

COSIN/mbs

Multi-Body Systems & Vehicle Dynamics Simulation Documentation and User's Guide

1	General Remarks	3
1.1	Aims and Scope of <i>COSIN/mbs</i>	3
1.2	Modeling Approach	3
1.3	Data File Formats and Notation Used in Data Tables Below	4
2	Simulation Workbench	5
2.1	Selection of Simulation and Model Data Files	6
2.2	Simulation Data	6
2.2.1	Simulation Data for <i>COSIN/mbs</i>	7
2.2.2	External Input Signals (Sources)	11
2.2.3	Road Profiles, Obstacles, and Wind Velocity	12
2.3	Model Data	12
2.4	Program Execution and Analysis of Results	15
3	Modeling and Model Data Files	17
3.1	Model Definition	18
3.1.1	Group Definition	19
3.1.2	Element Definition	21
3.2	Element Catalogue	23
4	Interfacing to Models for Road and Wind Velocity	24
4.1	<i>COSIN/road</i>	24
4.2	<i>COSIN/wind</i>	24
4.2.1	Type <i>calm</i>	24
4.2.2	Type <i>constant</i>	24
4.2.3	Type <i>file_v_of_t</i>	25
4.2.4	Type <i>file_v_of_x</i>	25
4.2.5	Type <i>table_v_of_t_and_x</i>	26
4.2.6	Type <i>table_v_of_x_and_y</i>	26
4.2.7	Type <i>function</i>	26
5	ECU Interfacing	28
5.1	Standard ABS	29
5.2	Advanced ABS	30
5.3	ESP (Electronic Stability Program)	30
5.4	Automatic Four-Wheel Drive Select	31
5.5	ATTS (Automatic Torque Transfer System)	31
5.6	LSD (Limited Slip Differential)	32
5.7	Active LSD	33
5.8	CLD (Controlled Partially Locking Differentials)	33
5.9	Four-Wheel Steering	34
5.10	EPS (Electronic Power Steering)	34
5.11	Advanced EPS	35
5.12	General Force Actuator	35
6	Interfacing to User-Supplied Driver-Models	37

7	Command Line Invocation	39
8	Coupling to Matlab/Simulink™	41

1 General Remarks

This documentation describes the multi-body and vehicle dynamics simulation package *COSIN/mbs*. All product or brand names mentioned here are trademarks or registered trademarks of their respective holders.

1.1 Aims and Scope of *COSIN/mbs*

COSIN/mbs is designed to be a fast and easy-to-use simulation tool for general multi-body mechanisms, especially suspensions and vehicle models for both ride comfort and vehicle handling maneuvers. It is equipped with a general road and tire interface, allowing the combination with several existing tire models like *FTire*, *FETire*, *RTire*, *TMeasy*, *Magic Formula* in several variants, and others. These tire models can be made available upon request.

By using the comfortable data file format *COSIN/io*, *COSIN/mbs* can be easily parameterized, and combined with other computation software like Excel™ or Matlab™ to provide these parameters.

COSIN/mbs comes with several sample data files, defining the most important suspension types both for front and rear axles. If you are licensed to, these will include

- five-link suspensions,
- double wishbone suspensions,
- several types of McPherson suspensions,
- race-car suspensions,
- customer-specific axles,
- a ‘generic suspension model’ using a general nonlinear force element, situated between wheel and car-body, and
- the ‘conceptual suspension’ approach, importing files (‘.scf-files’) that describe the full elasto-kinematic properties of a given suspension, from other vehicle dynamics simulation software.

Likewise, *COSIN/mbs* also enables the user to define **completely new suspension types** in terms of a list of *COSIN/mbs* elements.

COSIN/mbs is available for several Unix and Linux dialects, as well as for Windows NT™, 2000™, XP™, Vista™, and 7™ (32 and 64 bit). Even with a detailed rendered 3D on-line animation, it runs considerably faster than real-time.

1.2 Modeling Approach

COSIN/mbs is a specialized multi-body system software, especially focussing on those multi-body elements needed in vehicle suspension models, including

- flexible bodies,
- specialized ‘elasto-kinematic’ elements,
- steering assembly models,
- drive-train subsystems of different complexity,
- brake system elements,
- aerodynamics, and
- various road and tire model interfaces.

COSIN/mbs uses a tailored implicit integration routine, where only stiff subsystems are integrated implicitly. Even though implemented in lean and efficient, ‘number-crunching’ ANSI Fortran 90, it consequently observes an object-oriented approach.

COSIN/mbs provides a mechanism to get all **pre-load values** of all bearings, springs, and so on, in refer-

ence position. To this end, *COSIN/mbs* puts out all pre-load values at the end of a simulation into an output file, and appends the original input data file. By this, a new, compatible input file is created that carries not only all original vehicle data, but also all pre-load values to be used in subsequent simulations.

This documentation comprises the following chapters:

- [chapter 2](#) introduces and documents the working with the *COSIN/mbs* simulation environment,
- [chapter 3](#) gives a detailed documentation of *COSIN/mbs* models and elements,
- [chapter 4](#) describes the program interfaces to user-provided ECU software,
- [chapter 5](#) describes the program interfaces to user-provided ECU software,
- [chapter 6](#) documents the command-line invocation of *COSIN/mbs*,
- and [chapter 7](#) gives details about using *COSIN/mbs* within Matlab/Simulink™ models.

1.3 Data File Formats and Notation Used in Data Tables Below

All data files of *COSIN/mbs* (apart from imported data files like .csf-files, .tdx-files, and .fem-files) use the *COSIN/io* syntax that is described in the separate documentation [COSIN/io User's Guide](#). Additional relevant documentation sheets are *COSIN/ip User's Guide* (description of several plotting utilities, not yet available), [COSIN/road User's Guide](#) (description of road models), and [FTire User's Guide](#) (documentation of the *COSIN/mbs* compatible tire model *FTire*).

In the following chapters, several tables describing input data are included. Most of these tables comprise 4 columns: the **data names**, the **physical units**, the **types**, and a brief description of their **meaning**. Thereby, the following data types are distinguished:

i	a single integer variable
in	a vector of integer variables with <i>n</i> components
f	a single floating point variable
fn	a vector of floating point variables with <i>n</i> components
sig	the name of a <i>COSIN/ss</i> signal, to be used as input or output signal of an element
st	a text string of arbitrary length Remark: a text string must not contain any blank spaces between the first and the last non-blank character
stn	a text string of exact length <i>n</i>
m	a marker. Markers either can be defined directly by three numbers (that is, by f3 type data), or by referring to an entry in the \$markers data-block of the .cm-file, specifying the name of the respective marker in this block (cf. chapter 3)
sd	a spline data-block name. Spline data to be used are to be specified in a spline data-block with this name. All splines used in <i>COSIN/mbs</i> are 2D-splines with natural boundary conditions. That means, they approximate functions $f(x)$ where $f''(x) = 0$ for the smallest and largest <i>x</i> -value that appears in the data table
mo	a subsystem or element data-block name. This data type specifies the data-block name of a subsystem or an element of appropriate type
t2	a 2D look-up table
fi	a file name, observing the respective operating system's naming conventions
flag	boolean value only, which is true, if the data name appears in the data-block

Mandatory data are marked with (m). If not stated otherwise, all numerical data that are neither mandatory nor explicitly specified get the default value(s) 0.

2 Simulation Workbench

A Tcl/Tk-based graphical user interface (GUI) is part of the *COSIN/mbs* package. This ‘simulation workbench’ allows a convenient operation of *COSIN/mbs* and other related *COSIN* tools. Appearance and functionality of the GUI are nearly independent on the operating system. The workbench, as it looks like in all Windows™-type operating systems, is shown in [figure 1](#).

Besides that GUI, *COSIN/mbs* can also be launched by a simple command line invocation.

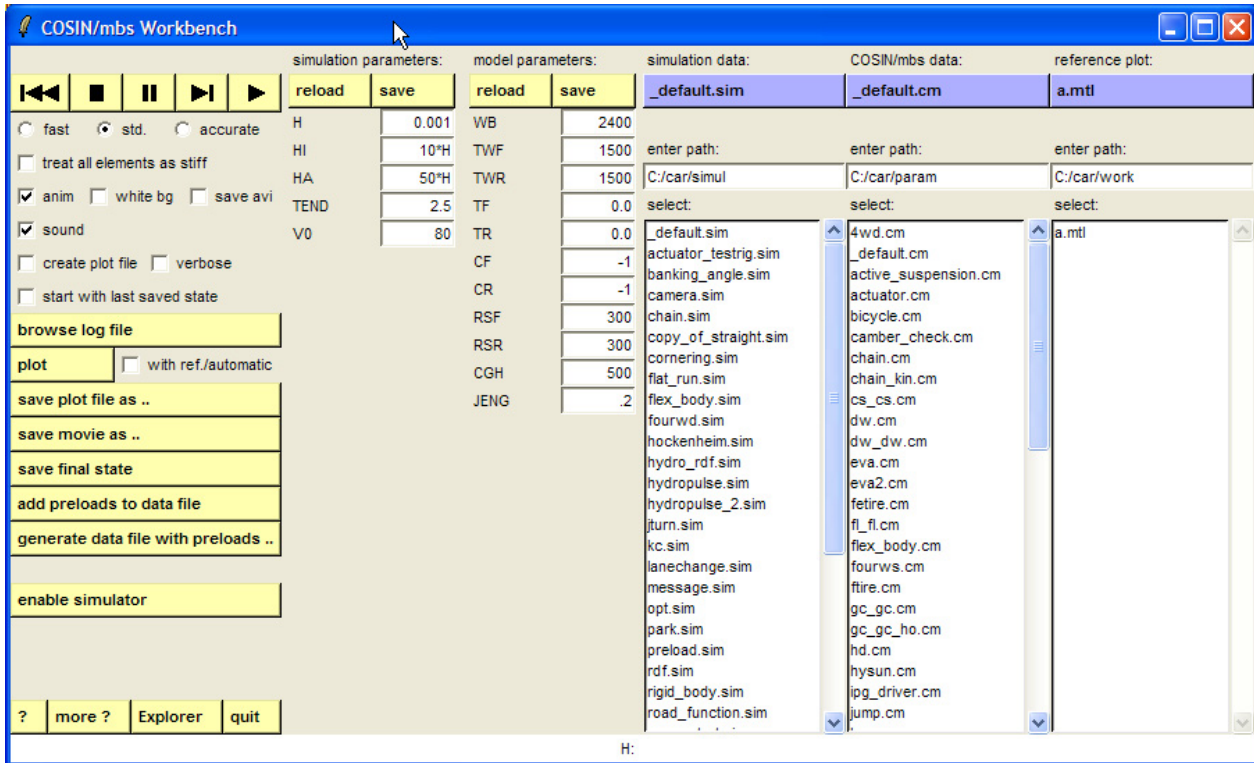


Figure 1: *COSIN/mbs* workbench

The workbench is divided into 5 or 6 main columns:

- the leftmost column contains several **yellow control buttons** for starting/stopping/resuming simulation runs, plotting the results, etc., as well as some **check-boxes** and ‘**radio buttons**’ that influence program execution;
- the second column lists the **parameters** that are found in the active **simulation** and **model data** files; all parameters can be changed by entering a new value or expression in the respective input field, without editing the simulation file;
- the third and fourth columns show the available simulation and model data files in a respective **file-box** (simulation and model data files are read by the simulation program after starting the simulation);
- depending on the kind of installation, an optional fifth column shows the available input files for the 3rd-party model **IPG-Driver**, which describes human vehicle control activities;
- the file-box of the last column lists output files of actual and previous simulation runs.

The upper **blue buttons** starting with column 3 are called **function buttons**. Depending on the mouse button used when clicking, they allow for different ways to process the displayed (‘active’) file. Below, these actions are described in detail.

A typical simulation run is performed as follows:

- select simulation file and model data file from the respective file-box,

- edit and adapt simulation and model data,
- click appropriate program execution control buttons,
- launch and control program execution, using the GUI's 'VCR' buttons and/or functions provided by an interactive animation window that appears when simulation starts;
- analyze results, for example by using the interactive plot software *COSIN/ip*.

In the following, these steps are described in more detail.

2.1 Selection of Simulation and Model Data Files

When *COSIN/mbs* is called for the first time, the file-boxes in columns 3 and 4 of the workbench show several sample files (cf. [figure 1](#)). The number and type of these sample files depend on the *COSIN/mbs* options available in your installation. A specific file is **selected** by left-clicking the file-name. By doing so, the name of the file will appear in the blue function button. If required, the file may then be **edited** by right-clicking the blue function button.

By default, the simulation and model data files are searched for in the directories `simul` and `param`, respectively. These folders themselves are located in the folder `car`, a sub-folder of that directory where you installed your COSIN products. If you keep your data or simulation files in different folders, specify these folders in the `path` entry of column 3 or 4, respectively. You can use every kind of path definition that is valid in the operating system you are using, for example

- `simul\steady_state`
- `..\temp\data`
- `c:\own_files\cosin`

in Windows™-type operating systems, and

- `simul/steady_state`
- `../temp/data`
- `$home/own_files/cosin`

in Unix-type operating systems.

The sample files shipped with *COSIN/mbs* may serve as a reference while creating new user specified files. In order to leave the original files unchanged, it is recommended to generate a copy of the reference file first, which covers the type of maneuver or vehicle needed. For example, to create a vehicle model with a double wishbone front and rear suspension for *COSIN/mbs*, choose file `dw_dw.cm`. The procedure to **copy** that file is executed in two steps: first, **left-click** the file to be copied in the list-box, then hit the **space-bar**. Now, a file `test_dw_dw.cm` will appear in the list box. The same procedure holds for creating a user defined simulation file.

Files are **deleted** by left-clicking the file name in the list-box, then hitting the '**Del**' key.

2.2 Simulation Data

To adapt the simulation data like simulation time span, number of output steps, kind of on-line animation, manoeuvre to be driven, road profile, etc., to the problem at hand, the corresponding simulation data file (sim-file for short) is to be edited. Typically, that file consists of the data-blocks

- `$simulation,`
- `$sources,`
- `$road_type.`

Above that, as in all COSIN data files, an optional data-block

- `$parameters`

may be used to parameterize the simulation file, cf. *COSIN/io* documentation. All parameters in this block are automatically displayed in the second column of the user interface, for convenient access.

2.2.1 Simulation Data for *COSIN/mb*s

If *COSIN/mb*s or a user-modified solver is used, the data-block `$simulation` is scanned for the following data:

<code>simulation</code>	s	f3 (m)	simulation starting time, time step, final time
<code>plot_output</code>	s	f3	starting time, time step and final time of plot output. If not specified, plot output is suppressed
<code>animation</code>	s	f3	starting time, time step and final time of animation output. If not specified, animation output is suppressed
<code>initial_position</code>	m	f3	initial position of all bodies' points-of-reference in global coordinates. Default: 3*0.0
<code>initial_angular_pos</code>	deg	f3	initial angular orientation of all bodies. Default: 3*0.0
<code>initial_velocity</code>	rads	f3	initial translational velocities of all bodies' points-of-reference in global coordinates. Default: 3*0.0
<code>initial_ang_velocity</code>	m/s	f3	initial angular velocities of all bodies in body-fixed coordinates. Default: 3*0.0
<code>initial_gear</code>	-	i	initial gear to be chosen in propulsion system and semi-automatic transmission Default: 0 (neutral)
<code>friction</code>	-	i	switch for activating all friction values in <i>COSIN/mb</i> s elements. 0 = no, 1 = yes Default: 1
<code>play</code>	-	i	switch for activating all play values in <i>COSIN/mb</i> s elements. 0 = no, 1 = yes Default: 1

<code>KC_analysis_type</code>	-	i	switch to select between normal simulation mode and certain specialized modes for K&C (kinematics and compliance) analysis: 0: standard mode (default) 1: vertical wheel travel kinematics, friction and play disregarded 2: steering kinematics, friction and play disregarded 3: vertical wheel travel kinematics including friction and play 4: steering kinematics including friction and play 11: wheel compliance by external forces and/or moments, friction and play disregarded 12: wheel compliance by external forces and/or moments, including friction and play
<code>road_file</code>	-	st	file name of road data file, containing data-block <code>\$road_type</code> . Default: simulation data file name (this file)
<code>udm_file</code>	-	st	name of a file containing data for a user-supplied driver-model. The format of this file is defined by the driver model, independently on <i>COSIN/mbs</i> . The user-supplied driver-model will be used to control the vehicle operation if and only if this data file is specified, cf. chapter 5 . Default: not specified
<code>plot_output_start_time</code>	s	f	time shift for plot output. For example, a value of 1 s shifts the time scale, beginning at $t = 1s$.
<code>plotfile_format</code>	-	st	plot-file format. At present, the following formats are available: <ul style="list-style-type: none"> • <code>standard</code> • <code>matlab</code> • <code>excel</code> • <code>gnuplot</code> • <code>matrix</code> Above that, the following proprietary formats are restrictively provided: <ul style="list-style-type: none"> • <code>dspf</code> (dsp file in single precision) • <code>dspd</code> (dsp file in double precision) • <code>dsp</code> (synonym for <code>dspd</code>) • <code>iplos_kdf</code>
<code>animation_zoom</code>	mm	f	animation zoom. If negative, animation is paused at the beginning and must be started by pressing the return or enter key while the animation window is activated. Paused mode may be convenient for choosing an appropriate viewing angle etc. Default: 100

image_shift	-	f2	relative x/y-shift of the focused point in the image plane of the animation window. Default: 0,0
on_line_camera_control	-	i	switch to make camera position and orientation user-controllable during a running simulation. If set to 1, azimuth angle, altitude angle, camera-to-object distance, and viewing angle will be controllable by sliders
camera_position	m	f3	camera position in the body-fixed coordinate system of the body that carries the camera
camera_angles	deg	f3	camera viewing direction angles (with ISO 8855 / DIN 70000 definition) relative to the orientation of the body that carries the camera
time_const_camera_shift	s	f	variable to define 'smoothness' of camera position control with respect to motion of the camera carrier. Camera instantaneously follows carrier, if time constant is less or equal 10^{-10} . Camera doesn't follow carrier at all, if time constant is greater or equal 99
time_const_camera_rot	s	f	variable to define 'smoothness' of camera viewing direction control with respect to orientation of the camera carrier. Viewing direction instantaneously follows carrier, if time constant is less or equal 10^{-10} . Viewing direction doesn't follow carrier at all, if time constant is greater or equal 99
sun_position	m	f3	position of the light source that illuminates the scene
visibility	m	f	visibility. Assumed to be 'infinite', if absolute value is greater than 10000. If the value is negative, <i>COSIN/mb</i> s will use orthogonal projection instead of central perspective as starting value. During a running simulation, the kind of projection can be toggled by hitting the F9 key repeatedly
show_tire_forces	-	i	switch that indicates whether tire forces should be shown at the beginning of the simulation or not. When simulation is running, you can toggle the display of tire forces by pressing the 'f' key while the animation window is active
tire_forces_scaling_factor	mm/ kN	f	scaling factor for tire forces in animation scene: length of arrow per force unit
tire_forces_arrow_sizes	mm	f3	geometrical properties of the tire forces arrows: <ul style="list-style-type: none"> • diameter of arrowshaft, • maximum diameter of arrowhead • length of arrowhead (this properties will only be used if animation_level is greater 1)

<code>show_road_below_tires</code>	-	f	type of road to be displayed below tires: 0 none 1 grid, centered in contact point 2 grid, center fixed 3 rendered, centered in contact point 4 rendered, center fixed
<code>animation_attribute_file</code>	-	st	file name of animation models attribute file Default: <code>cosingl.ini</code>
<code>scopes</code>	-	st,f2	array of up to 10 scope definitions that will appear in the animation window. Each three consecutive entries in the array will describe one scope, by <ul style="list-style-type: none"> • a valid <i>COSIN/ip</i> plot signal name, or a substring of such a signal name, or an arithmetic expression combining several <i>COSIN/ip</i> plot signal names, full or in partial • a minimum axis value, and • a maximum axis value. <p>If a single plot signal name contains blank spaces, it is to be enclosed into single quotes. Signal names in arithmetic expressions must not contain any blank spaces; replace them by an 'at' (@). If only a substring of a signal name is entered, the first signal which contains this substring will be used. If known, you may also refer to the plot signals channel number with a leading hash (#). The last scope definition should be followed by the keyword <code>endscopes</code>.</p> <p>Example:</p> <pre>scopes velocity 0 100 2*\track@curv -10 10 #17 -2 2 'lateral accel' -10 10 endscopes</pre> <p>will generate four scopes (if all plot signals are defined). Velocity will be scaled between 0 and 100, 2-times track curvature between -10 and 10, plot signal number 17 between -2 and 2, and lateral acceleration between -10 and 10. The placement of the scopes in the animation window will be chosen automatically Default: not specified</p>
<code>log_level</code>	-	i	level for output log text during simulation Default: 2
<code>engine_wav_file</code>	-	st	prefix of a set of 3 audio files for engine sound. The full names will be build by appending 1.wav, 2.wav, and 3.wav Default: not specified
<code>first_rpm_engine_sound</code>	f	rpm	engine speed of first engine sound file Default: 1680 rpm
<code>second_rpm_engine_sound</code>	f	rpm	engine speed of second engine sound file Default: 2560 rpm

<code>third_rpm_engine_sound</code>	f	rpm	engine speed of third engine sound file Default: 4160 rpm
<code>car_body_wav_file</code>	-	st	audio file for speed-dependent wind noise of car-body Default: not specified
<code>tire_wav_file</code>	-	st	audio file for slip- and load-dependent tire noise Default: not specified
<code>min_audible_slip_angle</code>	f	deg	minimum side-slip angle that will lead to audible tire noise Default: 3 deg
<code>tire_squeal_max_gain</code>	f	-	factor to reduce maximum tire noise Default: 1 (no reduction)
<code>tire_squeal_ref_wheel_load</code>	f	kN	wheel load at which tire noise at 10 deg slip angle will reach its maximum Default: 10 kN
<code>tire_squeal_pitch</code>	f	-	factor to increase or decrease tire noise frequencies. Admissible range is between 0.5 (noise spectrum shifted by one octave towards lower frequencies) and 2.0 (tire noise spectrum is shifted one octave towards higher frequencies) Default: 1 (no shift)
<code>outall1</code>	-	st	first 2-character element type or element name for which the plot level is to be set to 'all' no matter what is defined in the model file. The string is not case-sensitive Default: not specified
<code>outall2</code>	-	st	second 2-character element type or full element name for which the plot level is to be set to 'all' Default: not specified
<code>outall3</code>	-	st	third 2-character element type or full element name for which the plot level is to be set to 'all' Default: not specified
<code>interpolation_mode_2d</code>	-	i	interpolation mode used with tables of two independent variables 1: bilinear 2: bicubic Default: 2
<code>extrapolation_mode</code>	-	i	extrapolation mode used with tables of one or two independent variables 0: no extrapolation 1: smoothed, nearly constant 2: linear, using left- or rightmost interval slopes
<code>numerical_constants</code>	-	f4	system inherent numerical constants. Please do not alter; only for testing purposes

2.2.2 External Input Signals (Sources)

To define external input signals, the 'Sources&Sinks' functionality of [COSIN/ss](#) is used. With these input signals, a *COSIN/mbs* vehicle model can be controlled similar to a real car. To calculate the input signals, *COSIN/mbs* looks for the data-block `$sources` in the simulation data file. All available external

input signals for that data-block are specified in the element definitions of the model data-block, see [chapter 3](#).

2.2.3 Road Profiles, Obstacles, and Wind Velocity

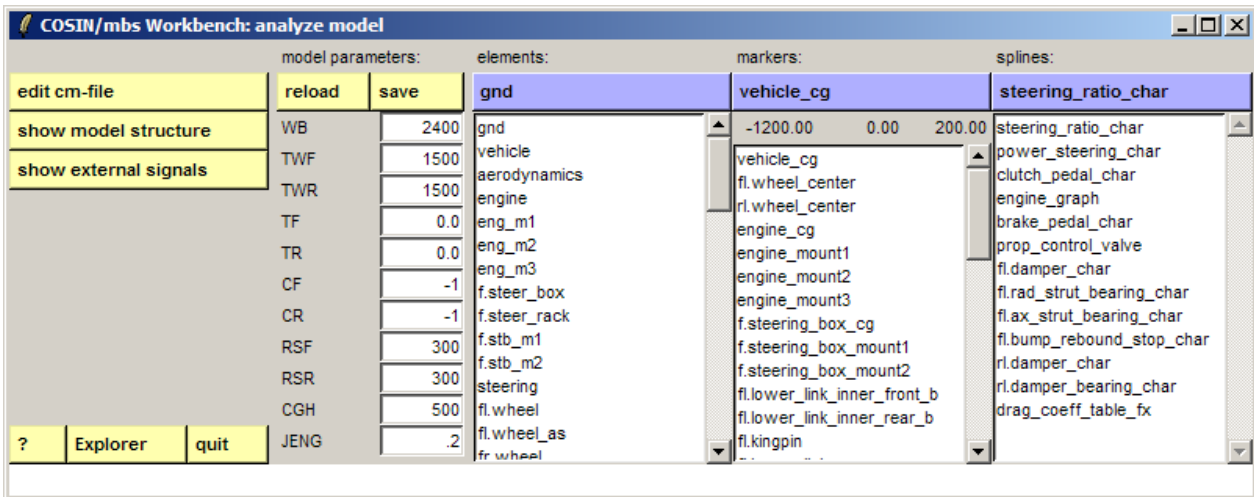
To specify road and wind excitation, *COSIN/mbs* uses the packages [COSIN/road](#) and *COSIN/wind*. These packages read the data-blocks `$road_type` and `$wind` (either in the simulation data file or, if `road_file` or `wind_file`, resp. is assigned a value, in that file). Road profile description with data-block `$road_type` is described in the separate [COSIN/road](#) documentation; whereas wind velocity specification with data-block `$wind` is presented in [chapter 4](#) below.

2.3 Model Data

To fit the model to the problem at hand, edit the corresponding model data file. The meaning and contents of model data files are subject of [chapter 3](#).

Effects of changes in the geometrical model data can be directly observed and validated by clicking the blue function button of column 3 with the **left mouse button**. Then, the program *COSIN/show* is launched, which displays the model in design position. Similar to the on-line animation when a simulation is running, *COSIN/show* can be interactively controlled through zooming in and out, shifting, rotating, and hard-copying. For detailed information, press the F1-button while the graphics window is active. *COSIN/show* is left by hitting the 'Esc' key, or by closing the window.

Most models use 1D and 2D **look-up tables** and **spline data** of different interpolation and extrapolation modes. For validation purposes a **smart browser tool** has been developed, which provides a graphical interactive representation of these data. In order to launch the browser, **right-click** the blue function button in column 3. The browser opens with the graphical user interface shown in [figure 2](#).



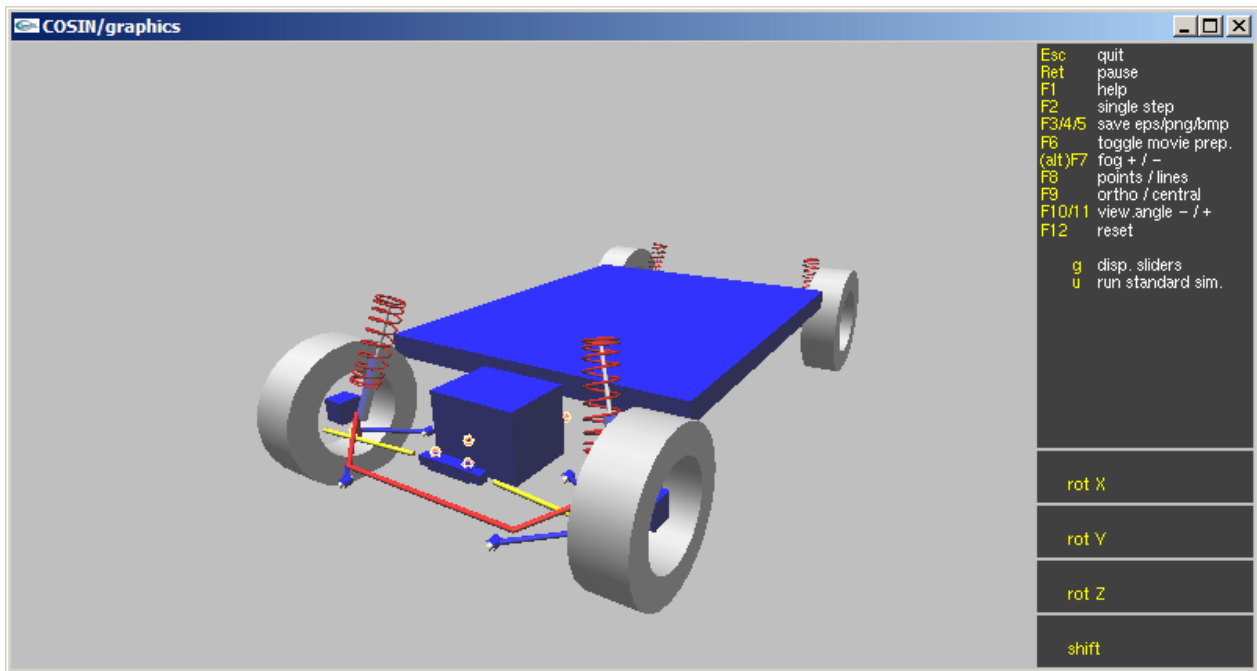
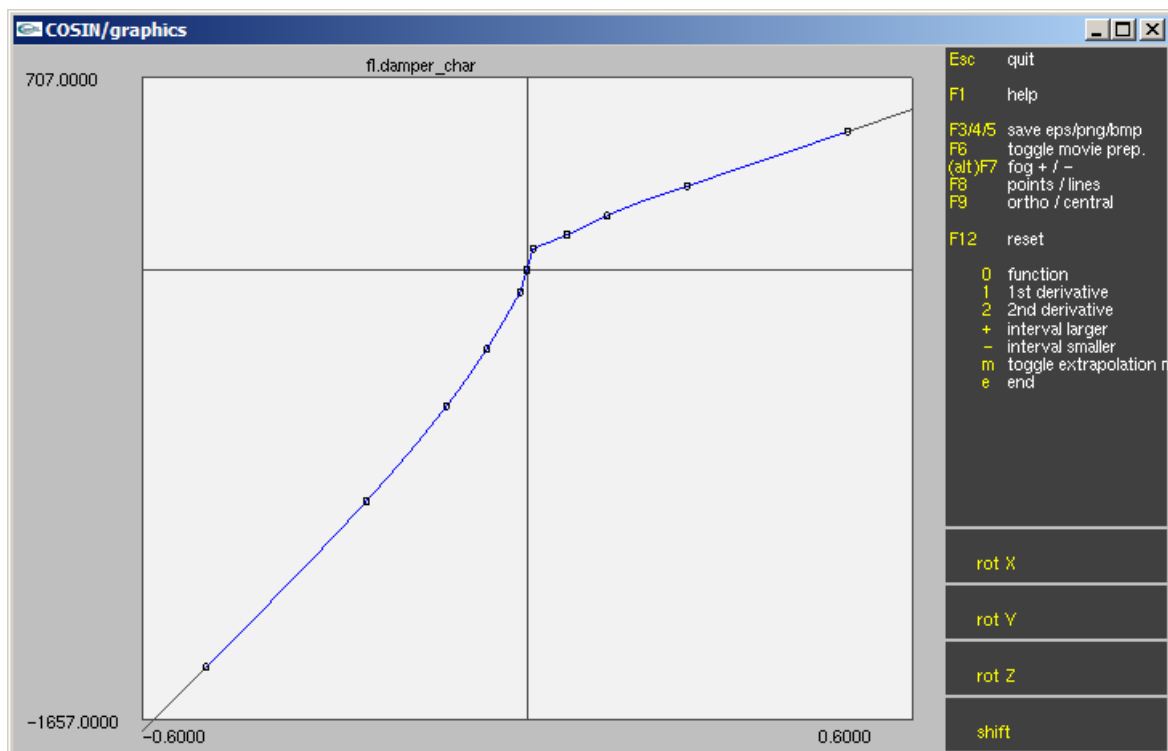


Figure 2: COSIN/show workbench

With launching the browser, the active model data file is scanned for look-up table and spline data-blocks. A list of all corresponding block names is then shown in the left file-box of the browser. The file selection is triggered the same way as for the model and simulation data files. After pressing the function button, the corresponding 1D or 2D function characteristic is shown.



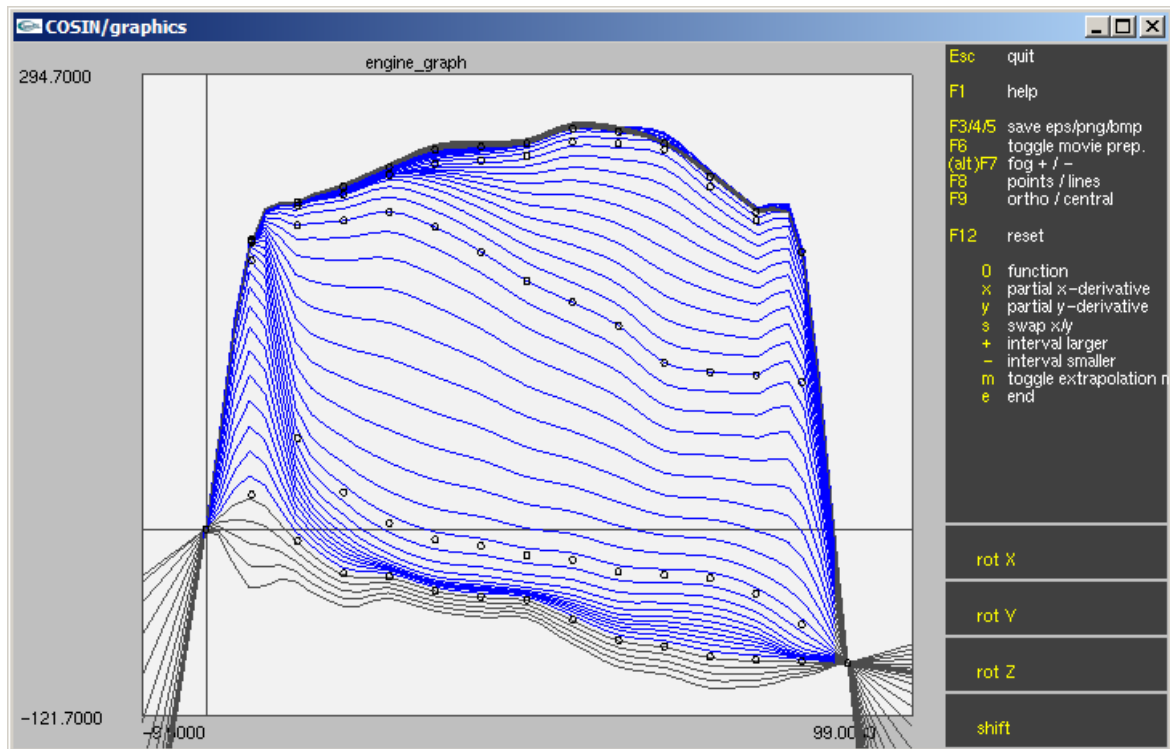


Figure 3: Browser: 1D and 2D spline representation

For 1D data, the browser is interactively controlled by the following function keys:

- 0 show spline,
- 1 show first derivative of spline,
- 2 show second derivative of spline,
- + increase the interval of independent variables,
- decrease the interval of independent variables,
- m toggle extrapolation mode (constant or linear),
- e leave graphics window.

2D data allow for different function keys:

- 0 show spline,
- x show first partial x-derivative of spline,
- y show first partial y-derivative of spline,
- s swap independent variable (x or y),
- + increase the interval of independent variables,
- decrease the interval of independent variables,
- m toggle extrapolation mode (constant, smoothed constant, or linear),
- e leave graphics window.

As for all applications that use *COSIN/graphics*, clicking and holding the middle mouse button displays the mouse coordinates in application units, that is the values of the independent and dependent variables.

Beside look-up table and spline plotting, the browser also displays **amplitude-frequency plots** for the *COSIN/mb*s elements '1D friction' and 'hydro-mount' (see [chapter 3](#)). For that purpose, a frequency sweep ranging from 1 to 30 Hz is performed. The required amplitude of excitation is interactively provided by the user. In order to change the default value of 1.0 mm, the mouse pointer has to be located within the entry. The calculation is started by left-clicking the corresponding function button in column 2 or 3. Please note: the list-box is empty, if the active *COSIN/mb*s model does not contain elements of corresponding type. To leave the graphics window, type 'e'.

It is recommended to always quit the browser after usage by clicking the **yellow quit button**, because the element lists are only refreshed when starting the browser.

2.4 Program Execution and Analysis of Results

The first column of the *COSIN/mbs* workbench shows several yellow control buttons, as well as some radio buttons and check boxes, cf. [figure 4](#), used to control program execution.

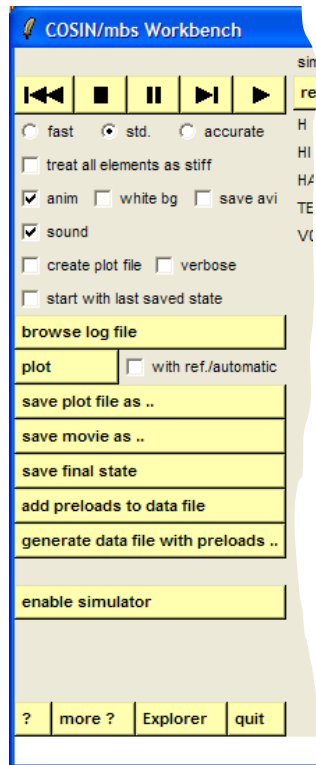







Figure 4: Program execution control buttons

The  button launches a **dynamic analysis simulation**.

Before entering the time loop, the respective solver opens and reads the *COSINmbs* data file, as well as the simulation file that appears in the blue function buttons of column 3 and 4.

Simulation can be:

-  **paused and resumed,**
-  **run in single steps,**
-  **restarted (if not yet finished),**
-  **terminated.**

With the radio buttons group **fast**, **std.**, and **accurate**, the simulation step size defined in data-block `$simulation` can be modified:

- **fast** increases step size by 50%,
- **std.** takes time step exactly as specified in `.sim`-file,
- **accurate** decreases step size by the same amount.

Some further solver parameters can be modified with additional buttons:

- **animate** toggles interactive on-line animation;
- **white bg** will set background color of animation window to 'white' instead of grey;
- **save avi** automatically grabs every single frame of the on-line animation and merges them into an .avi-file. The user will be queried for the name of this file. On-line animation is automatically activated, even if 'animate' is not checked;
- **sound** toggles output of engine sound as well as tire and wind noise (provided an OpenAL-compatible sound card is installed);
- **create plot file** toggles storage of plot-data. Please note: a plot file must have been created before clicking the **plot**-button;
- **start with last saved state** organizes next simulation to starting with the model in the operating condition at the time the preceding simulation was terminated. To make this mechanism work properly, the '**save state**' button has to be clicked after every simulation run, the final states of which are to be taken as initial conditions for the next run.

For analyzing the results, the plotting tool *COSIN/ip* is provided, which interactively displays the output variables of the last simulation run. It is launched via the '**plot**' button. If not only the last output file is to be opened, but also other 'reference' plot-files, check the '**with reference**' box. In that case, a list will appear before starting *COSIN/ip*, where additional plot-files can be entered.

The corresponding plot files can be stored by the user, in order to compare different simulation experiments or model data etc. After left-clicking the button '**save plot file as..**', a file selection window opens, requesting the user to specify a file name. Then, the plot file of the preceding simulation is copied to the specified file. The list box in column 4 shows a list of all previously stored files. In order to open *COSIN/ip* with one of these files directly, left-click the respective file name first, then left-click the blue function button in column 4.

All available PDF-based *COSIN/products* documentation chapters can be browsed by left-clicking '?', and finally *COSIN/mbs* workbench is closed by left-clicking '**quit**'.

3 Modeling and Model Data Files

A *COSIN/mbs* model is described by a *COSIN/mbs* model file (‘.cm-file’). Syntactically, a .cm-file is built according to the syntax rules of [COSIN/io](#) (cf. separate documentation). Normally, it consists of

- a parameter definition block,
- a model definition block, with data sub-blocks that are either
 - group definition blocks or
 - element definition blocks,
- a marker definition block,
- several element data-blocks.

The **parameter block** is an optional means to parameterize all *COSIN/io* input files, and not a special feature of only .cm-files. It is described in the *COSIN/io* documentation.

The **model definition block** `$model` contains a list of all *COSIN/mbs* elements to be included to the model, together with handles of those elements they are linked to. In addition, this data-block carries information specifying where to find element data, what external input and output signals are provided, and so on. Elements that ‘belong together’, like all elements that make up a vehicle’s front suspension, can be **grouped** together. Having done so, they can easily be given common properties like plotting or animation model level, coordinate systems, etc. These groups are also defined in `$model`.

The optional **marker definition block** `$marker` contains geometrical data in terms of marker names, together with their coordinates. These marker names may be used in subsequent *COSIN/mbs* element data-blocks, instead of numerical values. Marker names may be arbitrarily assigned by the user, to customize a *COSIN/mbs* data file. One advantage of the use of markers is the ability to concentrate the complete geometry information of a mechanical model in one single data-block.

Element data-blocks contain all data of the elements included in the `$model` data-block. Normally, to every such element there belongs at most one element data-block. The contents of these data-blocks are described below, in the respective element chapters.

The following is part of a *COSIN/mbs* model file, and illustrates the typical structure of .cm-files:

```

$parameters !*****
WB      2400 ! wheel base [mm]
TWF     1500 ! front track width [mm]
RSF     3000 ! front wheels static tire radius [mm]
...
parameter definition block

$model !*****
GR:vehicle      pl=1; al=5;          dz=RSF
GR:engine       pl=1; al=5;          dz=RSF
GR:steering     pl=1; al=5;          dz=RSF
GR:drive        pl=1; al=5;          dz=RSF
GR:brakes       pl=1; al=5;          dz=RSF
GR:front        pl=1; al=5;          dz=RSF
GR:rear         pl=1; al=5; dx=-WB; dz=RSF
...
group definition sub-blocks

RB:vehicle      pl=2; camera; o1=C1.v; o6=C1.a_lat;
                o8=C1.roll; o9=C1.pitch; o10=C1.yaw;
                o11=C1.roll-vel; o12=C1.pitch-vel;
                o13=C1.yaw-vel; o18=C1.track; o19=C1.curv; g=vehicle
AD:aerodynamics l=vehicle;          g=vehicle

DT:drive_torque l1=FL.tire; l2=FR.tire; l3=RL.tire;
                l4=RR.tire; i1=C1.gas_pedal;          g=drive
BR:brakes       l1=FL.tire; l2=FR.tire; l3=RL.tire;

```

```

l4=RR.tire; i1=C1.brake_pedal; g=brakes

RB:engine      subt=vehicle; g=engine
BE:eng_m1     l1=vehicle; l2=engine; stiff; g=engine
BE:eng_m2     l1=vehicle; l2=engine; stiff; g=engine
BE:eng_m3     l1=vehicle; l2=engine; stiff; g=engine

RB:FL.wheel   subt=vehicle; g=front
RB:FR.wheel   subt=vehicle; mdb=FL.wheel; g=front
RB:RL.wheel   subt=vehicle; g=rear
RB:RR.wheel   subt=vehicle; mdb=RL.wheel; g=rear

...
element definition sub-blocks

```

```

$markers !*****
vehicle_cg      -0.5*WB      0.00  CGH-RSF ! [mm]
FL.wheel_center 0.00  0.5*TWR  0.00 ! [mm]
RL.wheel_center 0.00  0.5*TWR  0.00 ! [mm]

engine_cg      -150.00  0.00  50.00 ! [mm]
engine_mount1  20.00  0.00  50.00 ! [mm]
engine_mount2 -320.00 -200.00 50.00 ! [mm]
engine_mount3 -320.00 200.00 50.00 ! [mm]

...
marker definition block

```

```

$vehicle !RB *****
mass          1200          ! [kg]
inertia_tensor 500  3000  3000  0 0 0 ! [kgm^2]
cg_in_ref_position vehicle_cg ! [mm]
!
animation_model models/car-body.str !
element data-block #1

```

```

$engine !RB *****
mass          120          ! [kg]
inertia_tensor 10  10  10  0 0 0 ! [kgm^2]
cg_in_ref_position engine_cg ! [mm]
!
animation_model brick; width 300 !
height 300; depth 400 !
element data-block #2

```

```

...
more element data-blocks

```

3.1 Model Definition

The model definition data-block `$model` of a *COSIN/mbs* model file consists of second-level, ‘type-equipped’ data-blocks. In contrast to most other *COSIN/io* files, the names of these second-level data-blocks do not start with a double `$`-sign, but with a 2-letter **type acronym**. This type acronym, like `RB` for ‘rigid body’, is separated from the element name by a colon, for example `RB:car_body`. All the rest of such a type-equipped data-block fully observes all syntax rules of *COSIN/io* data-blocks, giving users a maximum degree of flexibility in making a .cm-file better readable.

Every typed second-level data-block in `$model` either defines a new **group** or a **single *COSIN/mbs* element**. There is no need to sort them in any way. This is done automatically by *COSIN/mbs* during pre-processing. For example, it is allowed to refer to other elements inside an element definition, even if these other elements are defined only later.

Besides the second-level data-blocks defining groups and elements, data-block `$model` may contain several default values, which are described in the following table:

<code>vehicle_type</code>	-	i	type of vehicle; used to select ranges in driving simulator displays, etc. Possible values: 0 (passenger car), 1 (truck), 2 (race car)
<code>pl</code>	-	i	default plot level for all groups and elements. The actual plot level defines the number of plot signals to be saved during simulation. Possible values: 0 (no output), 1 (only most important plot signals), 2 (maximum number of plot signals). Default value 1
<code>al</code>	-	i	default animation model level for all groups and elements. The actual animation model level defines the degree of detailization of the elements' geometrical animation model. Possible values: 0 (no animation model) .., 9 (most detailed animation model). Default value 4
<code>dx</code>	mm	f	default x-direction shift of the design position of all groups and elements. Default: 0.0
<code>dy</code>	mm	f	default y-direction shift of the design position of all groups and elements. Default: 0.0
<code>dz</code>	mm	f	default z-direction shift of the design position of all groups and elements. Default: 0.0
<code>roll</code>	deg	f	default rotation angle about x-axis of the design position of all groups and elements. Default: 0.0
<code>pitch</code>	deg	f	default rotation angle about y-axis of the design position of all groups and elements. Default: 0.0
<code>yaw</code>	deg	f	default rotation angle about z-axis of the design position of all groups and elements. Default: 0.0

3.1.1 Group Definition

A **group definition** is a collection of certain attributes, equipped with a group name. The attributes are used as default values for elements that belong to the group. The membership of an element to a certain group is established in the respective element definition, **not** in the group definition itself.

For any individual element, every group attribute can be overridden by a respective entry in the element definition block. All data definitions in the group definition block are optional. Their respective default values are listed in the table below.

The type of a group definition block is `GR`. The data-block name of a group definition block, at the same time, serves as group name. A typical group definition might look as follows:

```
GR:steering pl=1; al=5; dz=300
```

This block introduces a new group named `steering`. The block sets the default plotting and animation

levels (**p1** and **a1**, resp.) to 1 or 5, respectively, for all elements belonging to the group. Furthermore, a certain default vertical shift (**dz**) is set to 300 mm.

Because **steering** is a block name, there is no need to separate this string by a semicolon from the first block data entry (**p1=1**). On the other hand, a semicolon was allowed there and wouldn't make any difference. The same holds for a semicolon at the end of the line:

```
GR:steering; p1=1; a1=5; dz=300;
```

is equivalent to the group definition above. Likewise, as described in *COSIN/io* documentation, the data entries could have been entered in separate lines, like

```
GR:steering
p1 = 1
a1 = 5
dz = 300
```

or

```
GR:steering
dz 300      ! vertical shift = front wheel static tire radius
p1 1; a1 5  ! plotting and animation level
```

Here is the complete syntax of a group definition block:

```
GR:name p1=value_p1; a1=value_a1; f=data_file;
dx=value_dx; dy=value_dy; dz=value_dz
```

name	-	s (m)	group name (and data-block name at the same time). The group name is used to refer to that group in element definitions
p1	-	i	default plot level for all group elements. The actual plot level defines the number of plot signals to be saved during simulation. Possible values: 0 (no output), 1 (only most important plot signals), 2 (maximum number of plot signals)
a1	-	i	default animation model level for all group elements. The actual animation model level defines the degree of detailization of the elements' geometrical animation model. Possible values: 0 (no animation model) .., 9 (most detailed animation model)
f	-	s	default file name, where the group elements' data-blocks are looked for. Default value: name of actual .cm-file
dx	mm	f	default x-direction shift of the design position of all group elements
dy	mm	f	default y-direction shift of the design position of all group elements
dz	mm	f	default z-direction shift of the design position of all group elements
roll	deg	f	default rotation angle about x-axis of the design position of all group elements
pitch	deg	f	default rotation angle about y-axis of the design position of all group elements
yaw	deg	f	default rotation angle about z-axis of the design position of all group elements

3.1.2 Element Definition

Similar to a group definition, a *COSIN/mbs* element is defined by a type-equipped sub-block of the `$model` block. in the *COSIN/mbs* data file. The type, a two-character acronym, of an element definition block must be one out of a list of valid types as described in the [COSIN/mbs Element Catalogue](#).

The data-block name of an element definition block serves as element name and, at the same time, as default value of the element's data-block name to read its data from.

Besides defining type and name of the element and where to find its data, the most important task of the element definition block is to specify the 'topology' of the *COSIN/mbs* model. This is done in terms of the names of the other elements that are linked to the element, and of the specification of external input and output signals (sources and sinks), to be managed by *COSIN/ss*. The specific meaning of the linked elements and the input and output signals depends on the element, and is described in the respective element's chapter below.

A typical **element definition** might look as follows:

```
SD:spring_f1 l1=car_body; l2=wheel_f1; stiff; g=front_susp
```

This line defines a spring-damper element named `spring_f1`, linking the two bodies `car_body` and `wheel_f1`, belonging to group `front_susp`, and to be recognized as 'stiff' in implicit integration. *COSIN/mbs* data for this element will be searched in data-block `$spring_f1`. Rigid-body elements named `car_body` and `wheel_f1` are also to be defined, either before or after the `spring_f1` definition.

All data definitions and switches in an element definition are optional. For example, there is no need to specify the group entry. Respective default values can be found in the tables below.

The number of other *COSIN/mbs* elements that are linked to an element varies, also the number of external input or output signals to or from the elements. Some of the data and flags of an element definition block can be used with all types of elements. They are described in the table below. Some other entries are element-specific, and are described in the respective element chapter.

This is the general syntax of an element definition block, together with a description of all data and switches common to all element types:

```
XX:name f=data_file; db=data_block_name; mdb=mirrored_data_block
db2=second_data_block_name; mdb2=mirrored_second_data_block
l1=name_l1; l2=name_l2; ...
i1=name_i1; i2=name_i2; ...
o1=name_o1; o2=name_o2; ...
pl=value_pl; an=value_an; camera; stiff; mirror; g=group
dx=value_dx; dy=value_dy; dz=value_dz
additional element-specific data and flags
```

xx	-	st2 (m)	type of element. Valid types are AC, AD, AS, BE, BO, BR, CS, DS, DT, EF, ET, F1, FB, PS, RB, RJ, SC, SD, SR, ST, TI, TJ, TS, WL
name	-	st (m)	arbitrary name of the element. This name can be used to refer to the element in other element definitions. In addition, the name of the element is used to make plot labels unique. Actually, <i>name</i> is the name of a type-equipped <i>COSIN/io</i> data- block

f	-	st	file name, where the group element's data-blocks are looked for. Default value: name of the actual input file, or file name specified in the element's group definition block, resp.
mf	-	sr	file name, where the group element's data-blocks are looked for, and mirrored with respect to the xz-plane (see below). Default value: name of the actual input file, or file name specified in the element's group definition block, resp.
db	-	st	name of the element's data-block (without \$). Default value: element name (<i>name</i>)
mdb	-	st	name of a data-block (without \$) to get the element's data from, after mirroring with respect to the xz-plane. When mirroring, all y-coordinates and all x- and z-angles of the element change sign. <i>COSIN/mbs</i> will first look for the original data-block, defined through the element's name or through the db entry. Only if that data-block is not found, it will search and mirror the mdb entry. Typically, the mechanism established by the mdb entry is used for vehicle suspensions. It allows for only describing the left wheel suspensions (if independent), and get the data for the right wheels by mirroring. The advantages are: smaller input files and considerable reduction of redundancy
db2	-	st	name of the second element data-block (without \$). Default value: element name (<i>name</i>). Used only with the TI element
mdb2	-	st	name of a second data-block (without \$) to get element's data from, after mirroring with respect to the xz-plane. When mirroring, all y-coordinates and all x- and z-angles of the element change sign. Used only with the TI element
i1, i2, ..	-	st	name of the first, second, .. element that is linked to the element to be defined. For force elements, this is the first (second, ..) body the force acts on. The number and meaning of the linked elements are documented with the individual elements below. If a linked element is not defined, ground (with element name #gnd#) is assumed
i1, i2, ..	-	st	name of the first, second, .. signal provided by <i>COSIN/ss</i> to be used as external input signal to the element. The number and meaning of the input signals are documented with the individual elements below. If an input signal is not assigned, its value is held zero

o1, o2, ..	-	st	name of the first, second, .. signal to be forwarded to <i>COSIN/ss</i> as external output signal of the element. If an output signal is not defined, it is not available as input to other elements
pl	-	i	plot level. Defines the number of plot signals to be saved during simulation. Possible values: 0 (no output), 1 (only most important plot signals) , 2 (maximum number of plot signals)
al	-	i	animation model level, defining the level of detail of the elements' geometrical animation model. Possible values: 0, .., 9
camera	-	flag	if set, element serves as 'camera support' (that is, the camera-fixed coordinate system is fixed to the element)
stiff	-	flag	if set, element will be taken into account with implicit BDF integration. Computational effort will increase, but also numerical stability. For force elements that approximate kinematic constraints (like the TJ and RJ element) stiff is automatically set
mirror	-	flag	if set, data in element data-block are mirrored (cf. mdb entry explanation above), no matter whether element data-block is defined by name , or by the db or mdb entry
g	-	st	name of group the element is assigned to
dx	mm	f	shift value for all markers connected to the element, in x-direction
dy	mm	f	shift value for all markers connected to the element, in y-direction
dz	mm	f	shift value for all markers connected to the element, in z-direction
roll	deg	f	rotation angle about x-axis for all markers connected to the element
pitch	deg	f	rotation angle about y-axis for all markers connected to the element
yaw	deg	f	rotation angle about z-axis for all markers connected to the element

3.2 Element Catalogue

A detailed list of all available *COSIN/mb*s element types is provided in a separate documentation, the [COSIN/mb](#)s Element Catalogue. This documentation contains a description of all **element-specific entries** in the element definition block, all **input and output signals**, all **data**, all **animation models**, and all **plot signals**.

All mandatory input data listed there are marked with (m). If not stated otherwise, all numerical data that are neither mandatory nor explicitly specified get the default value 0.

4 Interfacing to Models for Road and Wind Velocity

*COSIN/mb*s vehicle models can be coupled with models for road surface geometry and other road attributes, as well as with models for environmental wind velocity.

4.1 *COSIN/road*

The separate package [COSIN/road](#) (please refer to respective documentation) provides several models for deterministic, stochastic, or measured 2D and 3D road surface geometry and friction attributes. The package includes models for efficient evaluation of high-resolution measured road surfaces, models for non-rigid surfaces, interfaces to user-written road models, and more.

4.2 *COSIN/wind*

To specify environmental wind velocity, *COSIN/mb*s uses the subroutine `ewind`. During the first call to this subroutine in a simulation run, `ewind` opens, if present, the wind file (identifier `IWIND`), looks for data block `$wind` and, if present, reads the following data out of this block that control wind velocity calculation in all subsequent invocations:

<code>type</code>	-	s (m)	a character string out of <ul style="list-style-type: none"> • <code>calm</code> • <code>constant</code> • <code>file_v_of_t</code> • <code>file_v_of_x</code> • <code>table_v_of_t_and_x</code> • <code>table_v_of_x_and_y</code> • <code>function</code> to define the type of the wind velocity description. For most types additional parameters or data blocks are looked for, as explained in the following
<code>frame</code>	-	i	switch to set the co-ordinate system relative to which the wind velocities are defined: <ul style="list-style-type: none"> 0 wind is defined in inertial (global) coordinates 1 wind is defined in a co-ordinate system, the x-axis of which is parallel to the vehicle's velocity vector. This allows for simple prescription of headwind and cross wind, independent on the vehicle's orientation

Depending on the type chosen for wind velocity description, more data are read from data-block `$wind`, according to the following tables:

4.2.1 Type `calm`

COSIN/wind returns zero wind velocities. No other data are required.

4.2.2 Type `constant`

COSIN/wind sets all components of wind velocities to constant values, that are read from data-block `$wind`:

vx	m/s	f	constant x-component of wind velocity. Default value is 0
vy	m/s	f	constant y-component of wind velocity. Default value is 0
vz	m/s	f	constant z-component of wind velocity. Default value is 0

4.2.3 Type `file_v_of_t`

With this type, wind velocities are described by data columns of a file, giving the dependency of wind velocity on time. *COSIN/wind* reads from data-block `$wind`:

file_name	-	st	name of a file to read measured or calculated data from. These data describe the dependency of v_x and v_y on time. The file must be in standard or Matlab format (cf. COSIN/io documentation). <i>COSIN/wind</i> expects equidistant wind velocity data as columns of the matrix stored in the file. Unit is [m/s]
channel_vx	-	i	column index of v_x data (if file contains the time channel as first channel, this will typically be 2)
channel_vy	-	i	column index of v_y data (if file contains the time channel as first channel, this will typically be 3)
sampling_time	s	f	time difference between two consecutive measured wind velocities. Time values outside the range of data in the file get their wind velocity by extrapolation

Note that data columns in a file also can be used with the more general type `function` (see below), in conjunction with one of the interpolation functions `pconst()`, `linear()`, or `spline()`.

4.2.4 Type `file_v_of_x`

With this type, wind velocities are described by data columns of a file, giving the dependency of wind velocity on travel distance. *COSIN/wind* reads from data-block `$wind`:

file_name	-	st	name of a file to read measured or calculated data from. These data describe the dependency of v_x and v_y on inertial coordinate x , or travel distance, resp. The file must be in standard or Matlab format (cf. COSIN/io documentation). <i>COSINwind</i> expects equidistant wind velocity data as columns of the matrix stored in the file. Unit is [m/s]
channel_vx	-	i	column index of v_x data (if file contains the travel distance as first channel, this will typically be 2)

<code>channel_vy</code>	-	i	column index of v_y data (if file contains the travel distance as first channel, this will typically be 3)
<code>meas_distance</code>	m	f	distance between two consecutive measured points in longitudinal direction. Travel distances outside the range of data in the file get their wind velocity by extrapolation

Note that data columns in a file also can be used with the more general type `function` (see below), in conjunction with one of the interpolation functions `pconst()`, `linear()`, or `spline()`.

4.2.5 Type `table_v_of_t_and_x`

With this type, wind velocities are described by a 2D look-up table, giving the dependency of wind velocity on time and on travel distance. *COSIN/wind* looks for the table data in sub-data blocks `$vx_of_t_and_x` and `$vy_of_t_and_x`. For the syntax of spline data definitions, cf. [COSIN/io](#) documentation.

Note that table data also can be used with the more general type `function` (see below), in conjunction with one of the interpolation functions `bilin()` or `bicub()`.

4.2.6 Type `table_v_of_x_and_y`

With this type, wind velocities are described by a 2D look-up table, giving the dependency of wind velocity on inertial x/y co-ordinates, or on travel distance and lateral distance to travel path, resp. *COSIN/wind* looks for the table data in sub-data blocks `$vx_of_x_and_y` and `$vy_of_x_and_y`. For the syntax of table data definitions, cf. [COSIN/io](#) documentation.

Note that table data also can be used with the more general type `function` (see below), in conjunction with one of the interpolation functions `bilin()` or `bicub()`.

4.2.7 Type `function`

With this type, all 3 components of the wind velocity are described by general arithmetic expressions, that are allowed to contain t , x , y , and z . *COSIN/wind* reads from data-block `$wind`:

<code>vx</code>	m/s	f	x-component or longitudinal component, resp., of wind velocity, as function of x , y , z , t , and optional further COSIN/io parameters. That function may be an arbitrary arithmetic expression as documented in <i>COSIN/io</i> , including 1D and 2D lookup tables, random values, etc. Default value is 0
<code>vy</code>	m/s	f	y-component or lateral component, resp., of wind velocity, as function of x , y , z , t , and optional further COSIN/io parameters. That function may be an arbitrary arithmetic expression as documented in <i>COSIN/io</i> , including 1D and 2D lookup tables, random values, etc. Default value is 0

vz	m/s	f	z-component (vertical component) of wind velocity, as function of x , y , z , t , and optional further COSIN/io parameters. That function may be an arbitrary arithmetic expression as documented in <i>COSIN/io</i> , including 1D and 2D lookup tables, random values, etc. Default value is 0
-----------	-----	---	--

Under any of the Windows™ operating systems, compiling user-written ECU software in C language can be done with Microsoft™ Visual C++ V5.0 or higher, using the batch command file `mdll.bat`. To do this, copy the C code into *COSIN/products* sub-folder `car/c_code` (overriding the template C code of the respective ECU), then change to sub-folder `car` and enter

```
mdll c-code-name
```

where `c-code-name` is one out of the list

- `abs_std` (standard ABS),
- `abs_adv` (advanced ABS),
- `esp` (ESP),
- `fwd` (4WD select),
- `atts` (front-wheel ATTS),
- `lsd_std` (conventional LSD),
- `lsd_act` (active LSD),
- `clid` (controlled partially locking differentials),
- `fws` (four-wheel steering 4WS),
- `eps_std` (simple EPS),
- `eps_adv` (advanced EPS),
- `actuator_i` (general force actuator, $i = 1, \dots, 9$).

If compiling and linking was successful, a DLL (Dynamic Link Library) will be generated and copied to the *COSIN/products* `bin` folder. During a subsequent simulation, if the respective ECU functionality is requested in the `.cm`-file, this DLL will be used.

When implementing your code, it is very important to observe the following rule: if the C-code is called from a *COSIN/mbs* element with the `stiff` property set, then your code should not use any static variables. Such variables can act as ‘hidden’ state variables. Hidden state variables seriously disturb the implicit integration algorithm, because they make it impossible to properly calculate the Jacobian of the system.

Instead of using static variables, if the calling *COSIN/mbs* element is `stiff`, your code should ‘remember’ all necessary values by using the `state` array. At present, the `state` array is only available in `actuator_i`. On demand, all other ECU interfaces can be extended accordingly.

The following sub-chapters give more details for any of the C-code functions.

5.1 Standard ABS

Prototype:

```
extern void abs_std
(int*flag, double*bp,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*bp_fl, double*bp_fr, double*bp_rl, double*bp_rr,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
<code>int*flag</code>	input	-	indicates first or subsequent call. For <code>flag=0</code> , output need not to be provided
<code>double*bp</code>	input	bar	nominal brake pressure

double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*bp_fl double*bp_fr double*bp_rl double*bp_rr	output	bar	individual wheel brake pressures
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.2 Advanced ABS

Prototype:

```
extern void abs_adv
(int*flag, double*bp,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc, double*long_acc,
 double*bp_fl, double*bp_fr, double*bp_rl, double*bp_rr,
 double*t_eng,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*bp	input	bar	nominal brake pressure
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*yaw_rate	input	deg/s	vehicle yaw rate
double*lat_acc	input	m/s ²	vehicle lateral acceleration
double*long_acc	input	m/s ²	vehicle longitudinal acceleration
double*bp_fl double*bp_fr double*bp_rl double*bp_rr	output	bar	individual wheel brake pressures
double*t_eng	output	Nm	nominal engine torque, to be regulated in propulsion system. Will not be used, if negative
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.3 ESP (Electronic Stability Program)

Prototype:

```
extern void esp
(int*flag, double*bp,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc, double*long_acc,
 double*gp, double*swa,
 double*bp_fl, double*bp_fr, double*bp_rl, double*bp_rr,
 double*t_eng,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*bp	input	bar	nominal brake pressure
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*yaw_rate	input	deg/s	vehicle yaw rate
double*lat_acc	input	m/s ²	vehicle lateral acceleration
double*long_acc	input	m/s ²	vehicle longitudinal acceleration
double*gp	input	%	gas pedal stroke (100% = full throttle)
double*swa	input	deg	steering wheel angle, positive for left turns
double*bp_fl double*bp_fr double*bp_rl double*bp_rr	output	bar	individual wheel brake pressures
double*t_eng	output	Nm	nominal engine torque, to be regulated in propulsion system. Will not be used, if negative
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.4 Automatic Four-Wheel Drive Select

Prototype:

```
extern void fwd
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc, double*long_acc,
 double*t_eng, double*swa,
 int*fwd_switch,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*yaw_rate	input	deg/s	vehicle yaw rate
double*lat_acc	input	m/s ²	vehicle lateral acceleration
double*long_acc	input	m/s ²	vehicle longitudinal acceleration
double*t_eng	input	Nm	net engine torque
double*swa	input	deg	steering wheel angle, positive for left turns
int*fwd_switch	output	-	four-wheel drive select switch 0: 4WD switched off, 1: 4WD switched on
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.5 ATTS (Automatic Torque Transfer System)

Prototype:

```
extern void atts
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc,
 double*t_eng, double*swa,
 double*t_split_factor,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*yaw_rate	input	deg/s	vehicle yaw rate
double*lat_acc	input	m/s ²	vehicle lateral acceleration
double*t_eng	input	Nm	net engine torque
double*swa	input	deg	steering wheel angle, positive for left turns
double*t_split_factor	output	-	factor between 0 and 1 (continuously) that controls torque split between left and right front wheel 0: right wheel receives full torque 0.5: ideal differential, both wheels receive same torque 1: left wheel receives full torque
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.6 LSD (Limited Slip Differential)

Prototype:

```
extern void lsd_std
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*t_eng, double*dt_max,
 int*ier)
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*t_eng	input	Nm	net engine torque
double*dt_max	output	-	maximum difference torque of the two output shafts
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.7 Active LSD

Prototype:

```
extern void lsd_act
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc, double*long_acc,
 double*t_eng, double*swa, double*dt_max,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*yaw_rate	input	deg/s	vehicle yaw rate
double*lat_acc	input	m/s ²	vehicle lateral acceleration
double*long_acc	input	m/s ²	vehicle longitudinal acceleration
double*t_eng	input	Nm	net engine torque
double*swa	input	deg	steering wheel angle, positive for left turns
double*dt_max	output	-	maximum difference torque of the two output shafts
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.8 CLD (Controlled Partially Locking Differentials)

Prototype:

```
extern void cld
(int*flag,
 double*throttle, int*gear, double*brp, double*swa,
 double*speed, double*yaw_rate,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*lat_acc, double*long_acc,
 double*dt_maxf, double*dt_maxr, double*dt_maxc,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*throttle	input	deg	throttle opening angle
int*gear	input	-	actually actuated gear
double*brp	input	bar	pressure in main brake cylinder [bar]
double*swa	input	deg	steering wheel angle, positive for left turns
double*speed	input	m/s	vehicle speed [m/s]
double*yaw_rate	input	deg/s	vehicle yaw rate
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*lat_acc	input	m/s ²	vehicle lateral acceleration

double*long_acc	input	m/s ²	vehicle longitudinal acceleration
double*dt_maxf	output	-	maximum difference torque of the two front wheels output shafts
double*dt_maxr	output	-	maximum difference torque of the two rear wheels output shafts
double*dt_maxc	output	-	maximum difference torque of the central differential output shafts
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.9 Four-Wheel Steering

Prototype:

```
extern void fws
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc, double*swa,
 double*d_rack,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*yaw_rate	input	deg/s	vehicle yaw rate
double*lat_acc	input	m/s ²	vehicle lateral acceleration
double*swa	input	deg	steering wheel angle, positive for left turns
double*d_rack	output	mm	rack displacement of rear axle steering assembly
int*ier	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.10 EPS (Electronic Power Steering)

Prototype:

```
extern void eps_std
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*swt,
 double*f_rack, double*t_col,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
int*flag	input	-	indicates first or subsequent call. For flag=0, output need not to be provided
double*ws_fl double*ws_fr double*ws_rl double*ws_rr	input	rad/s	wheel rotational speeds
double*swt	input	Nm	steering reaction torque (pinion torque)

<code>double*f_rack</code>	output	N	assisting force on steering rack
<code>double*t_col</code>	output	Nm	assisting torque on steering column
<code>int*ier</code>	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.11 Advanced EPS

Prototype:

```
extern void eps_adv
(int*flag,
 double*ws_fl, double*ws_fr, double*ws_rl, double*ws_rr,
 double*yaw_rate, double*lat_acc,
 double*swa, double*swv, double*swt, double*swtr,
 double*f_rack, double*t_col,
 int*ier);
```

Parameter:

name	data flow	unit	meaning
<code>int*flag</code>	input	-	indicates first or subsequent call. For <code>flag=0</code> , output need not to be provided
<code>double*ws_fl</code> <code>double*ws_fr</code> <code>double*ws_rl</code> <code>double*ws_rr</code>	input	rad/s	wheel rotational speeds
<code>double*yaw_rate</code>	input	deg/s	vehicle yaw rate
<code>double*lat_acc</code>	input	m/s ²	vehicle lateral acceleration
<code>double*swa</code>	input	deg	steering wheel angle (positive in left turns)
<code>double*swv</code>	input	rad/s	steering wheel angle velocity
<code>double*swt</code>	input	Nm	steering reaction torque (pinion torque)
<code>double*swtr</code>	input	Nm/s	steering reaction torque rate
<code>double*f_rack</code>	output	N	assisting force on steering rack
<code>double*t_col</code>	output	Nm	assisting torque on steering column
<code>int*ier</code>	output	-	error indicator. Should be non-zero at return from function, if an error occurred

5.12 General Force Actuator

Prototype:

```
extern void actuator_i
(int*flag, double*s, double*sd, double*dt, double*state, double*F, int*ier);
(i = 1,..,9)
```

Parameter:

parameter	data flow	unit	meaning
<code>int*flag</code>	input	-	indicates first or subsequent call. For <code>flag=0</code> , output need not to be provided
<code>double*s</code>	input	mm	deflection
<code>double*sd</code>	input	m/s	deflection velocity
<code>double*dt</code>	input	s	integration step-size
<code>double*state</code>	input/ output	user defined	state vector, may contain up to 10 components
<code>double*F</code>	output	N	scalar actuator force

<code>int*ier</code>	output	-	error indicator. Should be non-zero at return from function, if an error occurred
----------------------	--------	---	---

6 Interfacing to User-Supplied Driver-Models

COSIN/mbs provides an interface to user-supplied driver-models. This interface is defined and implemented in a similar way like the ECU models. A driver-model can be provided by the user as C, C++, or Fortran code. This code must observe certain interfacing rules, which are described below.

Usage of a user-supplied driver-model is selected only by specifying a related data-file (`udm_file`) in data-block `$simulation` of the sim-file, cf. [chapter 2.2.1](#).

A C-language template (`udm.c`) for a driver-model can be found in *COSIN/mbs* sub-folder `c_code`. Note that all parameters of this C-function must be referenced by their addresses; that is, all parameters are either of data type `double*`, `int*`, or `char*`.

Under any of the Windows™ operating systems, compiling the driver-model source code in C language can be done with Microsoft™ Visual C++ V5.0 or higher, using the batch command file `md11.bat`. To do this, copy the C code into *COSIN/products* sub-folder `car/c_code` (overriding the template C code of the driver-model), then change to sub-folder `car` and enter

```
mdll udm
```

If compiling and linking was successful, a DLL (Dynamic Link Library) will be generated and copied to the *COSIN/products* `bin` folder. During a subsequent simulation, if the user-supplied driver-model is called, this DLL will be used.

The driver-model is only called once per simulation step, with a fixed step-size. As a consequence, there is no complicated step-size controlled co-simulation technique necessary. In contrast to ECU models called by elements with the stiff attribute set (cf. [chapter 5](#)), the driver-model may use arbitrary hidden state variables without interfering with *COSIN/mbs'* integrator in any way.

Prototype:

```
extern void udm
(int*job, double*din, double*dout, int*ier, char*file);
```

Parameter:

name	data flow	meaning																																				
<code>int*job</code>	input	job control flag 0: normal call, output signals to be provided 1: final call, no output signals required by <i>COSIN/mbs</i> At present, the driver model has to take care for loading the data file only during the first call to <code>udm</code> ; no special job-value for 'initialization' is provided at that time.																																				
<code>double*din</code>	input	vehicle signals made available as sensor signals: <table border="1"> <tr> <td><code>din[0]</code></td> <td>s</td> <td>actual simulation time</td> </tr> <tr> <td><code>din[1]</code></td> <td>m</td> <td>x-position driver's eye in global coord.</td> </tr> <tr> <td><code>din[2]</code></td> <td>m</td> <td>y-position driver's eye in global coord.</td> </tr> <tr> <td><code>din[3]</code></td> <td>m/s</td> <td>x-velocity driver's eye in global coord.</td> </tr> <tr> <td><code>din[4]</code></td> <td>m/s</td> <td>y-velocity driver's eye in global coord.</td> </tr> <tr> <td><code>din[5]</code></td> <td>m/s²</td> <td>x-acceleration driver's eye in global coord.</td> </tr> <tr> <td><code>din[6]</code></td> <td>m/s²</td> <td>y-acceleration driver's eye in global coord.</td> </tr> <tr> <td><code>din[7]</code></td> <td>deg</td> <td>vehicle's yaw angle (ISO 8855)</td> </tr> <tr> <td><code>din[8]</code></td> <td>deg/s</td> <td>vehicle's yaw velocity (ISO 8855)</td> </tr> <tr> <td><code>din[9]</code></td> <td>deg/s²</td> <td>vehicle's yaw acceleration (ISO 8855)</td> </tr> <tr> <td><code>din[10]</code></td> <td>Nm</td> <td>steering torque</td> </tr> <tr> <td><code>din[11]</code></td> <td>-</td> <td>not used</td> </tr> </table>	<code>din[0]</code>	s	actual simulation time	<code>din[1]</code>	m	x-position driver's eye in global coord.	<code>din[2]</code>	m	y-position driver's eye in global coord.	<code>din[3]</code>	m/s	x-velocity driver's eye in global coord.	<code>din[4]</code>	m/s	y-velocity driver's eye in global coord.	<code>din[5]</code>	m/s ²	x-acceleration driver's eye in global coord.	<code>din[6]</code>	m/s ²	y-acceleration driver's eye in global coord.	<code>din[7]</code>	deg	vehicle's yaw angle (ISO 8855)	<code>din[8]</code>	deg/s	vehicle's yaw velocity (ISO 8855)	<code>din[9]</code>	deg/s ²	vehicle's yaw acceleration (ISO 8855)	<code>din[10]</code>	Nm	steering torque	<code>din[11]</code>	-	not used
<code>din[0]</code>	s	actual simulation time																																				
<code>din[1]</code>	m	x-position driver's eye in global coord.																																				
<code>din[2]</code>	m	y-position driver's eye in global coord.																																				
<code>din[3]</code>	m/s	x-velocity driver's eye in global coord.																																				
<code>din[4]</code>	m/s	y-velocity driver's eye in global coord.																																				
<code>din[5]</code>	m/s ²	x-acceleration driver's eye in global coord.																																				
<code>din[6]</code>	m/s ²	y-acceleration driver's eye in global coord.																																				
<code>din[7]</code>	deg	vehicle's yaw angle (ISO 8855)																																				
<code>din[8]</code>	deg/s	vehicle's yaw velocity (ISO 8855)																																				
<code>din[9]</code>	deg/s ²	vehicle's yaw acceleration (ISO 8855)																																				
<code>din[10]</code>	Nm	steering torque																																				
<code>din[11]</code>	-	not used																																				

		din[12]	-	not used
		din[13]	-	not used
		din[14]	rpm	engine speed
		din[15]	-	actual gear selection (-1: reverse gear, 0: neutral, 1: 1 st gear, etc.)
		din[16]	%	wheel slip front left wheel
		din[17]	%	wheel slip front right wheel
		din[18]	%	wheel slip 1 st rear axle left wheel
		din[19]	%	wheel slip 1 st rear axle right wheel
		din[20]	deg	side-slip angle front left wheel
		din[21]	deg	side-slip angle front right wheel
		din[22]	deg	side-slip angle 1 st rear axle left wheel
		din[23]	deg	side-slip angle 1 st rear axle right wheel
double*dout	output	driver's control signals computed by the driver model:		
		dout[0]	deg	steering wheel angle (counterclockwise)
		dout[1]	-	gas pedal operating travel (normalized between 0 and 1)
		dout[2]	-	brake pedal operating travel (normalized between 0 and 1)
		dout[3]	-	clutch pedal operating travel (normalized between 0 and 1)
		dout[4]	-	gear selection (-1: reverse gear, 0:neutral, 1: 1 st gear, etc.)
		dout[5]	-	hand-brake lever operating travel (normalized between 0 and 1)
int*ier	output	error flag 0: normal and successful operation of the driver model All other values will abort the <i>COSIN/mbs</i> simulation		
char*file	input	name of the data file containing model data, to be opened, read, interpreted, and used solely by the driver-model. Path separators etc. follow the respective operating system's rules		

7 Command Line Invocation

Instead of using the *COSIN/mbs* workbench, likewise the *COSIN/mbs* solver can be called directly from a command window (Windows™) or command shell (Unix), respectively. For Windows™, that invocation reads

```
<cosin-path>\bin\cosinmbs cfd-file -option1 -option2 ...
```

or simply

```
cosinmbs cfd-file -option1 -option2 ...
```

if you have added `<cosin-path>\bin` to your search-path environment variable. For Unix-type operating systems, replace the back-slashes by slashes.

In the above commands, `<cosin-path>` is the path where you installed *COSIN/products* (during installation under Windows™, you might have changed the default value `c:\cosin_products` for that path). `<cfd-file>` is the file-definition file (cf. [COSIN/io](#)), containing a list of assignments between logical file identifiers and ‘physical’ file names. A *COSIN/mbs* .cfd-file might look as follows (in that example, it is assumed your actual working directory is `<cosin-path>/car`):

```
* sample .cfd-file for COSIN/mbs command line invocation *****

ISIM=simul/cornering.sim
ISRC=simul/cornering.sim
IROAD=simul/cornering.sim
IWIND=simul/cornering.sim
IPAR=param/st_fl.cm

OPAR=param/st_fl.cmp
OBIST=simul/temp.cms
OPLOT=work/temp
```

Note that simulation data (data-block `$simulation`), external input-signal data (data-block `$sources`), and road-profile data (data-block `$road_type`) are combined to only one file (`simul/cornering.sim`). This is also done when calling the solver from the workbench, but by no means obligatory.

Note also that even if called under Windows™, a path in a .cfd-file may be defined by using slashes instead of back-slashes. This facilitates interchange of files between Windows™ and Unix.

The following logical file identifiers can be specified:

logical file identifier	data flow (m: mandatory)	meaning of file
ISIM	input (m)	simulation file, containing data-block <code>\$simulation</code>
ISRC	input	sources& sinks file, containing data-block <code>\$sources</code>
IROAD	input	road data file, containing data-block <code>\$road_type</code>
IWIND	input	wind data file, containing data-block <code>\$wind</code>
IPAR	input (m)	model data file (.cm-file)
IBIST	input	model states input file
OPAR	output (ASCII)	copy of model data file (.cmp-file), pre-load values added
OBIST	output (binary)	model states output file
OPLOT	output (ASCII or binary)	plot file (if file name has no extension, an extension will be appended according to the plot-file format specified in the <code>\$simulation</code> data-block)

In addition to the .cfd-file, certain program options can be defined. The following is a list of all avail-

able options:

program option	meaning
-acc	('accurate'): decrease simulation time step by 50%
-anim	perform <i>COSIN/graphics</i> on-line animation during simulation
-bpglpathxxxx	set path to look for <i>COSIN/graphics</i> binary files to <i>xxxx</i> . Normally, <i>xxxx</i> will be <i><cosin-path>\bin</i> or <i><cosin-path>/bin</i> , resp., if you have installed <i>COSIN/products</i> to <i><cosin-path></i> . Be sure not to put a blank space between <i>bpglpath</i> and <i>xxxx</i> Remark: if that option is set, but not properly, on-line animation will not work
-clean	before launching simulation, delete temporary files which might be left over from previous simulations that did not finish regularly
-fast	increase simulation time step by 50%
-movie	prepare movie generation by grabbing every single animation frame into a .bmp-file. These files will be saved to folder <i>.\temp</i> or <i>./temp</i> , resp.
-plot	generate a plot-file

8 Coupling to Matlab/Simulink™

If licensed to, you can use any *COSIN/mbs* assembly as block in a Matlab/Simulink™ model, see [figure 8](#).

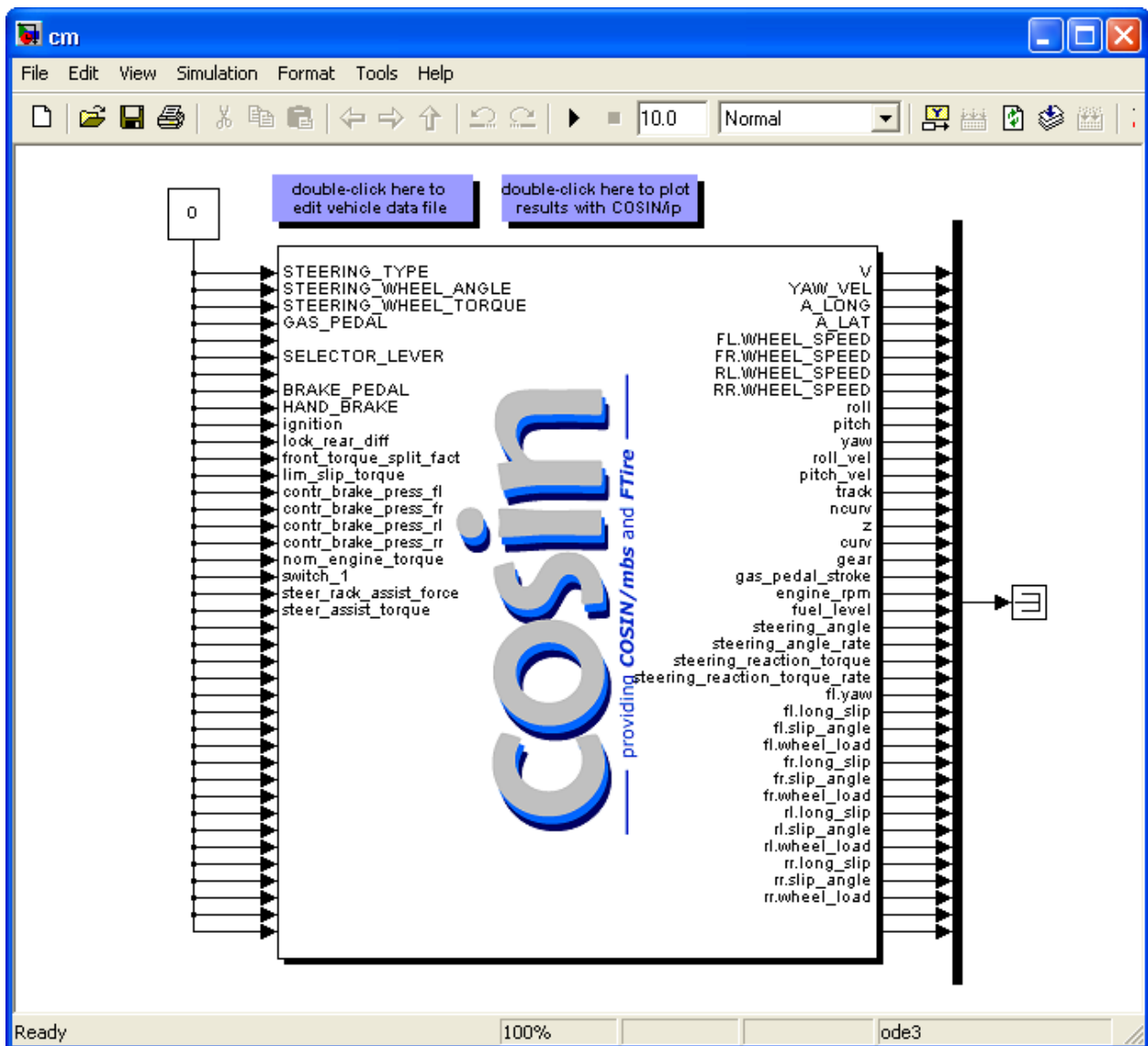


Figure 8: *COSIN/mbs* Simulink™ block

The central block in [figure 8](#), the *COSIN/mbs* block, can be used in several instances, which will in no way interfere with each other. The block (with those sample external connections as shown in [figure 8](#)) is called *cm.mdl* and can be loaded into Simulink™ after changing to sub-folder *matlab* of the *COSIN/products* folder. Note that simulations with *COSIN/mbs* under Simulink™ can not reach the full speed experienced with *COSIN/mbs* stand-alone. This is inevitable, and due to a certain computational overhead in Simulink™.

The block (for experts: it is implemented as time-discrete wrapped and masked C-code level-2 S-function) automatically shows all those input and output signals that are defined in the underlying *COSIN/mbs* model file (the *cm*-file). It is up to the user to define all those input and output signals he wants to use with the Simulink™ block. All *COSIN/mbs* sample files provide the most important ones of these signals.

Due to certain restrictions with the programming and masking of Simulink™ blocks, the *COSIN/mbs* block will always show 40 input signals and 40 output signals, even if the *cm*-file defines less. Surplus input

signals are not used, while surplus output signals will put out constantly zero.

To get the best compromise between flexibility in user-defined signals, and interchangeability of *COSIN/mb*s blocks for different vehicles, the most commonly used signals (with pre-defined names) will always appear in the same places of the block. These signals are labeled with upper-case letters, while all other signals are shown in lower-case letters. If a pre-defined signal does not appear in the cm-file, its place is left blank.

At present, the following input signals are pre-defined:

steering_type	-	i	steering type (signal i1 in SR or SC element)
steering_wheel_angle	deg	f	steering wheel angle (signal i2 in SR or SC element)
steering_wheel_torque	Nm	f	steering wheel torque (signal i3 in SR or SC element)
gas_pedal	-	f	normalized gas pedal travel (signal i1 in SR or DT element)
clutch_pedal	-	f	normalized clutch pedal travel (signal i2 in PS element)
selector_lever	-	i	selector lever position (signal i2 in ST element)
gear	-	i	gear to actuate (signal i3 in ST element)
brake_pedal	-	f	normalized brake pedal travel (signal i1 in SR or DT element)
hand_brake	-	f	normalized hand-brake handle travel (signal i2 in BR element)

These are the pre-defined output-signals:

v	m/s	f	vehicle total speed (signal o1 in car-body BO element)
yaw_vel	rad/s	f	vehicle yaw velocity (signal o13 in car-body BO element)
a_long	m/s ²	f	vehicle longitudinal acceleration in car-body fixed co-ordinates (signal o5 in car-body BO element)
a_lat	m/s ²	f	vehicle lateral acceleration in car-body fixed co-ordinates (signal o6 in car-body BO element)
fl.wheel_speed	rad/s	f	rotational wheel speed of front left wheel (signal o2 in front left wheel TI element)
fr.wheel_speed	rad/s	f	rotational wheel speed of front right wheel (signal o2 in front right wheel TI element)
r1.wheel_speed	rad/s	f	rotational wheel speed of rear left wheel (signal o2 in rear left wheel TI element)
rr.wheel_speed	rad/s	f	rotational wheel speed of rear right left wheel (signal o2 in rear right wheel TI element)

When double-clicking on the *COSIN/mb*s blocks, an input mask opens like shown in [figure 9](#).



Figure 9: COSIN/mbs Input Mask in Simulink™

Here, you can enter, in Matlab/Simulink™ syntax,

- file names of the .cm-file and of the road model data file to be used for the assembly,
- values for initial position and velocity of the car-body,
- the *COSIN/mbs* integration, plot-output, and animation step-sizes ('time-step'),
- the log-level which determines the details of terminal output (which will appear in the Matlab™ Command Window).

If the plot-output step-size is greater than zero, during a simulation a file **temp.mtl** will be generated in the working directory. This file contains all *COSIN/mbs* plot-output signals and can be loaded in the Matlab™ command window as follows:

```
load -ascii temp.mtl
```

Equally well, temp.mtl can be browsed with *COSIN/ip*. *COSIN/ip* is opened by double-clicking the right magenta block above the *COSIN/mbs* block. Double-clicking the left one will open the cm-file with Matlab™'s text editor.

Before running a simulation, you can check in the input mask whether or not to observe dry friction and backlash ('play') in all *COSIN/mbs* elements. For linearization in Matlab™, it is highly recommended to switch off both dry friction and backlash. In contrast, for non-linear time-domain simulations it is recommended to switch on both.

Using the last check-box in the input mask, you can toggle on-line animation.