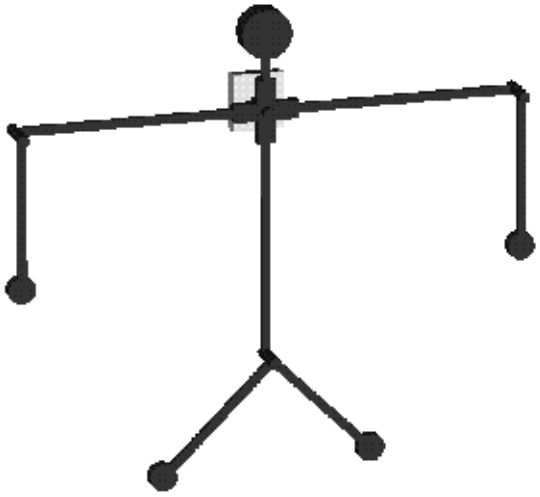
The
KISMET Guide
To
DYNAMICS

(KISMET Version 4.95 and later)



Document Version 1.1
Revision: 1998-08-13

Hans-Georg Krumm
Dr.-Ing. Uwe Kühnapfel

Forschungszentrum Karlsruhe
Institut für Angewandte Informatik

Table of Contents

List of Illustrations

List of Tables:

1 Introduction

This document covers specific KISMET features enabling the user to simulate the dynamic behaviour of a mechanism with or without control system in real time as well as the elastomechanical behaviour of a mechansim for visualizing the quasi-static deformations.

The document consists of 3 parts:

Part I: The Example Part (Chapter 2):

This part explains the basic steps for creating dynamic models.

Part II: The Manual Part: (Chapters 3 to 5)

This part explains most of the commands you need in part I as well as commands for simulating the dynamic behaviour in real-time.

Part III: The Data Specification part:(Chapter 6)

This part explains the underlying data structure for dynamic simulation

2 Examples: A Step by Step Guide

2.1 Introduction

Chapter 2 demonstrates how to build a kinematic and dynamic model of a mechanism.

2.2 Important menus:

The most important menus to build kinematic models are listed in Figure 1:

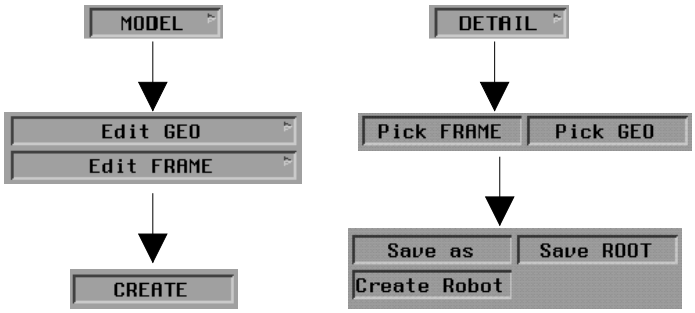


Figure 1:Important menus

2.3 How to create a kinematic model

The basic steps are:

1. Open am empty model
2. Create frames to define the "kinematic skeleton" using the **Edit Frame/Create frame** sub-menu. Use frametype 0 to define frames with a rotational degree of freedom or frametype 1 to define frames with a translational degree of freedom. Use frametype 50 for modeling frames.
3. Create geometric shapes and connect them with the created frames using the **Edit GEO/CREATE** command.
4. Save the model by
    - Using the **Detail/Save robot** in order to create a complete new model or
    - Using the **Detail/Pick GEO** or **Detail/Pick Frame** command followed by the **Save as** command in order to save only parts of the existing models.

2.4 How to create a dynamic model

The basic steps are:

1. Create a kinematic structure as described in chapter 2.3 or use an already existing kinematic model.
2. Make sure that the last joint frame (type 0 or 1) has a kinemtic successor.
3. Create a "Deflected Joint Frame" (frametype 10) and connect it with the first joint frame (type 0 or 1) using the **"Model/EDIT Frame/CREATE"** command. Use the next joint frame as successor frame.
4. Repeat step 3 for all joints.

5. Create a "Center of mass Frame" (frametype 60) and connect it at the first "Deflected Joint Frame" using the "**Model/EDIT Frame/CREATE"** command. Enter the data describing the dynamic behaviour of the corresponding link.
6. Move the "Center of mass" frame to the center of gravity of the corresponding link.
7. Repeat step 5 and 6 for all joints.
8. Save your model.

3 Creation of dynamic specific frames and data sets

3.1 Introduction

For dynamic modeling the user needs to create certain frames of type 10, 15, 60 and 61. By creating these frames the user is enabled to enter dynamic parameters into the model data base.

3.2 The Model/EDIT FRAME/Create submenu.

Using the **Model/EDIT Frame/Create** submenu produces the following output on the textport:

```
Enter JOINT_TYPE[0 = REVOLUTE_JOINT,  1 = PRISMATIC_JOINT,
                 2 = REVOLUTE_LINK,   3 = PRISMATIC_LINK,
                10 = DEFLECTED_LINK,  15 = Branch_Base
                50-53 = MODELING_FRAME
                60 = CG_FRAME,        61 = JNT_DEFORM_FRAME ]
```

The input of dynamic relevant parameters for one link or joint is done by creating a *Center of Gravity* frame (type 60). The user has to create interactively up to 3 files, containing the parameters:

.def -file The ".def"-file describes certain parameters of one link of a mechanism that are mandatory for "Elastomechanics" and / or "Dynamics"

.stf-file The ".stf"-file contains the stiffness matrix of one link.

.ita-file The ".ita"-file defines the inertia matrix of one link.

3.2.1 Creating the ".def"-files

Selecting frame type 60 produces the following output:

```
Do you want to use an existing data set (y/n) ?
```

The KISMET user has 2 possibilities to enter the parameters by:

a) using an already existing ".def" file and value modification, which is strongly recommended, or by

b) typing in each demanded value via textport.

Possibility a) leads to a file selection menu enabling the user to select a ".def"-file. The following mask will appear on the screen after file selection:

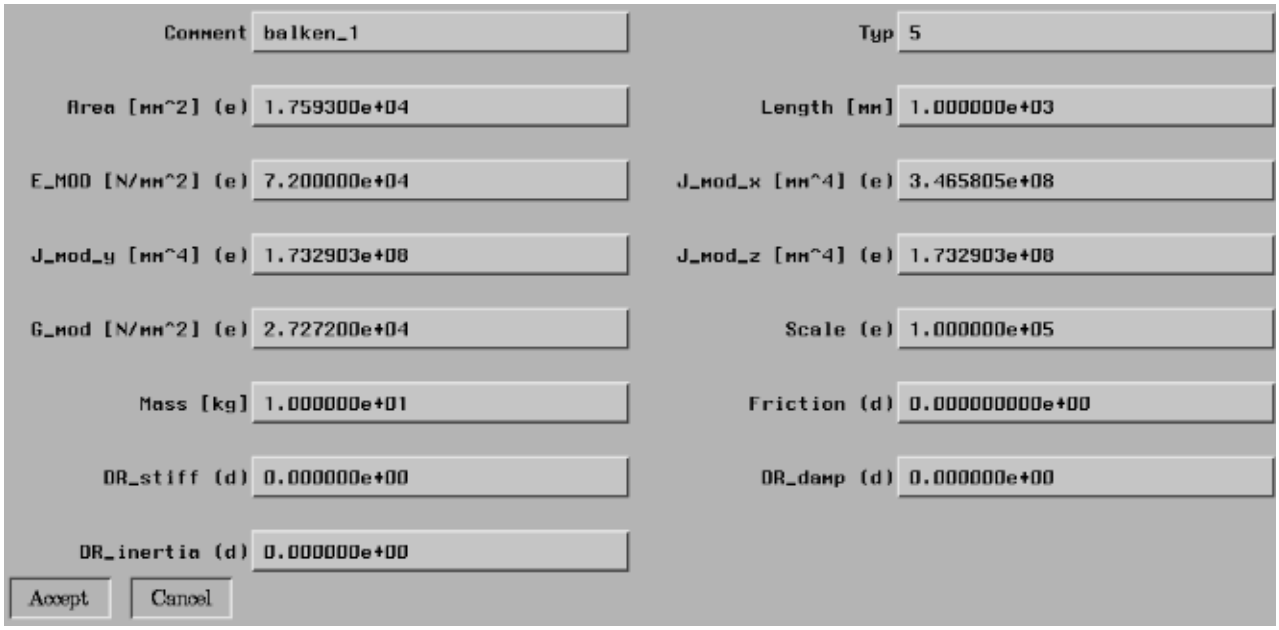


Figure 2: The ".def" file mask

This file contains values necessary for Elastomechanics simulation marked with (e) as well as values necessary for Dynamics simulation marked with (d). Unmarked values belong to both features. See chapter 6.1 for details.

For dynamic simulation the next step will be to produce the ".ita"-file containing the inertia matrix of one link of the mechanism.

3.2.2 Creating the ".ita"-files

After having defined the ".def"-file values the user has to enter the values of the inertia matrix of the corresponding link. There are 3 possibilities to enter the inertia matrix:

Figure 3: The Inertia Matrix Creation box

1) **Geo:** If the shape of the corresponding link can be described by one of the following solid geometric primitives:

   - box
   - pipe
   - sphere

   the inertia matrix will be calculated automatically using the geometric information in combination with the already specified mass.

Figure 4: The Inertia-by-geo box

2) **File:** If the shape cannot be calculated by using possibility 1) the user can use an existing inertia-file and modify the values or

3) **Edit:** the user has to define a totally new inertia matrix by typing in each single value via textport.

3.2.3 Creating the .stf file

Depending on the value of the variable *mpd-Typ* (see Chapter 6.1) the user has to enter values for the stiffness matrix file (suffix ".stf") or not. The user has the possibility to use an existing ".stf-file or to enter each single value via textport.

4 The DETAIL submenu

4.1 Introduction

This chapter describes the **DETAIL** submenu, visible for the user while executing KISMET in the non-full-screen-mode. In full-screen-mode a different user interface using pop-up menus will appear. The corresponding menu-names are surrounded by {}.

4.2 DETAIL Submenu {Edit ABSTRACT}

The "DETAIL" submenu is used

- to load, activate and deactivate model-parts and details and
- to create an new (robot) model.

In KISMET, an ABSTRACT is a kind of special FRAME, which acts as a reference for an assembly, or to another, more detailed representation of the part. The physical representation of the ABSTRACT is a file ('.mpc-file') which defines a part with its FRAMES and file-references to the GEO-element data ('.mpo-files'). An ABSTRACT-file is also used to define exactly one ROBOT.

As a special feature in KISMET, each frame defined in an ABSTRACT file can store the file reference to another ABSTRACT file. This feature is used to define multiple levels of detail in KISMET. It is possible to control the simulation performance (frame rate) over a wide range in KISMET, using this feature. To get the optimum performance, you should only activate those parts and details of the application model which are necessary for operation in the current stage of the simulation.

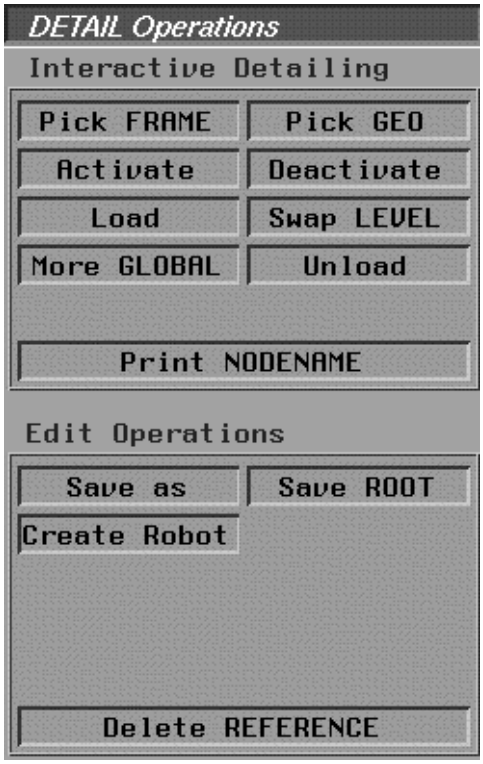


Figure 5: The Detail Panel

The following section explains the different commands in the "DETAIL" Submenu.

- **Pick Frame** {Pick ABS_Node}

  This command is used to start a PICK operation to identify the ABSTRACT node for successive operations. An ABS-node is picked at the origin of its associated coordinate system as drawn in the workcell display. When an ABSTRACT is picked, all GEO-parts connected to this ABSTRACT are highlighted.

- **Pick GEO** {Pick ABS_NODE via GEO}

  This is another command to PICK an ABSTRACT. It is computationally more demanding as the previous command. Here you can pick an ABSTARCT by picking any GEO-part connected to this ABSTRACT, i.e. to all FRAMES in the ABSTRACT file.

  Activate {Activate ABS-Branch}

A previously deactivated ABSTRACT is activated for display. All GEO-parts connected to the reference FRAME of the ABSTRACT remain active for display (see 'Swap Level' command).

- **Deactivate** {Deactivate ABS-Branch}

  An active ABSTRACT branch is deactivated for display. This command is useful to increase drawing speed (frame rate) drastically in the simulation. You should deactivate all model parts which are currently not essential for the operation. The ABSTRACT branch is not removed from the runtime database (main memory). Therefor it can be reactivated immediately for display by an 'Activate' operation.

- **Load** {Load ABS-Branch}

  Another ABSTRACT branch is loaded from the model database and activated for display. The memory required for model storage is allocated from the operating system. The function is used to load a model part or detail level which is marked in the database ('.mpc-file') with the 'W' attribute (wait), or to reload parts which were deleted previously in the same simulation session.

- **Swap LEVEL** {Swap DETAIL Level}

  This command loads/activates the next detail level (more detailed) for the picked ABS-Branch. At the same time, all GEO-parts connected to the reference frame of the ABSTRACT are deactivated for display. This is, the current GEOMETRY of the ABSTRACT-frame is replaced by a more detailed alternative geometry, which is defined in the loaded ABSTRACT-file ('.mpc file').

- **More GLOBAL** {More Details}

  This command is used to increase the detail level for the whole application model by one. The command works similar like a set of 'Activate' or 'Load'/'Swap LEVEL' commands. Only, it works with the whole model. It is mainly for "lazy" operators. Using this function, there is no fine control concerning which parts to load/activate/swap or not. You may find for some models, that the performance is reduced drastically after using this command. This can happen. when to many parts (more than expected) are activated.

- **Save as** {Write ABS_FILE}

  This function is used to save changes in the model to the file system. After picking an ABSTRACT (and the corresponding '.mpc file) using the 'Pick' Frame' or 'Pick GEO' command, the identified ABSTRACT is written to the database. Since an mpc file can have file references to other mpc-files and one '.dof'-file, the user has different options:

Figure 6: The Save-as Panel

– **Update dof-file:**

  The referenced dof file is checked and updated if necessary. KISMET checks, whether each moveable frame in the picked mpc-file is referenced in the dof-file or not. If a moveable frame is not referenced, the dof-file will be updated using the following default values (for frame No. i):

```
<menu_text> ::= "name_of_frame"_DOF
<outp_text> ::= Z_i
<n_entries> ::= 1
<type> ::= 0
n_const ::= 1
<const> ::= 0.1
```

(see User Data Specification).

– **Update mpc-file**

The referenced mpc-file will be updated.

– **Update both**

The referenced dof and mpc file will be updated

– **Update none**

Only the picked mpc-file will be updated, none of the referenced mpc- or dof-file will be updated.

- **Save ROOT** {Write ABS-Root File}

The ABSTRACT-root, i.e. the uppermost ’.mpc’ file in the application model, is written to the file system.

- **Unload** {Delete ABS-Branch}

Deletes an ABSTRACT-branch from the runtime database (main memory): The reference in the current model is not changed or deleted. This is the abstract can be reloaded later in the simulation session. The function is useful to make space in the main model for other parts, for example to avoid memory swapping. It is also used to increase drawing performance.

- **Delete REFERENCE** {Delete ABS-Reference}

Removes a REFERENCE to an ABSTRACT-file from the specified (picked) node. From this moment, the ABSTRACT file (detail or assembly) cannot be loaded or activated in the current simulation session. The abstract file in the database is not affected by this operation, because it may be referred to in other application models or parts of the current model (instancing).

- **Print NODENAME** {Print Nodename}

the fully qualified nodename of the ABSTRACT is printed in the KISMET-textport. The nodename is a list of all FRAME-names from the model-root (ABS-ROOT) to the selected ABSTRACT. The abstract names are separated by a ’.’ (dot). The full qualified nodename is used in SCRIPT-files to uniquely identify a FRAME or ABSTRACT node. This naming convention is necessary, because KISMET allows instancing of GEO-parts, FRAMEs and ABSTRACT-files.

- **Create Robot**

A new model is created: Starting with the .sim-File and leading the user through the complete data structure a new model will be created. This command can be used for example to give an existing model a new name after changes have been made. If no dof-file is available in the current model, a dof-file will be generated automatically.

Usage:

Create robot opens the "Enter sim file name" panel:

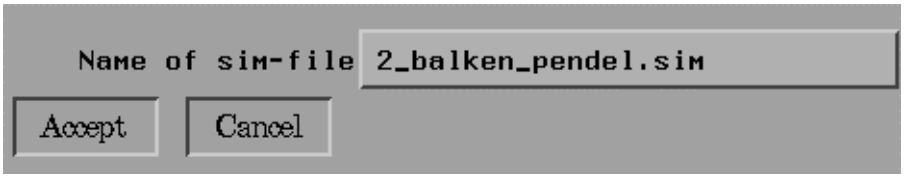


Figure 7: The Enter-Sim-file Panel

The next step will lead you to the Enter mpc-base filename Panel:

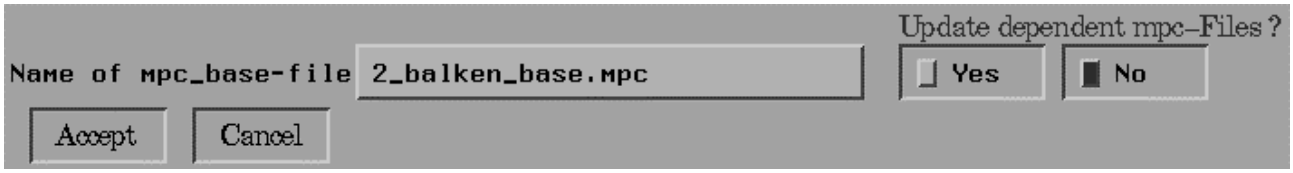


Figure 8: The "Enter base_mpc filename" panel

Finally the menu gives you the possibility to update and save all mpc-files of the current model. The "Save as" Panel will appear (see Figure 6: The Save-as Panel).

5 MBS-Dynamics - List of Commands

5.1 Overview

Kismet simulates the so called *"Inverse-"* or *"Indirect-Dynamic"* behaviour of a kinematic structure as well as the so called *"Direct Dynamics"*. *Inverse Dynamics* means to calculate the necessary torques $\tau(t)$ at the joints in order to get the desired behaviour of the joint variables $q(t)$, i.e. the motion. *Direct Dynamics* means to calculate the behaviour of the joint variables $q(t)$ due to given torques $\tau(t)$.

The *Inverse Dynamics* feature enables the user to calculate and display the torques and forces at joint level during execution of a robot teachfile or robot program. Pendulums are simple examples for the *Direct Dynamics* feature of KISMET. If you want to simulate the *"direct dynamics"* behaviour of robots, use the *"Control"* menu, which is responsible for the simulation of the dynamic behaviour of a mechanism combined with control algorithms enabling the user to specify $\tau(t)$.

	Direct Dynamics	Inverse Dynamics
Kinematic structures with no active elements (e.g. pendulums)	Dynamics Sub-menu	Dynamics Sub-menu
Kinematic structures with active elements (e.g. robots)	Control Sub-menu	Dynamics Sub-menu

Table 1: Sub-menu Selection

5.2 Dynamics Main Menu

The following sub-menus are provided by the MBS-Dynamics panel:

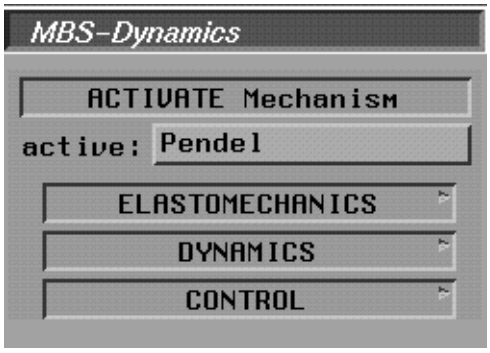


Figure 9: MBS-Dynamics panel

Explanation:

- **Elastomechanics**
  Leads to the Elastomechanics Sub-menu (see Chapter 5.2.1)
- **Dynamics**
  Leads to the Dynamics Sub-menu (see Chapter 5.2.2)
- **Control**
  Leads to the Control Sub-menu (see Chapter 5.2.3)

5.2.1 The Elastomechanics Panel

The following menus and functions are provided by the *Elastomechanics* panel:

Figure 10: Elastomechanics Panel

```
Enter Gravity (Base_frame):
Gx =
Gy =
Gz =
```

Edit Gravity:

Input of gravity direction and acceleration with reference to the base frame. Default values: $G_x = 0\ m/s^2, G_y = 9.810621\ m/s^2, G_z = 0\ m/s^2$

```
Enter external Forces (Base
Frame):
fx =
fy =
fz =
mx =
my =
mz =
```

Edit external Forces:

Input of external forces and torques acting on the last created "Deflected Joint Frame".

```
Change at Joint 1 (y/n)?
Actual  Value:  0.4567  New
Value:
```

Enter Deflection VALUES:

Input of static deflections of the "Deflected Joint Frames"

```
Rotational scale:
Old Value: 1.0 New Value:
Translational Scale:
Old Value New Value:
```

Scale Deflections:

Input of the deflection scaling factor. Enables the magnifying of the deflections.

```
Do you want to use an
existing data
set (y/n)?
comment:
type:
area:
.
.
.
```

Enter Stiffness:

Creation of a new stiffness matrix file by either typing in the values or by modification of an existing ".stf"-file.

```
Enter Joint Data:
Kx =
Ky =
Kz =
```

Enter JOINT Data:

Input of the spring constants describing the flexible behaviour of the joints.

Set Limits:

Determination of the greatest absolute value of a stiffness matrix file. Can be useful to determine a scaling factor.

Show Deflections:

The elastostatic behaviour of a mechanism will be calculated and visualised on the screen. Fastkey: "L"

Diagnostic FILE:

Saves a diagnostic file containing all stiffness Matrices, forces and torques of the current model. Filename: "defl_diag_out".

5.2.2 Dynamics Sub-menu

The following functions and menus are provided by the *Dynamics* sub-menu:

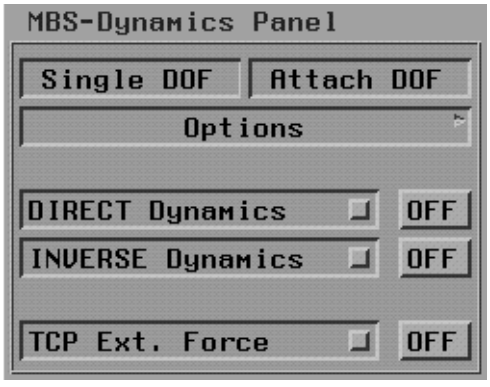


Figure 11: The Dynamics Panel

Options:

Leads to the Options sub-menu (see Chapter 5.2.2.1: Options sub-menu).

Direct Dynamics [On/Off]:

Start (or Stop) of the real-time simulation of the dynamic behaviour of all mechanisms in the current model that have been enabled for this purpose. Default parameters such as simulation time, step

size, etc. are defined in the ".kismetrc"-file and can be changed using the Options sub-menu "Initial Conditions" (see Chapter 5.2.2.1: Options sub-menu).

Inverse Dynamics [On/Off]:

Starts the Inverse Dynamics feature. According to the options defined in the Options sub-menu torques and/or forces will be displayed. There are basically 3 main applications for this menu:

1. Using *Inverse Dynamics* together with one of the *"Motion"* sub-menu: according to the configuration of the kinematic structure the static forces and torques will be displayed.
2. Using *Inverse Dynamics* together with the "Direct Dynamics On" sub-menu: the forces and torques in the non-static dynamic case will be displayed.
3. Using *Inverse Dynamics* together with the *"Run teachfile"* sub-menu: during the execution of a robot teachfile the forces and torques necessary to perform the demanded motion will be displayed. Usage: 1.) "Inverse Dynamics Start" 2.) "Run teachfile"

TCP Ext. Force:

Visualisation of the forces resulting from the contact of a mechanism with an elastodynamic object. Furthermore the resulting forces and torques in all joints of the colliding mechanism will be calculated and visualised if demanded.

5.2.2.1 Options sub-menu

The following commands and sub-menus are provided :

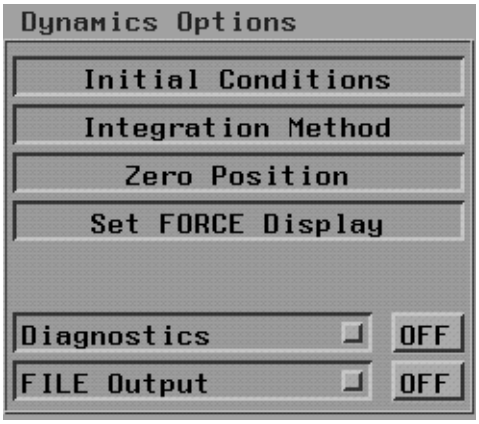


Figure 12: Dynamics Options

Initial Conditions:

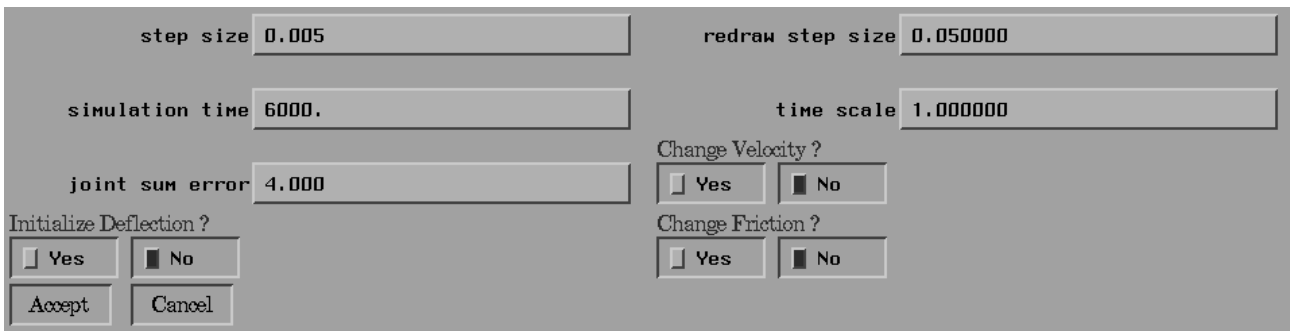


Figure 13: Initial Conditions

step size: Definition of integration step size

redraw step size: influences the relationship of CPU-time provided for drawing routines and CPU-time spent for calculation of the dynamic behaviour.

simulation time: self explainable, isn't it?

time scale: Defines a "slow motion" factor. time_scale = 10 means that a for a simulation time of 6 sec the simulation will be 10 times slower than real-time and therefor will last 60 seconds.

joint sum error: Normally the moment of inertia matrix will be calculated and inverted each time step, which is very CPU-time consuming. For slow moving objects it might not be necessary to calculate this matrix every time step. For a joint_sum_error > 0 the calculation of the inertia matrix will take place if

$$\sum_{i=0}^{n}\left(j_{i,new} - j_{i,old}\right) > jo\,\text{int}\_sum\_error$$

$$j_{i,new} = current\ \ jo\,\text{int}\ value\ of\ \ jo\,\text{int}\ i$$

$$j_{i,old} = value\ of\ \ jo\,\text{int}\ i\ during\ last\ matrix\ update$$

$$n = \text{deg}\,ree\ of\ \ freedom$$

Change velocity ?: The default velocity of all joints is set to 0 at the beginning of the dynamic simulation. Use Change_Velocity to change the initial velocity of all joints.

Initialise Deflection? If your dynamic model contains flexible joints you can change the initial deflection of the joint and it's corresponding link. The default value is set to *0*.

Change friction: Changes the friction parameters.

Integration Method: Defines the integration method: Runge-Kutta, Polygon, Heun, Craig or Adams-Bashford. Default: Polygon

Zero Position: If your model contains flexible joints you can change the initial deflection of the joint and it's corresponding link by defining a new zero position of your joint.

Set Force Display:

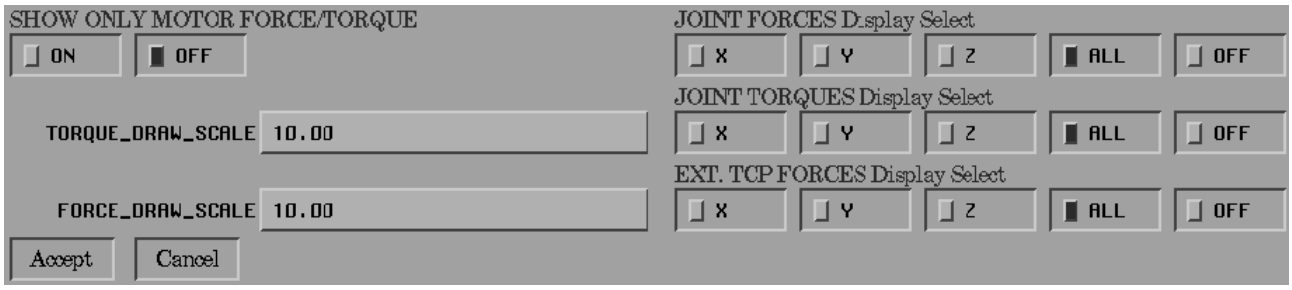


Figure 14: Set Force Display

Visualisation of torques and forces as arrows. You can display all components of the force/torque vectors or just the x-, z, or z- component. "Show Only Motor Force/Torque" is very useful in combination with the "TCP Ext. Force" feature of the Dynamics-Sub-menu (see Chapter 5.2.2 Dynamics Sub-menu). Only the components that correspond to the motor torques / forces are displayed (z-component of the torque for rotational DOF, z-component of the force for translational DOF). This is for example very useful for force feedback applications where these torque components can be used to control motors. The default values can be found in the ".kismetrc"-file in the current kis_home directory.

Diagnostics: Only for development purpose.

FILE Output: Output of different user definable values into a file during a dynamic simulation. This function contains 2 defined configurations with default values specially chosen for Direct- and Inverse- Dynamics and one function enabling the user to define his own configuration with variables of interest.

Figure 15: File Output Panel

Status [On/Off]: Defines whether the chosen variables will be written out into a file during the dynamic simulation or not.

Dir. Dyn.: *s(t)* will be saved in a file.

Ind. Dyn.: s(t), v(t), a(t) and τ(t) will be saved in a file.

User Def.: A user defined set of variables will be saved in a file. The user can select a set of variables created and saved earlier or create a new file (".diag"-file).

Usage: After selecting User Def a file selection menu displaying already existing ".diag". files will be shown. The user can

a) select a file and leave the selection menu with *Accept* or

b) the user can leave the selection menu with *Cancel* in order to create a new *".diag"* file. The following variable select menu will appear:

Figure 16: Variable select menu

After leaving the variable select menu the file select menu will appear again.

5.2.3 The Control Panel

Figure 17: The Control Panel

POSITION Control, VELOCITY Control, USER defined Control:

Selection of the desired type of control system. **Position-** and **Velocity Control** uses a default PID-control system which can be configured as P-, PI, or of course PID- control system by using the **Controllers** submenu. See chapter 5.2.3.1 for details.

User Defined Control:

Uses the control system defined by the ".csf" files. See chapter 5.2.3.1 for details.

Single DOF/Attach DOF, Constant POS/VEL:

Input of nominal values. The user can either specify a constant nominal value by using *Constant POS/VEL* or non-constant values .

Non constant nominal values can be changed by using the *Single DOF/Attach DOF* command. The user can associate a specific joint to one of the mouse buttons (left- or middle- mouse-button). The velocity- or position- nominal-value of the specified joint will be changed by moving the mouse. Both, the nominal values and the current joint values will be displayed on the screen in the upper right corner. To select this feature press **Single DOF** first followed by **Attach DOF**. Use the right mouse button to get a list of all available joints.

Controllers: Leads to the Controllers sub-menu. See chapter 5.2.3.1 for details

Initial Conditions: Leads to the Initial Conditions sub-menu. See chapter 5.2.2.1, Options sub-menu, for details.

Integration method: Selection of the integration method. See chapter 5.2.2.1, Options sub-menu, for details.

Run Control: Starts the simulation of the dynamic behaviour of the mechanism and it's control system.

Diagnostics:

Velocity Limits:

Control Limits:

5.2.3.1 The Controllers Panel

The *Position Control/Velocity* Control option uses a default PID-control system. With this sub-menu the user can

- select a P-, PI- or PID-control system
- change the default control parameters.

After selection of either Position Control or Velocity Control the default values are loaded by reading the ".con" files from the directory "kis_home/mpdlib". The filenames consist of the name of the *Center-of Gravity frame* (type 60) and the suffix ".con". If no files are available the user has to define them via textport.

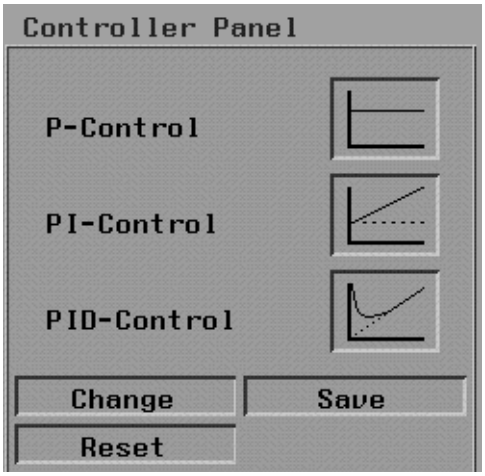


Figure 18: The Controller Panel

P-Control:

Automatic loading of the P-Control Parameter K_r or input via textport for each Joint:

```
Joint(Example_1)Actual Value Kr = 1   New Value:
```

PI-Control:

Automatic loading of the PI-Control parameters K_r and T_i for each joint or input via textport:

```
Joint(Example_1)Actual Value Kr = 1   New Value:
Joint(Example_1)Actual Value Ti = 1   New Value
```

PID-Control:

Automatic loading of the PI-Control parameters K_r, T_i, T_d and T_v for each joint or input via textport:

```
Joint(Example_1)Actual Value Kr = 1   New Value:
Joint(Example_1)Actual Value Ti = 1   New Value
Joint(Example_1)Actual Value Td = 1   New Value:
Joint(Example_1)Actual Value Tv = 1   New Value:
```

Change:

Select the type of controller you want to change and enter the parameters as described above.

Save:

Saves the control parameters for your selected type of controller.

Reset:

Resets the parameters of the selected type of controller to default values (if available).

6 Data Specification

6.1 The "Dynamics-Definition"-Files (.def)

6.1.1 Syntax of "Dynamics-Definition"-Files

These files are created during frame creation (frame-type 60 or 61) and are loaded automatically. The file can be found in the directory "kis_home/mpdlib". The name of a ".def"-file consists of the name of the corresponding frame and the suffix ".def".

The ".def"-Files describe certain parameters of one link of a mechanism which are mandatory for beam-model based "Elastomechanics-" or "Multibody System Dynamics" simulation.

```
<DEFORMATIONS_file>   =     DEF_FILE   ::=
     <comment>              ::=  char[80]
     <mpd-Typ>              ::=  int
                                 1|2|3|5|6|7|8
     <DEF_values>           ::=  <LINK> | <JOINT>
END_DEF_FILE


<LINK> = LINK_DATA    ::=
     <DEF_VALUES> ::=double • 13
                           <area>, <length>, <E_mod>,
                           <J_mod_x>, <J_mod_y>, <J_mod_z>,
                           <G_mod>, <scale>, <mass>, <friction>,
                           <motor_stiffness>, <motor_damping>,
                      <motor_inertia>
END_LONK_DATA

<Joint> = JOINT_DATA ::=
     <DEF_VALUES>     ::=  double • 8
                           <length>, <mass>, <joint_x>,
                           <joint_y>, <joint_z>,
                           <scale>,   <future_a>, <future_b>
END_JOINT_DATA
```

Figure 19: Syntax of ".def"-Files

comment Comment string.

mpdtyp Definition of type of Dynamics-Definition file

- **1** File can be used for Elastomechanics and Dynamics purpose. The corresponding element has a beam like shape. The stiffness matrix of the corresponding element (frame type 60 or 61) will be generated automatically.
- **2** File can be used for Elastomechanics and Dynamics purpose. The corresponding element has a shape that cannot be considered as beam like. Therefor the stiffness matrix will not be generated automatically.
- **3** File can be used for Elastomechanics and Dynamics purpose. The file

describes a joint element. The stiffness matrix will be generated automatically.

4 **Not allowed**!!

5 File can be used for Dynamics purpose only!

6 File can be used for Elastomechanics purpose only. The corresponding element has a beam like shape. The stiffness matrix of the corresponding element (frame type 60 or 61) will be generated automatically

7 File can be used for Elastomechanics purpose only. The corresponding element has a shape that cannot be considered as beam like. Therefor the stiffness matrix will not be generated automatically

8 File can be used for Elastomechanics purpose only. The file describes a joint element. The stiffness matrix will be generated automatically.

area Cross-section area of a beam shaped element in [*mm*]

length Length of a beam shaped element or distance between the appropriate joint frame (frame type 0 or 1) and it's kinematic predecessor in [*mm*]

E_mod Modulus of elasticity (Young's modulus) in [N/mm^2]

J_mod_x Area moment of inertia about the x-axis in [N/mm^4]

J_mod__y Area moment of inertia about the y-axis in [N/mm^4]

J_mod_z Area moment of inertia about the z-axis in [N/mm^4]

G_mod Shearing modulus in [N/mm^2]

scale Scaling factor for stiffness matrices

mass mass in [*kg]*

friction Friction coefficient of the corresponding joint

motor_stiffness Spring coefficient for Dynamics and Control purpose in order to describe the coupling of a motor/actuator and the corresponding link in [N/grad] for rotational joints or [N/mm] for translational joints.

motor_damping Damping coefficient for *Dynamics* and *Control* applications. motor_damping describes the coupling of the motor/actuator with it's corresponding link in [*Nsec/mm*].

motor_inertia Motor/actuator moment of inertia in [*Nmm/sec*]

joint_x Joint-stiffness in x-direction in [*Nmm/grad*]

joint_y Joint-stiffness in y-direction in [*Nmm/grad*]

joint_z Joint-stiffness in z-direction in [*Nmm/grad*]

future_a For future applications only!

future_b For future applications only!

6.1.2 Dynamics-Definition-File Example

The following example of an Dynamics-Definition File defines a link of a pendulum:

```
balken_1
5
1.759300e+04
1.000000e+03
7.200000e+04
3.465805e+08
1.732903e+08
1.732903e+08
2.727200e+04
1.000000e+05
1.000000e+01
0.000000000e+00
0.000000e+00
0.000000e+00
0.000000e+00
```

6.2 The "Inertia"-Files (.ita)

6.2.1 Syntax of "inertia"-Files

These files are created during frame creation (typ 60) and are loaded automatically. The files can be found in the directory "kis_home/mpdlib". The name of a ".ita"-file consists of the name of the corresponding frame and the suffix ".ita".

The files contain the value of the inertia matrix of the corresponding link with reference to its center of gravity.

```
<INERTIA_file> = INERT_FILE ::=
     <comment>  ::   char[80]
     <inertia>  ::   double • 9
END_INERT_FILE
```

Figure 20: Syntax of =.ita"-files

comment Comment string

inertia Inertia values of the 3x3 inertia matrix in [*Nmm/s*]

6.2.2 Inertia-File Example

The following example defines the inertia matrix of one link of a pendulum.

```
link_1
1.666666e+06
0.000000e+00
0.000000e+00
0.000000e+00
8.333333e+05
0.000000e+00
0.000000e+00
0.000000e+00
8.333333e+05
```

6.3 The "Stiffness"-Files (.stf)

6.3.1 Syntax of "stiffness"-Files

These files are created during frame creation (type 60 and 61) and are loaded automatically. The files can be found in the directory "kis_home/mpdlib". The name of a ".stf"-file consists of the name of the corresponding frame and the suffix ".stf".

The files contain the value of the stiffness matrix of the corresponding link with reference to its center of gravity.

```
<STIFFNESS_file> = STIFF_FILE ::=
     <comment>  ::=  char[80]
     <scale>    ::=  int
     <stiffness>     ::=  double • 36
END_STIFF_FILE
```

Figure 21: Syntax of ".stf"-File

comment Comment string

scale Scaling factor

stiffness Values of the 6x6 stiffness matrix with reference to the center of gravity of the corresponding link in

$$\left[\begin{array}{ccc|ccc} N/mm & N/mm & N/mm & N/rad & N/rad & N/rad \\ N/mm & N/mm & N/mm & N/rad & N/rad & N/rad \\ N/mm & N/mm & N/mm & N/rad & N/rad & N/rad \\ \hline N & N & N & Nmm/rad & Nmm/rad & Nmm/rad \\ N & N & N & Nmm/rad & Nmm/rad & Nmm/rad \\ N & N & N & Nmm/rad & Nmm/rad & Nmm/rad \end{array}\right]$$

6.4 The "Simple Control System"-File (.con)

6.4.1 Syntax of "Simple Control System"-Files

These files belong to frames of type 60 or 61 and are loaded automatically if available in the current model. The files can be found in the directory "kis_home/mpdlib". The name of a ".stf"-file consists of the name of the corresponding frame and the suffix ".stf". The files contain the specification and parameters of a simple control system for one corresponding joint as described in Figure 22.

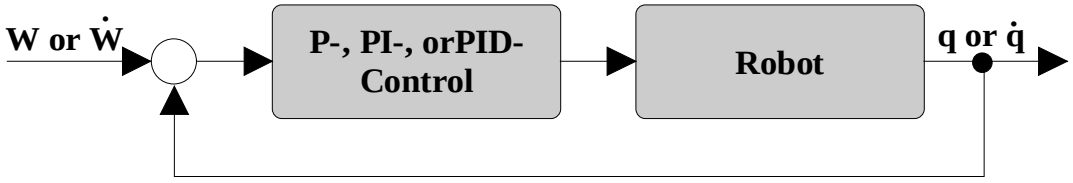


Figure 22: Simple Control System

The syntax of the simple control system files is listed in Figure 23:

```
<Control_file> = CONTROL_FILE   ::=
     <comment>       ::=  char[80]
     <p_control>     ::=  float
                          <Kr>
     <pi_control>    ::=  float • 2
                          <Kr>, <Ti>
     <pid_control>   ::=  float • 4
                          <Kr>,  <Ti>, <Td>, <Tv>
END_CONTROL_FILE
```

Figure 23: Syntax of ".con"-file

comment	Comment string
p_con	Parameter for a P-Controller
pi_con	Parameter for a Pi-Controller
pid_con	Parameter for a PID-controller

6.4.2 Example for a "Simple-Control-File"

The following example defines the Parameter for a P-, PI-, and PID-Controller for one joint of a robot.

```
Control_parameter
1.000000
1.000000
1.000000
5.000000
12.00000
0.100000
0.200000
```

6.5 The "Complex Control System"-Files (.csf)

6.5.1 Syntax of "Complex Control System"-Files

These files belong to frames of type 60 or 61 and are loaded if the **User Defined Control** option has been chosen at the **Control Panel** (see Chapter 5.2.3). The files can be found under "kis_home/mpdlib". The names consist of the names of the corresponding frames and the suffix ".csf". For each link a different control system can be defined and saved in a ".csf" file.

```
<COMPLEX_CONTROL_file> = CONTROL_FILE ::=
     <comment>             ::=  char[80]
     <block_number>  ::=  int
     <CONTROL_BODY>        ::=  <block_primitive> • <block_number>
END_COMPLEX_CONTROL_FILE


<block_primitive>          ::=  P_BLOCK | P_TEMP | INT |
                                D_BLOCK | PI_BLOCK | PD_BLOCK
                                PID_BLOCK | SUM | LIMIT |
                                RAMP | SWITCH | NOM_TYPE
```

Figure 24: Syntax of ".csf"-files

comment comment string

number_of_blocks Number of control primitives (=block primitives)

CONTROL_BODY Structure of the control system consisting of different kinds of block primitives. See chapter 6.5.1.1 - 6.5.1.12 for details

- **P_BLOCK** Block primitive with P-behaviour. See chapter 6.5.1.1.
- **P_TEMP** Block primitive with P-behaviour for a certain time. See chapter 6.5.1.2.
- **INT** Block primitive with I-behaviour. See chapter 6.5.1.3
- **D_BLOCK** Block primitive with differentiating behaviour. See chapter 6.5.1.4
- **PI_BLOCK** Block primitive with PI-behaviour. See chapter 6.5.1.5.
- **PD_BLOCK** Block primitive with PD-behaviour. See chapter 6.5.1.6.
- **PID_BLOCK** Block primitive with PID-behaviour. See chapter 6.5.1.7.
- **SUM** Block primitive with "adding up" behaviour. See chapter 6.5.1.8
- **LIMIT** Block primitive with limiting behaviour. See chapter 6.5.1.9
- **RAMP** Block primitive with "ramp"- behaviour. See chapter 6.5.1.10.
- **SWITCH** Special block primitive for changing the control system during simulation. See chapter 6.5.1.11.

NOM_TYPE Special block primitive for changing the type of nominal value. Default type of nominal value: τ . Can be changed to $\dot{\omega}$ if NOM_TYPE is used and the corresponding parameter is set to 1. See chapter 6.5.1.12 for details.

6.5.1.1 Syntax of the P_BLOCK primitive

```
P_BLOCK    ::=
     <comment>        ::=  char[80]
     <id>             ::=  p
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <p_param>        ::=  float
                           <Kr>
END_P_BLOCK


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 25: Syntax of the P_BLOCK Primitive

comment	comment string	
id	identifies this block as P_BLOCK primitive	
input	identifies the type of input for this block	
	nominal_pos	input is the nominal position of the corresponding joint
	nominal_vel	input is the nominal velocity of the corresponding joint
	back_pos	input is the actual position of the corresponding joint
	back_vel	input is the actual velocity of the corresponding link
	<block>	input is the output of another block primitive
p_param	Parameter of the P_BLOCK primitive: Kr	
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.	

6.5.1.2 Syntax of the P_TEMP primitive

The P_TEMP primitive behaves like a P_BLOCK primitive but only for a certain period of time defined by the second parameter <T>. After this period of time the output of the P_TEMP primitive is set to 0.

```
P_TEMP     ::=
     <comment>        ::=  char[80]
     <id>             ::=  p_temp
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <p_param>        ::=  float • 2
                           <Kr>, <T>
END_P_TEMP


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 26: Syntax of the P_TEMP Primitive

comment	comment string	
id	identifies this block as P_TEMP primitive	
input	identifies the type of input for this block	
	nominal_pos	input is the nominal position of the corresponding joint
	nominal_vel	input is the nominal velocity of the corresponding joint
	back_pos	input is the actual position of the corresponding joint
	back_vel	input is the actual velocity of the corresponding link
	<block>	input is the output of another block primitive
p_param	Parameter of the P_BLOCK primitive: Kr, T	
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.	

6.5.1.3 Syntax of the INT-Primitive

```
INT        ::=
     <comment>        ::=  char[80]
     <id>             ::=  int
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <int_param>           ::=  float
                           <Ti>
END_INT


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 27: Syntax of the INT Primitive

comment	comment string
id	identifies this block as INT primitive
input	identifies the type of input for this block
	nominal_pos input is the nominal position of the corresponding joint
	nominal_vel input is the nominal velocity of the corresponding joint
	back_pos input is the actual position of the corresponding joint
	back_vel input is the actual velocity of the corresponding link
	<block> input is the output of another block primitive
int_param	Parameter of the INT primitive: Ti
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

6.5.1.4 Syntax of the D_BLOCK primitive

```
D_BLOCK          ::=
     <comment>        ::=  char[80]
     <id>             ::=  d
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <int_param>           ::=  float • 2
                           <Kr>, <Tv>
END_D_BLOCK


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 28: Syntax of the D_BLOCK primitive

comment	comment string
id	identifies this block as D_BLOCK primitive
input	identifies the type of input for this block
	nominal_pos input is the nominal position of the corresponding joint
	nominal_vel input is the nominal velocity of the corresponding joint
	back_pos input is the actual position of the corresponding joint
	back_vel input is the actual velocity of the corresponding link
	<block> input is the output of another block primitive
int_param	Parameter of the D_BLOCK <primitive: Kr, Tv
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

6.5.1.5 Syntax of the PI_BLOCK primitive

```
PI_BLOCK        ::=
     <comment>       ::=  char[80]
     <id>            ::=  pi
     <input>         ::=  nominal_pos | nominal_vel |
                          back_pos | back_vel |
                          <block>
     <pi_param>      ::=  float • 2
                          <Kr>, <Ti>
END_PI_BLOCK


<block>         ::=  block <ident>
<ident>         ::=  int
```

Figure 29: Syntax of the PI_BLOCK primitive

comment	comment string	
id	identifies this block as PI_BLOCK primitive	
input	identifies the type of input for this block	
	nominal_pos	input is the nominal position of the corresponding joint
	nominal_vel	input is the nominal velocity of the corresponding joint
	back_pos	input is the actual position of the corresponding joint
	back_vel	input is the actual velocity of the corresponding link
	<**block**>	input is the output of another block primitive
pi_param	Parameter of the PI_BLOCK primitive: Kr, Ti	
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.	

6.5.1.6 Syntax of the PD_Block primitive

```
PD_BLOCK        ::=
     <comment>       ::=  char[80]
     <id>            ::=  pd
     <input>         ::=  nominal_pos | nominal_vel |
                          back_pos | back_vel |
                          <block>
     <pd_param>      ::=  float • 3
                          <Kr>, <Td>, <Tv>
END_PD_BLOCK


<block>         ::=  block <ident>
<ident>         ::=  int
```

Figure 30: Syntax of the PD_BLOCK primitive

comment	comment string	
id	identifies this block as PD_BLOCK primitive	
input	identifies the type of input for this block	
	nominal_pos	input is the nominal position of the corresponding joint
	nominal_vel	input is the nominal velocity of the corresponding joint
	back_pos	input is the actual position of the corresponding joint
	back_vel	input is the actual velocity of the corresponding link
	<**block**>	input is the output of another block primitive
pid_param	Parameter of the PD_BLOCK Primitive: Kr, Td, Tv	
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.	

6.5.1.7 Syntax of the PID-Block primitive

```
PID_BLOCK       ::=
     <comment>        ::=  char[80]
     <id>             ::=  pid
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <pid_param>           ::=  float • 4
                           <Kr>, <Ti>, <Td>, <Tv>
END_PID_BLOCK


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 31: Syntax of the PID_BLOCK primitive

comment	comment string	
id	identifies this block as PID_BLOCK primitive	
input	identifies the type of input for this block	
	nominal_pos	input is the nominal position of the corresponding joint
	nominal_vel	input is the nominal velocity of the corresponding joint
	back_pos	input is the actual position of the corresponding joint
	back_vel	input is the actual velocity of the corresponding link
	<**block**>	input is the output of another block primitive
pid_param	Parameter of the PID_BLOCK Primitive: Kr, Ti, Td, Tv	
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.	

6.5.1.8 Syntax of the SUM-primitive

```
SUM         ::=
     <comment>        ::=  char[80]
     <id>             ::=  sum
     <in_cnt>         ::=  int
     <in_typ> • in_cnt
     <sum_fact> • in_cnt
END_SUM


<in_typ>         ::=  nominal_pos | nominal_vel |back_pos |
                      back_vel | <block>
<sum_fact>       ::=  +1 | -1


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 32: Syntax of the SUM_BLOCK primitive

comment	comment string
id	identifies this block as SUM primitive
in_cnt	Number of inputs of the SUM Primitive
in_typ	Identifies the type of input for this SUM primitive
	nominal_pos input is the nominal position of the corresponding joint
	nominal_vel input is the nominal velocity of the corresponding joint
	back_pos input is the actual position of the corresponding joint
	back_vel input is the actual velocity of the corresponding link
	<block> input is the output of another block primitive
sum_fact	Parameter of the SUM primitive: (+1 = Addition; -1 = Subtraction)
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

6.5.1.9 Syntax of the LIM Primitive

```
LIM        ::=
     <comment>        ::=  char[80]
     <id>             ::=  lim
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <lim_param>           ::=  float • 2
                           <min>, <max>
END_LIM


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 33: Syntax of the PID_BLOCK Primitive

comment comment string

id identifies this block as LIM primitive

input identifies the typ of input for this block

- **nominal_pos** input is the nominal position of the corresponding joint
- **nominal_vel** input is the nominal velocity of the corresponding joint
- **back_pos** input is the actual position of the corresponding joint
- **back_vel** input is the actual velocity of the corresponding link
- **>block>** input is the output of another block primitive

lim_param Parameter of the LIM Primitive:

- **min** Lower limit
- **max** Upper limit

ident identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

6.5.1.10 Syntax of the RAMP-primitive

```
RAMP       ::=
     <comment>        ::=  char[80]
     <id>             ::=  ramp
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <int_param>           ::=  float
                           <G>
END_RAMP


<block>          ::=  block <ident>
<ident>          ::=  int
```

Figure 34: Syntax of the RAMP primitive

comment	comment string
id	identifies this block as RAMP primitive
input	identifies the typ of input for this block
	nominal_pos input is the nominal position of the corresponding joint
	nominal_vel input is the nominal velocity of the corresponding joint
	back_pos input is the actual position of the corresponding joint
	back_vel input is the actual velocity of the corresponding link
	<block> input is the output of another block primitive
int_param	Parameter of the RAMP primitive: G = gradient of the "ramp"
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

6.5.1.11 Syntax of the SWITCH-primitive

The SWITCH primitive enables the user to change the structure of the control system during dynamic simulation. The SWITCH primitive checks weather:

- *input < min (switch position = 0)*
- *min <= input <= max (switch position = 1)*
- *input > max (switch position = 2)*

For each of the 3 possible switch positions you can define dependent BLOCK primitives which are active if the input variable is inside the specified range and inactive else.

```
SWITCH          ::=
     <comment>        ::=  char[80]
     <id>             ::=  switch
     <input>          ::=  nominal_pos | nominal_vel |
                           back_pos | back_vel |
                           <block>
     <switch_param>   ::=  float • 2
                           <min>, <max>
     <start_pos>      ::=  0 | 1 |2
     <dependencies>   ::=  DEP • 3
END_SWITCH


<block>          ::=  block <ident>
<ident>          ::=  int

DEP   ::=
     <dep_cnt>   ::=  int
     <dep_ident> • dep_cnt>
END_DEP

dep_ident  ::=  int
```

Figure 35: Syntax of the SWITCH primitive

comment comment string

id identifies this block as SWITCH primitive

input identifies the type of input for this block

- **nominal_pos** input is the nominal position of the corresponding joint
- **nominal_vel** input is the nominal velocity of the corresponding joint
- **back_pos** input is the actual position of the corresponding joint
- **back_vel** input is the actual velocity of the corresponding link
- **>block>** input is the output of another block primitive

switch_param Parameter of the SWITCH Primitive:

- **min** Lower limit
- **max** Upper limit

start_pos Defines the default condition of the SWITCH block primitive. Defines which of

the dependent blocks are active and/or inactive.

ident identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

dep_cnt Number of dependent blocks for one of the 3 switch positions.

dep_ident identifies the dependent block primitives

6.5.1.12 Syntax of the NOM_TYPE primitive

```
NOM_TYPE   ::=
     <comment>       ::=  char[80]
     <id>            ::=  nom_type
     <input>         ::=  nominal_pos | nominal_vel |
                          back_pos | back_vel |
                          <block>
     <param>         ::=  int
                          0 | 1
END_NOM_TYPE


<block>         ::=  block <ident>
<ident>         ::=  int
```

Figure 36: Syntax of the NOM_TYPE Primitive

comment	comment string
id	identifies this block as NOM_TYPE primitive
input	identifies the typ of input for this block
	nominal_pos input is the nominal position of the corresponding joint
	nominal_vel input is the nominal velocity of the corresponding joint
	back_pos input is the actual position of the corresponding joint
	back_vel input is the actual velocity of the corresponding link
	<block> input is the output of another block primitive
param	Parameter of the NOM_TYPE primitive:
	0 Nominal value = τ
	1 Nominal value = $\dot{\omega}$
ident	identifies the block primitive whose output is the input of this block primitive. The parameter "ident" depends on the position of the block primitive in the ".csf"-file. The first block primitive has the identifier *ident = 1*, the second has *ident = 2* and so on.

6.5.2 Example of a ".csf-file"

Figure 37 shows a structure of a complex control system followed by the corresponding ".csf"-file.

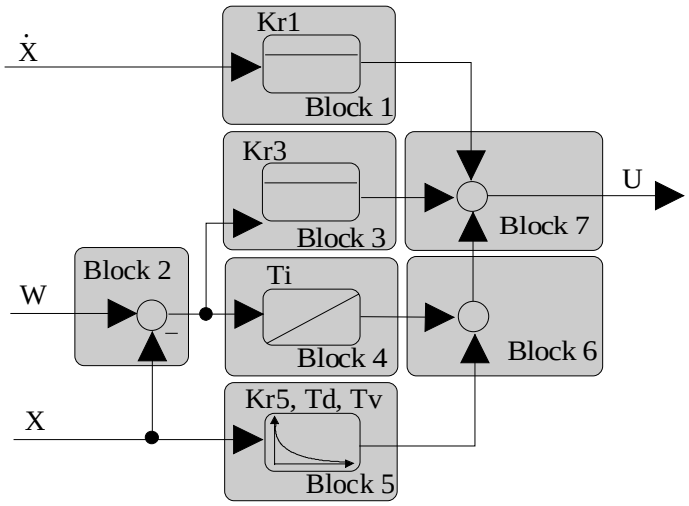


Figure 37: Complex Control System

```
Example_for_a_csf_file
7
*****BLOCK_1*****
p
nominal_vel
500
*****BLOCK_2*****
sum
2
nominal_pos
back_pos
1 -1
*****BLOCK_3*****
p
block 2
500
*****BLOCK_4*****
i
block 2
10
*****BLOCK_5*****
pd
back_pos
500 27
*****BLOCK_6*****
sum
2
block 4
block 5
1 1
*****BLOCK_7*****
sum
3
block 1
block 3
block 6
1 1 1
```

6.6 The "Diagram"-Files (.diag)

6.6.1 Introduction

Since KISMET Version 4.95.x a new syntax for the ".diag"-files has been introduced. The new syntax is to some extent self explainable enabling the user to change existing files with an editor. KISMET still understands the old syntax but creates only files with the new syntax from now on.

6.6.2 Syntax of the ".diag"files (old syntax)

```
<DIAGNOSE_file> ::=  <DIAG_FILE ::=
     <comment>  ::=  char[80]
     <n_joints> ::=  int
     <    DIAG_BODY> • n_joints
END_DIAG_FILE
```

Figure 38: Syntax of the ".diag"-file (old syntax)

```
<DIAG_BODY> ::= <DIAG_body> ::=

     <joint_name>    ::=  char[20]
     <bit_pattern>   ::=  bit_type • 40
                          <position>, <position_x>, <position_y>,
                          <position_z>, <velocity>, <acceleration>,
                          <force_x>, <force_y>, <force_z>, <force>,
                          <torque_x>, <torque_y>, <torque_z>,
                          <torque>,<link_defl_tx> <link_defl_ty>,
                          <link_def_tz>;<link_defl>,
                          <link_defl_rx>,<link_defl_ry>,
                          <link_defl_rz>, <control_in>,
                          <control_out>, <dummy>, <actuator_angle>,
                          <nominal_pos>,<control_block>,
                          <control_block>
     <bit_type> ::=  boolean
                     0 | 1
END_DIAG_BODY
```

Figure 39: DIAG_BODY syntax (old syntax)

comment Comment string

n_joints Number of joints

joint_name Joint name

bit_pattern Selects the values to be printed out into the selected file

- **position** Relative position of the selected joint with reference to its predecessor frame
- **position_x** X-component of the absolute position vector of the selected joint frame with reference to the base frame
- **position_y** Y-component of the absolute position vector of the selected joint frame with reference to the base frame
- **position_z** Z-component of the absolute position vector of the selected joint frame with reference to the base frame
- **velocity** Relative velocity of the selected joint with reference to its predecessor frame
- **acceleration** Relative acceleration of the selected joint with reference to its predecessor frame
- **force_x** X-component of the force vector acting on the selected joint
- **force_y** Y-component of the force vector acting on the selected joint
- **force_z** Z-component of the force vector acting on the selected joint
- **force** Absolute value of the force vector acting on the selected joint
- **torque_x** X-component of the torque vector acting on the selected joint
- **torque_y** Y-component of the torque vector acting on the selected joint
- **torque_z** Z-component of the torque vector acting on the selected joint
- **torque** Absolute value of the torque vector acting on the selected joint
- **link_defl_tx** X-component of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
- **link_defl_ty** Y-component of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
- **link_defl_tz** Z-component of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
- **link_defl** Absolute value of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
- **link_defl_rx** X-component of the vector describing the rotational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint

link_defl_ry Y-component of the vector describing the rotational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint

link_defl_tz Z-component of the vector describing the rotational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint

control_in input variable of the simple control system for the selected joint.

control_out output variable of the control system of the selected joint

actuator_angle Actual motor position

nominal_pos Nominal position of the selected joint

dummy for future use only!

bit_type print status of selected variable

1 corresponding variable will be saved

0 corresponding variable will not be saved

6.6.3 Example for a ".diag"-file (old syntax)

The following example defines only 2 values to be written out into a file during dynamic simulation: joint position and motor position.

```
Example_for_a_diag_file_old_synatx
2
Joint_1
10000000000000000000000010000000000000000
Joint_2
10000000000000000000000010000000000000000
```

Figure 40: Example of a ".diag"-file (old syntax)

6.6.4 Syntax of the ".diag"-files (new syntax)

```
<DIAGNOSE_file> ::=  <DIAG_FILE>     ::=
     <identifier>    ::=  Diagnose_File
     <ROBOT_NAME>    ::=  char[80]
     <n_joints>      ::=  int
     <JOINT_NAME> • n_joints
     <n_var>         ::=  int
     <VAR_NAME> • n_var
END_DIAG_FILE


JOINT_NAME ::=  char[80]
VAR_NAME   ::=  VARIABLE | BLOCK_VARIABLE

VARIABLE   ::=  char[30]
                _Position | _position_X | _position_Y |
                _position_Z |_Velocity | _Acceleration |
                _Force_X | _Force_Y | _Force_Z | _Force |
                _Torque_X | _Torque_Y | _Torque_Z |
                _Torque | _Link_Defl._Tx < _Link_Defl._Ty
                _Link_Defl._Tz | _Link_Defl. | _Link_Defl._Rx
                _Link_Defl._Ry | _Link_Defl._Rz |_Controler_In
                _Controler_Out | _actuator_angle | _Demanded_Pos

BLOCK_VARIABLE  ::=
     <block_slect>   ::=  char[30]
                          _Control_Block_a | _Control_Block_b
     <block_ident>   ::= int
END_BLOCK_VARIABLE
```

Figure 41: Syntax of ".diag"-file (new syntax)

identifier Necessary for KISMET to distinguish between old and new syntax

ROBOT_NAME Name of the selected mechanism/robot

n_joints Number of DOF of the selected robot

JOINT_NAME Name of the selected joint

n_var Number of variables of interest to be saved in a file during dynamic simulation

VAR_NAME Identifies the variables of interest

- **_Position** Relative position of the selected joint with reference to its predecessor frame
- **_position_X** X-component of the absolute position vector of the selected joint frame with reference to the base frame
- **_position_Y** Y-component of the absolute position vector of the selected joint frame with reference to the base frame
- **_position_Z** Z-component of the absolute position vector of the selected joint frame with reference to the base frame

_Velocity	Relative velocity of the selected joint with reference to its predecessor frame
_Acceleration	Relative acceleration of the selected joint with reference to its predecessor frame
_Force_X	X-component of the force vector acting on the selected joint
_Force_Y	Y-component of the force vector acting on the selected joint
_Force_Z	Z-component of the force vector acting on the selected joint
_Force	Absolute value of the force vector acting on the selected joint
_Torque_X	X-component of the torque vector acting on the selected joint
_Torque_Y	Y-component of the torque vector acting on the selected joint
_Torque_Z	Z-component of the torque vector acting on the selected joint
_Torque	Absolute value of the torque vector acting on the selected joint
_Link_defl._Tx	X-component of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Link_defl._Ty	Y-component of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Link_defl._Tz	Z-component of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Link_Defl.	Absolute value of the vector describing the translational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Link_defl._Rx	X-component of the vector describing the rotational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Link_defl._Ry	Y-component of the vector describing the rotational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Link_defl._Tz	Z-component of the vector describing the rotational deformation of the deflected joint frame (type 10) with reference to its predecessor frame, the selected joint
_Controler_In	input variable of the simple control system for the selected joint.
_Controler_out	output variable of the control system of the selected joint

_actuator_angle Actual motor position

_Demanded_Pos Nominal position of the selected joint

block_select Since KISMET Version 4.95.x the user can select 2 blocks of a complex control system (see Chapter 6.5) whose output will be saved during dynamic simulation. The 2 blocks are identified by _Control_Block_a and _Control_Block_b followed by the identification number "block_ident".

block_ident identifies the block primitive. The parameter "block_ident" depends on the position of the block in the "-.csf"-file. You refer to the first block defined in the ".csf"-file with "1", to the second with "2" and so on.

6.6.5 Example of a ".diag"-file (new syntax)

The following example of a ".diag" file defines the following 6 variables to be saved during dynamic simulation:

1) the position of joint "z_4"
2) the output of the control system responsible for joint "z_4"
3) the motor position
4) the nominal position
5) the output of control block 3 of the corresponding complex control system
6) the output of control block 11 of the corresponding complex control system

```
Diagnose_File
EDITH_B
1
z_4
6
_Position
_Controler_Out
_actuator_angle
_Demanded_Pos
_Control_Block_a 3
_Control_Block_b 11
```

Figure 42: Example of a ".diag"-file (new syntax)

6.7 Dynamics relevant Script Commands

SET_FORCE_ARROW <int_1> <int_2> <int_3>

Defines the parameter for the **Force Display** option (see Chapter 5.2.2.1).

<int-1>	[0 | 1 | 2 | 3 | 4] Parameters for the force arrows. 0 = Draw only the X-componet 0 = Draw only the Y-componet 0 = Draw only the Z-componet
<int_2>	[0 | 1 | 2 | 3 | 4] Parameters for the force arrows. (see <int_1>)
<int_3>	[0 | 1 | 2 | 3 | 4] Parameters for the TCP-force arrows. (see <int_1>)

SET_TORQUE_LIMIT <max_force> <max_torque>

Defines limits for forces and torques during interaction with elastodynamical objects (see **TCP-Ext.Force** command in chapter 5.2.2). If the forces/torques are violating the limits a warning will appear on the screen if

- the limits are $\neq 0$
- and the force/torque values are send to via Shared Memory (key 1313) to another process

SET_EXT_TCP_TEST [on | off] <value_1> <value_2> <value_3> >value_4>

Enables or disables the test mode for the **TCP-Ext.Force** command (see chapter 5.2.2). The specified values (<value_1>, ... ,<value_4>) will be send to Shared Memory (key 1313) if test mode is enabled.

SET_FRC_ARR_SPECIAL [on | off]

Correspond to the SHOW ONLY MOTOR FORCES/TORQUES button in the Set FORCE Display command in the DYNAMICS Option Panel (see chapter 5.2.2.1). Only the component of the force/torque vector that correspond to the motor torques/forces (Z-component of the torque vector for rotational DOFs and Z-component of the force vector for translational DOFs) will be visualized.

MASTER_SLAVE [on | off] <robot_id> <number_of_slaves>
<
<robot_id>,<number_of_dof>
<
<name_of_master_joint> <name_of_slave_joint>
> * number_of_dof
> * number_of_slaves

Enables/disables the Master-Slave option:

The first robot_id defines the master mechanism the others up to 5 the slaves mechanisms. The master DOFs will be connected to the specified slave DOFs. For identification the joint names defined in the mpc-files are used.