimGetLogVariables**: Export the logged variables after the simulation run to Octave/Matlab.

## Examples

8. In Octave/Matlab, change the local working directory to the tutorial folder. E.g. type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\examples'
```

here you can find more example scripts.

## Writing your own Scripts

## Example

We will show you how to write your own scripts using a simple example. We assume that you have installed a scripting folder and its location is:

```
'C:\Users\yourusername\Documents\20simscripting'
```

of course you can use own location. We will copy a 20-sim model to the scripting folder and write a script that will open this model in 20-sim and run a simulation.

1. **Copy** the example model **FastManipulator.emx** to the Octave\tutorials folder. E.g copy:

```
'C:\Program Files (x86)\20-sim 4.7\Models\Examples\Drivetrains
\FastManipulator.emx'
```

to

```
'C:\Users\yourusername\Documents\20simscripting\Octave\tutorials
\FastManipulator.emx'
```

2. Open a text editor (e.g. notepad) and enter the following lines:

```
run('../library/xxsim/xxsimAddToPath.m');
xxsimConnect();
xxsimOpenModel( 'FastManipulator.emx' );
xxsimProcessModel();
xxsimRun();
xxsimDisconnect();
```

3. **Save** the text file as:

```
'C:\Users\yourusername\Documents\20simscripting\Octave\tutorials
\MyScript.m'
```

4. Open **20-sim**.
5. Open **Octave** (or **Matlab**).
6. In Octave/Matlab, change the local working directory. Type in the command line:

```
cd 'C:\Users\yourusername\Documents\20simscripting\Octave\tutorials'
```

7. In Octave/Matlab, run your own script. Type in the command line:

```
MyScript
```

Now you will see the model being loaded in 20-sim and a simulation being run.

## Writing your own scripts

In the tutorial folder there are more scripts. Use these as a template for writing you own scripts and follow the guidelines below:

### Location

Create your own subfolder inside your scripting working folder. This allows you to update the 20-sim scripting files when new versions of 20-sim are released.

### Functions

You can find help on scripting functions in the 20-sim **Editor** by selecting **Help** - **Octave Scripting API**.

## 16.4 Scripting in Python

### Installation for Scripting: Python

### What is Python

Python is a general-purpose high-level programming language with an emphasis on code readability and writing algorithms in fewer lines of code than other programming languages. Python is open-source and managed by the Python Software Foundation. It has an extensive standard library and can be extended with many external libraries including a rapidly growing set of scientific and mathematical libraries such as SciPy, NumPy and Sympy and an extensive plotting library Matplotlib. 20-sim scripting has been tested with the following versions of Python: Python 2.7.x, Python 3.4.x and Python 3.5.x (32-bit and 64-bit).

### Installation

During installation of 20-sim, you are asked to install the (optional) **Python 3.4 package**. We advise to keep the default setting (**Yes**) which will install the Python 3.4 installation that includes 20-sim scripting support and the following packages: NumPy, Matplotlib and IPython. This installation provides just enough support to get started with 20-sim scripting. However, it does not provide a development IDE or an extensive set of scientific and mathematical libraries.

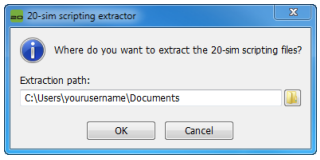
### Prepare Scripting Folder

20-sim comes with a *Scripting Folder* that contains documentation of all scripting functions, the function library and example scripts. You have to install this folder to use scripting.

### Installation

1. Open the **Install Scripting** program from the Windows *Start* menu (located under ***20-sim 4.7***)
   *- or -*
   Go to the folder where 20-sim is installed (e.g. ***C:\Program Files (x86)\20-sim 4.7\Scripting*** or ***C:\Program Files (x86)\20-sim 4.7\Scripting***) and open **20simScripting.exe**

2. This will open a dialog where you can choose where to extract the 20-sim scripting files. **Change** the path to a local working folder of your choice (for example: ***C:\Users\yourusername\Documents\20simscripting***)




*20-sim Scripts extraction*

Note: To write/modify scripts, the scripting folder should be accessible and writable by the user. Do not install the scripting folder on *C:\Program Files (x86)* or *C:\Program Files.*

For the remainder of this chapter, we use the name *scripting working folder* when we refer to the folder where you just extracted the 20-sim scripting files.

## Contents

Your newly created *scripting working folder* contains a number of subfolders:

1. **Models:** This folder contains the 20-sim models and data files that are used for the example and tutorial scripts .
2. **Octave and Octave-patch:** These folders contain Octave/Matlab scripting functionality and documentation (see the Scripting in Octave/Matlab section for more information)
3. **Python:** This folder contains all Python scripting functionality and documentation:
    a. **controllab:** Folder containing the Python classes that allow communication with 20-sim.
    b. **documentation**: This folder contains the scripting API documentation: a list of supported functions and their syntax. It is a copy of the help file that you can open in the 20-sim **Editor** by selecting **Help** - **Python Scripting API**.
    c. **examples**: This folder contains some more advanced scripts.
    d. **tutorials**: This folder contains basis scripts with a step by step explanation. You can use these scripts as a base for your own scripts.

## Basic Script

When the scripting files are properly installed in your *scripting working folder*, we can run some tutorial scripts. Tutorial scripts are step by step demonstrations of usage of the scripting functionality in 20-sim. These scripts can be found in the *tutorials* subfolder of the *scripting working folder*. We will start with a basic script that opens and runs a 20-sim model.

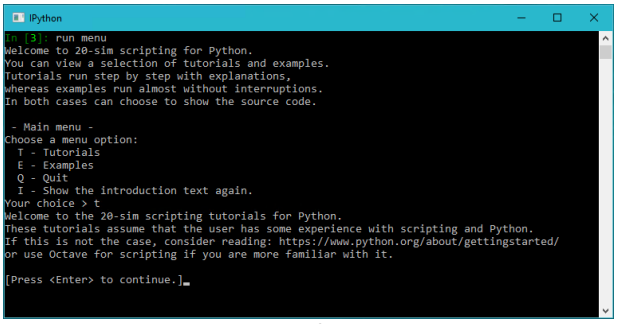
1. Open **20-sim**.
2. Open **IPython** (Interactive Python shell) from the Start menu (under 20-sim 4.7).
3. In IPython, **change** the local **working directory** to the tutorial folder inside your *scripting working folder* . E.g . type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\Python'
```

4. In IPython, execute the following **command** (e.g. type the following **case sensitive** command followed by ENTER):

```
run main_menu
```

Note that the `run` command is specific for IPython. For a standard Python session, you can start this script on the command line using: `python.exe menu.py`. This command will show a menu with several options including **T** for Tutorials.




*IPython session for the tutorials*

5. Select option **T** - Tutorials (**press t, ENTER**) to show the tutorial menu:

```
 - Tutorial menu -
Select a tutorial:
1 - Run a simulation.
2 - Set a parameter in 20-sim, then run a simulation.
3 - Execute multiple runs with a changing parameter.
4 - Basic simulation result analysis.
5 - Read a parameter from a CSV file and set it in 20-sim.
6 - Retrieve 20-sim model variables and their properties.

Or choose a menu option:
  Q - Quit
  I - Show the introduction text again.
Your choice > 1
```

6. Press **ENTER** again to show the available tutorials and choose option **1** *Run a simulation* followed by **ENTER**.

```
In this tutorial the scripting interface will:
 - Open a 20-sim model (starting 20-sim if necessary)
 - Process and run the model
 - Close the 20-sim model
```

7. Press **ENTER**

```
The Python scripting interface will now connect to 20-sim.
If 20-sim is not running it will be started automatically.
```

8. Press **ENTER**

```
Connecting, please wait...

The scripting interface has successfully connected to 20-sim.
The tutorial model will be opened.
If you still have an open model. SAVE YOUR MODEL, unsaved changes
will be overwritten.
```

9. Press **ENTER**

```
The model ControlledSystem.emx has been opened in 20-sim.
The model will be processed and simulated.
The 20-sim plot window will open.
```

10. Press **ENTER** to load the model *ControlledSystem.emx* in 20-sim and to simulate it.

```
The tutorial will now close the 20-sim model and exit.
```

11. Press **ENTER** to close the simulation and this 20-sim model

## Inspecting the script

```
Tutorial completed!
Do you want to see the source code? [y/N]
```

To see how the script is made, you can inspect it by choosing *y.* This will print the relevant script lines on the Python console. You can also open the real script in a text editor like **Notepad** by opening the file: `C:\Users\`*`yourusername`*`\Documents\20simscripting\Python\tutorials\run_simulation.py`.

## Important Functions

The important functions / lines of the *runSimulation* script are:

- **import controllab:** Tell Python to load the Controllab package with the 20-sim scripting functions in the XXSim() class.
- **my20sim = controllab.XXSim():** create a 20-sim scripting object
- **my20sim.connect()**: This command opens a connection to 20-sim.
- **my20sim.set_scriptmode():** Tell 20-sim that we are in scripting mode (does not show confirmation dialogs)
- **my20sim.open_model()**: This command opens a model in 20-sim by giving the file name including the full path.
- **my20sim.process_model()**: This command will process the model.
- **my20sim.run()**: This command will run a simulation.
- **my20sim.close_model()**: This command will remove the simulation model from 20-sim.

These functions are the basis of scripting in 20-sim and will be present in this order in most scripts.

## Writing your own Scripts

### Example

We will show you how to write your own scripts using a simple example. We assume that you have installed a scripting folder and its location is:

```
'C:\Users\yourusername\Documents\20simscripting'
```

Of course you can use own location. We will copy a 20-sim model to the scripting folder and write a script that will open this model in 20-sim and run a simulation.

1. **Copy** the example model **FastManipulator.emx** to the `Python\tutorials` folder. E.g copy:

```
'C:\Program Files (x86)\20-sim 4.7\Models\Examples\Drivetrains
\FastManipulator.emx'
```

to

```
'C:\Users\yourusername\Documents\20simscripting\Python\tutorials
\FastManipulator.emx'
```

2. Open a text editor (e.g. notepad) and enter the following lines:

```
import controllab
xxsim = controllab.XXSim()
xxsim.connect()
xxsim.open_model('FastManipulator.emx')
xxsim.process_model()
xxsim.run()
xxsim.disconnect()
```

3. **Save** the text file as:

```
'C:\Users\yourusername\Documents\20simscripting\Python\tutorials
\myscript.py'
```

4. Open **20-sim**.
5. Open **IPython**.
6. In IPython, change the local working directory. Type in the command line:

```
cd 'C:\Users\yourusername\Documents\20simscripting\Python\tutorials'
```

7. In IPython, run your own script. Type in the command line:

```
run myscript
```

Now you will see the model being loaded in 20-sim and a simulation being run.

## Writing your own scripts

In the tutorial folder there are more scripts. Use these as a template for writing your own scripts and follow the guidelines below:

### Location

Create your own subfolder inside your scripting working folder. This allows you to update the 20-sim scripting files when new versions of 20-sim are released.

### Functions

You can find help on scripting functions in the 20-sim **Editor** by selecting **Help** - **Python Scripting API**.

## Advanced Functionality

## Python Distributions

When you need more functionality or prefer to use an IDE with syntax highlighting and debugging support, it is strongly advised to install one of the following external Python distributions or IDEs:

- PyZo: a free and open-source Python distribution that comes with many scientific packages and the powerful IEP IDE.
- Spyder: the Scientific PYthon Development EnviRonment with a powerful IDE for the Python language with advanced editing, interactive testing, debugging and introspection features and a a numerical computing environment based on SciPy, NumPy, Matplotlib and IPython.
- Python, extended with the Visual Studio IDE and Python Tools for Visual Studio.

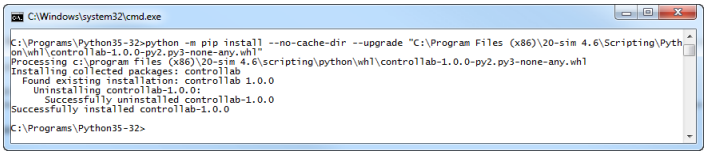
## Running 20-sim scripts in Python distributions

To add the 20-sim scripting support to your Python distribution, you can use the Python pip command (installed by default since Python 2.7.10 and 3.4.x) to install the Controllab package.

1. Open a Windows command prompt (**cmd.exe**) and **type**:

```
cd YOUR_PYTHON_INSTALLATION_DIR\
python -m pip install --no-cache-dir --upgrade "C:\Program Files (x86)
\20-sim 4.7\Scripting\Python\whl\controllab-1.0.0-py2.py3-none-any.whl"
```

*Note:* use C:\Program Files\20-sim 4.7\ on 32-bit Windows systems.



*Manual installation of the Controllab package in Python*

# Index

20-sim is a modeling and simulation program that runs under Microsoft Windows. With 20-sim you can simulate the behavior of dynamic systems, such as electrical, mechanical and hydraulic systems or any combination of these.

20-sim fully supports graphical modeling, allowing to design and analyze dynamic systems in a intuitive and user friendly way, without compromising power. 20-sim supports the use of components. This allows you to enter models as in an engineering sketch: by choosing components from the library and connecting them, your engineering scheme is actually rebuilt, without entering a single line of math!


**Controllab Products B.V.** Hengelosestraat 500, 7521 AN Enschede, The Netherlands
T +31(0)85 773 18 72, E info@controllab.nl, **www.20sim.com**