DINO v0.9 Reference Manual

www.dino3d.org

Manual Version 12th September 2003

Contents

List of Tables

1 Installation and Startup

1.1 Installation

DINO is distributed in binary form for several platforms that can be downloaded from the DINO homepage at `http://www.dino3d.org`:

- Linux i386 kernel 2.4+, `libc.so.6`
- IRIX 6.5+
- Digital OSF1 4.0+
- SunOS 5.7+
- MacOSX

DINO uses OpenGL (`http://www.opengl.org`) as its 3D library, therefore a working OpenGL implementation is required, which is present on most of todays workstations. If not, then a software-only variant called Mesa (`http://www.mesa3d.org`) can be used.

There are no special requirements for installation, the executable can be called from anywhere and it does not expect any other files.

1.2 Command Line Arguments

The following command line arguments are available:

- **`-debug`** print out lots of debugging info during execution
- **`-help`** displays usage
- **`-log LOGFILE`** writes all entered commands into specified logfile. This defaults to `logfile.dino`
- **`-nolog`** does not write a logfile
- **`-noom`** the object menu is not displayed
- **`-nostartup`** the startup file `.dinorc` is ignored
- **`-nostencil`** no attempt is made to initialize the stencil buffer that is used for surface solidification
- **`-nostereo`** on SGI, stereo is deactivated
- **`-s SCRIPTFILE`** executes `SCRIPTFILE` immediately after startup
- **`-f SCRIPTFILE`** same as -s, but the gfx is not updated during script parsing
- **`-stereo`** on Linux, searches for a stereo visual
- **`-vidmode`** on SGI, uses the specified number of videomode, as printed with `-debug`.
- **`-trace`** all script lines are echoed as they are run
- X-toolkit parameters such as `-geometry`.

1.3 Basics

The unit used throughout DINO is Å ($10^{-10}m$).

Color can be given either by a name (pre-defined colors are displayed with `scene showrgb`) or as a RGB triplet in the form `{r,g,b}`, where each color component lies between 0.0 and 1.0.

Vectors and matrices are also entered using curly braces:

$\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is written as `{x,y,z}`, $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$ is written as `{{a,b,c},{d,e,f},{g,h,i}}`.

1.4 Manual Conventions

In a *Syntax* statement, several characters have a special meaning:

`CAPITAL` letters indicate placeholders

`[]` square brackets enclose optional parts

`|` the vertical bar denotes a logical 'or':

- in square brackets this means *one of these statements or none*
- in round brackets this means *exactly one of these statements*
- without brackets it separates different syntaxes altogether

`...` ellipsis stand for an arbitrary amount of similar statements

The terms `SET_EXPR`, `MATERIAL_EXPR` and `RENDER_EXPR` are used to designate one or more comma-separated assignments in the form of

```
PROPERTY1=VALUE, PROPERTY2=VALUE2, ...
```

where the available properties and possible values are usually documented in a table.

1.5 Startup File

Upon startup, the file `.dinorc` is first looked for in the current directory and then in home directory. If it is present, it will be parsed. At the moment, parameters to adjust the input device speed and an exec block are supported, as well as comments starting with #.

Speed parameters are:

`mouse_rot_scale`, `mouse_tra_scale`, `sb_rot_scale`, `sb_tra_scale`, `dials_rot_scale`, `dials_tra_scale`

Each speed parameter must be followed by a single floating point number which is used as a multiplication factor. Default is 1.0 for all factors.

All commands within the exec block are executed immediately upon startup and can consist of any DINO command.

Example `.dinorc` file:

```
mouse_rot_scale 0.5
mouse_tra_scale 0.5
exec {
  #set background to white
  scene set bg=white
  # adjust field of view
  scene set fov=45
  # turn on depth cueing by default
  scene set depthc,fogo=30
}
```

2 Shell and GUI

The shell accepts commands and passes them to the dino-engine. The shell syntax itself is very limited, in particular there is no support for control loops (such as `for` or `while`) or conditionals (such as `if-else`); variables however can be defined.

2.1 Shell Navigation

Commands are typed into the terminal. Keystrokes occuring in the graphics window are routed directly to the terminal. Special keys:

← →	Move cursor within command line
↑ ↓	Navigate the history of previous commands
DEL	Removes character the cursor is positioned on
BKSPC	Removes character just left of the cursor
^K	Erases entire line

2.2 Shell Commands

!

Syntax: **`!SHELLCOMMAND`**

Abbreviation for the shell command `system` (see below).

@

Syntax: **`@SCRIPTFILE`**

Causes to specified `SCRIPTFILE` to be parsed, each line interpreted as a command. To spread a single command over several lines, use the backslash as the last character to protect the following newline. The commands `break` and `pause` (see below) are only valid during execution of a scriptfile.

$

Syntax: **`$VAR`**

Expands variable `VAR` to its value. Characters following immediately afterwards cannot be one of a-z, A-Z or 0-9[1] (because they would be interpreted as part of the variable name). Variables can be nested, i.e. `$$var` would first expand `$var`, and then the interpreter would try to expand the result again as a variable.

//

Syntax: **`// COMMENT`**

Ignores the rest of the line up to a newline, allowing comments to be added to scripts.

\

Syntax: **`\X`**

Protects character `X` from being interpreted, e.g. a newline in a script, a dollar sign, brackets or quotes.

[1] system call `isalnum()`

[]

Syntax: **[SUBCOMMAND]**

Allows nested commands. The square brackets are part of the syntax! The expression contained in SUBCOMMAND is first evaluated as a command by itself, and then, if no error occurred, the result replaces the [SUBCOMMAND] expression. This can be arbitrarily nested. Especially useful in combination with the echo command; e.g printing the current value of the global transformation:

```
echo [scene get rtc]
```

;

Syntax: **COMMAND1 ; COMMAND2**

The semicolon can be used to seperate individual commands that appear on the same line.

alias

Syntax: **alias ABBR EXPR**

Sets an alias, with the effect that if ABBR appears as the first word in a command, it is replaced by EXPR. ABBR must be a single word, EXPR can be several words. The resulting EXPR is *not* parsed for aliases again.

break

Syntax: **break**

Stops script execution and returns to the caller. Has no effect interactively.

cd

Syntax: **cd PATH**

Changes the working directory to PATH. If PATH is omitted the directory is reset to the initial startup directory.

echo

Syntax: **echo EXPRESSION [> file | >> file]**

Prints EXPRESSION, evaluating all subcommands and expanding all variables first. Output can be optionally redirected (> FILE) or appended (> > FILE) to a file.

exit

Syntax: **exit**

See quit.

pause

Syntax: **pause**

Halts script execution until a key is pressed. If the key is ESC, script execution will be aborted. Has no effect interactively.

pwd

Syntax: **pwd**

Displays current working directory.

quit

Syntax: **quit**

See exit.

set

Syntax: **set VAR EXPR**

Assigns EXPR to the variable VAR (see also $ above).

system

Syntax: **system EXPR**

Executes EXPR as a shell command and returns after it has completed.

unalias

Syntax: **unalias ABBR**

Removes the alias entry for ABBR (see also alias above).

unset

Syntax: **unset VAR**

Removes the variable VAR (see also set above).

var

Syntax: **var**

Lists all currently defined variables with their values.

Variables	
PI	3.14159
protein	(rname=ALA, CYS, ASP, GLU, PHE, GLY, HIS, ILE, LYS, LEU, MET, ASN, PRO, GLN, ARG, SER, THR, VAL, TRP, TYR)
dna	(rname=A,ADE,C,CYT,G,GUA,T,THY)
rna	(rname=A,ADE,C,CYT,G,GUA,U, URA)
aliphatic	(rname=ALA,GLY,ILE,LEU,MET,PRO,VAL)
aromatic	(rname=PHE,TYR,TRP)
basic	(rname=ARG,LYS)
basic2	((rname=LYS and aname=NZ) or (rname=ARG and aname=NH1,NH2))
acidic	(rname=ASP,GLU)
acidic2	((rname=GLU and aname=OE1,OE2) or (rname=ASP and aname=OD1,OD2))
polar	(rname=SER,THR,TYR,HIS,CYS,ASN,GLN)
polar2	((rname=SER and aname=OG) or (rname=THR and aname=OG1) or (rname=TYR and aname=OH) or (rname=HIS and aname=ND1,NE2) or (rname=CYS and aname=SG) or (rname=ASN and aname=OD1,ND1) or (rname=GLN and aname=OE1,NE1) or (rname=TRP and aname=NE1))
hydrophobic	(rname=ALA,VAL,PHE,PRO,MET,ILE,LEU,TRP)

Aliases	
stereo	scene stereo
mono	scene mono
write	scene write
bench	scene bench

Table 2.1: Predefined Variables and Aliases

2.3 Shell RPN calculator

The shell implements a RPN calculator: values are pushed onto a stack, and operators are applied to the stack to yield results:

clear

Syntax: **`clear`**

Removes all entries from the RPN stack.

dup

Syntax: **`dup`**

Duplicates the topmost RPN stack entry.

opr

Syntax: **`opr OP1 [OP2 ...]`**

Applies one or several operators to the RPN stack. See table 2.2 on the next page for a list of all operators.

peek

Syntax: **`peek`**

Returns the topmost value from the RPN stack without removing it.

pop

Syntax: **`pop [VAR [,VAR2 ...]]`**

Returns and removes the topmost value(s) from the RPN stack, optionally writing them in the comma-separated variable names given after the `pop` command. If no variable is provided, returns and removes only the topmost value.

push

Syntax: **`push W1 [W2 ...]`**

Pushes all words (from left to right) onto the RPN stack, the rightmost word will be on top.

show

Syntax: **`show`**

Lists the current RPN stack on the terminal.

swap

Syntax: **`swap`**

Swaps the two topmost entries on the RPN stack.

*Example calculate (2+3)*4:*

```
push 4 2 3; opr + *; show
```

unary operators		***valid types***[x]
`+-`	changes the sign on scalars, elements of vector or matrix	SVM
`inc, dec`	increases, resp. decreases scalar, elements of vector or matrix by one	SVM
`abs`	return absolute of scalar or length of vector	SV
`inv`	inverse	S
`log, ln, exp, sqrt`	calculates the decimal resp natural logarithm, exponential function or square root	S
`sin, cos, tan, asin, acos, atan`	trigonometric functions	S
`int, float`	returns the integer part or float, or forces return of float	S
`det`	returns determinant of matrix, invalid for scalar and vector	S
binary operators		***valid types***[x]
`+, -`	basic addition and subtraction, vectors and matrices must have identical elements	SS VV MM
`*`	multiplication or scalar product	SS SV SM VV VM MM
`/`	simple division	SS SV SM
`pow`	power of x (1st position) to y (2nd position)	S
`x`	calculates cross product between two vectors	VV
special		***valid types***[x]
dist	calculates distance between two vectors	VV
angle	calculates angle between two lines formed by three vectors	VVV
torsion	calculates torsion as defined by four vectors	VVVV
rmat	Requires a direction vector V and a scalar value S, returns the rotation matrix of a S degree rotation around axis V	WV

[x] S: scalar, V: vector, M: matrix

Table 2.2: Shell RPN Stack Operators

2.4 GUI

All keypresses occuring in the graphics window are routed directly to the DINO shell prompt and appear there as if typed into the terminal window directly.

The following input devices are available, if installed:

- `mouse`
- `dialbox`
- `spaceball`

The transformations caused by these input devices are per default routed to the scene, modifying the camera view. The dataset command `grab` (see section 4.3.1 on page 24) can be used to re-route the transformations to a dataset.

For each of these input devices, there is a special variant that is named in the same way, but with a `2` appended: `mouse2`, `dialbox2` and `spaceball2`. These `*2` variants specify the input device together with the *Ctrl* modifier key: If for example the mouse is transforming a dataset (through the use of `grab mouse`), pressing *Ctrl* and moving the mouse will still cause the camera view to be updated (because `mouse2` is still bound to the scene).

Mouse Clicking[2] the left mouse button in the graphics window has the following effects:

1. A line $\mathcal{L}$ is constructed that is perpendicular to the xy plane and goes through the current mouse position.The middle point of the intersection of line $\mathcal{L}$ with the near and far clipping plane is stored in the variable `CP` (for current point)
2. Of all atoms of *shown* coordinate objects that are within 0.2 Å of line $\mathcal{L}$ and that are between the near and far clipping plane, the one closest to the near clipping plane is selected and the following things happen:
   (a) its numeric form (.ds:#number) is stored in the variable `CS` (current selection).
   (b) its position is pushed onto the scene stack *and* stored in the variable `CP` (thereby overriding the interpolated point described above)
   (c) its name is displayed in the status bar
   (d) its label is toggled if the shift key was pressed during the mouse click

Depressing the right mouse button in the graphics window will cause the user menu to appear. Some shortcuts are accessible from there, unfortunately not user-customizable.

The table below lists the effects of mouse movement in the graphics window, depending on the mouse button(s) and modifier keys pressed.

Default Mouse Input Settings

Left MB	rotate x+y
Middle MB	translate z (slow and fast)
Left MB & Middle MB	rotate z
Left MB & Shift	translate x+y
Middle MB & Shift	slab width
Left MB & Middle MB & Shift	slab translation
MouseWheel	translate z (slow)
MouseWheel & Shift	translate z (fast)

[2] time between mouse button press and release < 200ms

Dialbox The table below lists the settings of the eight dials on a standard dialbox.

Default Dialbox Settings

slab width	⊙	⊙	slab translate
rotate z	⊙	⊙	translate z
rotate y	⊙	⊙	translate y
rotate x	⊙	⊙	translate x

Spaceball A spaceball combines the three translational and rotational axes into one device, and hence is much better suited for 3D navigation than the mouse or dialbox.

3 Scene

3.1 Scene Commands

Commands addressed to the scene are issued in an object-oriented manner: The target `scene` precedes the command and its parameters:

Syntax: **`scene COMMAND PARAMETERS`**

autoslab

Syntax: **`scene autoslab`**

Adjusts the front and back clipping plane to the minimal and maximal z value of the currently displayed objects. The coordinates of the object units will be used, not of the 3D primitives.

bench

Syntax: **`scene bench`**

Toggles benchmarking. If on, the scene will be continously updated and the refresh rate (in frames per second) will be displayed in the status bar. Use for qualitative benchmarking.

center

Syntax: **`scene center {X,Y,Z}`**

Centers the global rotation on the coordinates provided by `{X,Y,Z}`. Predestined for combination with a nested command (see shell commands above) such as:

```
scene center [.ds.obj]
```

Note: as explained below, the geometric center of an object is return if no command is given

get

Syntax: **`scene get PROP`**

Returns the scene property PROP (see table 3.1 on page 16).

grab

Syntax: **`scene grab INPUTDEVICE`**

Grabs INPUTDEVICE (see section 2.4 on page 12), routing its transformations to the scene camera. Upon startup, all available input devices are grabbed by the scene.

hide

Syntax: **`scene hide`**

Turns display of all objects *off*.

hidecp

Syntax: **`scene hidecp`**

Hides the marker for the current position stored in `$CP` (off at startup).

message

Syntax: **`scene message EXPR`**

Displays `EXPR` in the status bar on the bottom of the graphics window.

peek

Syntax: **scene peek**

Returns the topmost scene stack entry without removing it.

pop

Syntax: **scene pop**

Returns and removes the topmost scene stack entry.

push

Syntax: **scene push W1 [W2 ...]**

Pushes all following words (from left to right) onto the scene stack.

reset

Syntax: **scene reset [rot] [trans] [cent] [clip]**

Resets the scene transformation. If no additional keyword is provided, everything is reset, otherwise only the specified components:

rot The rotation matrix is set to identity

trans The translation vector is set to `{0,0,-100}`

cent The center of rotation is set to the origin `{0,0,0}`

clip the near clipping plane is set to 1.0, the far clipping plane to 1000.0.

rotm

Syntax: **scene rotm MATRIX**

Multiplies the supplied 3x3 MATRIX to the current rotation matrix. This matrix is *not* checked for validity.

rotx roty rotz

Syntax: **scene (rotx | roty | rotz) ANGLE**

Rotates the scene around one of the major axis by `ANGLE` degrees.

set

Syntax: **scene set SET_EXPR**

Sets one or more scene properties (see table 3.1 on the next page).

show

Syntax: **scene show**

Turns display of all objects *on*. This is the default.

showcp

Syntax: **scene showcp**

Displays a marker for the position stored in `$CP` (see also section 2.4 on page 12).

showrgb

Syntax: **scene showrgb [EXP]**

If `EXP` is omitted, lists all symbolic color names and their associated RGB values. Otherwise lists only those colors that include `EXP` in their name.

spin

Syntax: **scene spin**

Toggles scene spinning on and off. If active, the scene will rotate in the direction of the last rotation induced with the mouse.

split

Syntax: **scene split**

Toggles stereo display using split screen on and off. The orientation is determined by the scene property splitmode: 0 (default) is straight, 1 is cross-eye. During split screen mode, scene write (see below) will actually generate a stereo image as seen on the screen.

stereo

Syntax: **[scene] stereo [on | off]**

Toggles hardware stereo on and off. This command has been aliased so scene can be omitted. This is currently only supported on equipped SGI systems and some linux hardware. The availability of hardware stereo mode will be reported during startup (see also the startup parameters above).

transm

Syntax: **scene transm VECTOR**

Adds the supplied VECTOR to the current translation vector.

transx transy transz

Syntax: **scene (transx | transy | transz) VALUE**

Translates VALUE Å along one of the major axis.

Property	Description	type	default
bg	Background color	color	black
center	Center of rotation	position	{0,0,0}
depthc	If true, depth effect with fog is enabled	flag	false
dither	If true, dithers colors on displays with less than 24 bitdepth	flag	false
eyedist	Eyedistance (stereo parameter)	float	150
far	Distance of far clipping plane from observer	float	400
fixz	If true, clipping planes move along z-translation	flag	true
fogc	Fog color	color	black
fogm	Fog mode, one of linear, exp or exp2	flag	linear
fogd	Fog distance for modes exp and exp2	float	1.0
fogo	Fog offset from far clipping plane for mode linear	float	25.0
fov	Field of View Angle of perspective projection	float	0
mmat	Modelview, includes rotation and translation	4x4 mat	
near	Distance of near clipping plane from observer	float	10
rot	Rotation matrix	3x3 mat	identity
rtc	Rotation, Translation and Center of Rotation in one	4x4 mat	
slabw	Width of slab (equals far-near)	float	390
splitmode	Determines viewing mode for split screen stereo, 0 is straight, 1 is cross-eye	int	0
trans	Translation vector	vector	{0,0,-100}
view	view mode, one of center, left or right	mode	center

Table 3.1: Scene Properties

3.2 Lighting

Initially, only one lightsource is on (number 0). There are six more that can be used. In the syntax statements below, `LIGHT` stands for `scene:light`*N*, where *N* ranges from 0 to 7.

on

Syntax: **`LIGHT on`**

Turns the specified lightsource on.

off

Syntax: **`LIGHT off`**

Turns the specified lightsource off.

get

Syntax: **`LIGHT get PROP`**

Retrieves a light property (see table 3.2 on the next page).

set

Syntax: **`LIGHT set SET_EXPR`**

Sets one or more light properties (see table 3.2 on the following page).

show

Syntax: **`LIGHT show`**

Displays all properties of this light.

3.3 Additional Clipping Planes

In addition to the front and back clipping plane, six additional clipping planes in arbitrary position can be specified. In the syntax statements below, `CLIP` stands for `scene:clip`*N*, where *N* ranges from 0 to 5.

on

Syntax: **`CLIP on`**

Turns specified clipping plane on.

off

Syntax: **`CLIP off`**

Turns specified clipping plane off.

set

Syntax: **`CLIP set PROP1[,PROP2 ...]`**

Sets on or more clipping plane properties (seet table 3.2 on the next page).

get

Syntax: **`CLIP get PROP`**

Retrieves a clipping plane property (see table 3.2 on the following page).

grab

Syntax: **`CLIP grab INPUTDEVICE`**

Routes all transformations from INPUTDEVICE to specified clipping plane.

Property	**Description**	**default light0**	**default other**
Light Properties			
on *or* off	flag to turn the light on or off	on	off
global *or* local	flag to set either a local lightsource (fixed position in scene) or global lightsource (will move along with camera)	global	global
pos	4 dimensional vector in the form {x,y,z,w}. For w=0, {x,y,z} defines a direction vector, mimicking an infinite lightsource. For w≠ 0, {x,y,z} denotes a position in space.	{.1,.2,1,0}	{0,0,1,0}
amb	Ambient lighting contribution, either a grayscale value or an rgb triplet {r,g,b}, value range 0-1	0.05	0.0
diff	Diffuse lighting contribution, either a grayscale value or an rgb triplet {r,g,b}, value range 0.0-1.0	0.6	1.0
spec	Specular hilight contribution, either a grayscale value or an rgb triplet {r,g,b}, value ange 0.0-1.0	0.3	1.0
kc	Constant attenuation factor	1.0	1.0
kl	Linear attenuation factor	0.0	0.0
kq	Quadratic attenuation factor	0.0	0.0
spotc	Spotlight cutoff angle from 0-90 degrees, or 180 to turn spotlight off	180	180
spote	spot exponent	0.0	0.0
spotd	spot direction	{0,0,-1}	{0,0,-1}
Clipping Plane Properties			
Property	Description	Default	
pos	Position of the clipping plane	{0,0,0}	
dir	Direction of the plane normal	{0,0,1}	

Table 3.2: Light and Clipping Plane Properties

4 Datasets

4.1 Concepts

4.1.1 Basic Data-Units

Each dataset is build of basic data-unites that are usually loaded from an external file. Each data-unit is characterized by properties. The dataset itself is not visible on the screen, it is a memory-only representation of the data. To visualize the data, objects must be created.

4.1.2 Objects

Objects are three dimensional representations of datasets. They are constructed from the data-units with the dataset command `new` (see on page 24), resulting in a collection of 3D primitives which are displayed in the graphics window. Objects are characterized by:

Object Name A sequence of alphanumeric characters (allowed are pretty much all characters except a blank space and special shell characters such as `$ // @ [ ] { } ;`), must be unique within a dataset.

Object Type Represents a specific way of converting and interpreting the structural data, resulting in different representations for the same dataset.

Selection Designates a subset of the data-units to be included in the object construction.

Dataset and Object Properties Additional parameters unique to the dataset and object type.

Render Properties Parameters affecting the display of the 3D primitives.

4.1.3 Addressing Datasets and Objects

Every dataset has a unique name (set during loading or creation). Commands addressed to the dataset are issued in an object-oriented manner:

Syntax: **`.DS COMMAND PARAMETERS`**

The name of the dataset (`DS`), preceded with a dot, appears first, followed by the command and its parameters.

Objects are part of the dataset they were created from and are addressed as:

Syntax: **`.DS.OBJ COMMAND PARAMETERS`**

The object name (`OBJ`) is always appended to its dataset (`DS`), separated with a dot.

4.1.4 Properties

Properties are name-value pairs characterizing dataset, objects and data-units. The data-unit properties can be divided into two classes: some are *copied* from the dataset to the object during object (re-) creation and some are *shared* between the dataset and its objects. As described in more detail below, the commands `set` and `get` are available on the dataset and the object level to modify and retrieve properties.

4.1.5 Selection

Selection can be applied for various commands and allows a filtering of dataset or object elements, to which the specified command is applied. Each element is queried against the complete selection, and only if the selection holds true the element is used.

A selection is build up from individual selection statements, connected through boolean operators, optionally employing parenthesis to group statements. Possible boolean operators are `and`, `or` (both binary) as well as `not` (unary). There are three types of selection statements, *property*, *within* and *object*, explained in more detail below.

4.1.5.1 Selection by property

Syntax: **`PROP OP VALUELIST`**

`PROP` is a data-unit property that is valid within a selection statement (as indicated in the tables for each dataset), `OP` is one of the comparison operators listed below, and `VALUELIST` consists of one or more (comma separated) `VALUE`s. A `VALUE` is either a string, a number or a numeric range (`MIN:MAX`). For the string and range `VALUE`s, only the equal / not-equal operators are valid.

The selection statement is true if the comparison between the queried element property and any one of the listed values holds true.

Comparison operators for selection statement
= (equal) **!=** (not equal) **<** (smaller) **<=** (smaller or equal) **>** (larger) **>=** (larger or equal)

Examples:

```
rname=ALA,LEU,ILE
rnum=1:20,30:40 and chain=A
(rnum<50 and aname=C,N,O,CA) or (rnum=55,76,129)
```

4.1.5.2 Selection within a distance

Syntax: **`DIST <> TARGETLIST`**

The within statement allows to select based on a distance `DIST` Å from a `TARGETLIST`, which consists of one or more (comma separated) `TARGET`s. A `TARGET` is either an point in space - with the syntax `{x,y,z}` - or an arbitrary object from *any* other dataset - with the syntax `.DS.OBJ`. In the latter case, the selection statement will be true if the queried element falls within the specified distance of *any* of the object elements .

Examples:

```
10 <> {0,0,0}
5.5 <> $CP
20 <> .myo.hem
20 <> [.myo.hem]
```

The difference between the last two expressions is the following: without square brackets, the selection is true within 20Å of any element of `.myo.hem`, while the square brackets are first evaluated and return the center of gravity for the object, resulting in a spherical selection.

4.1.5.3 Selection based on other objects

Syntax: **`OBJLIST`**

This statement contains one or more (comma separated) object names of the dataset itself. The selection is true if any dataset element that is queried is contained within one of the objects.

4.1.6 Ranges

Syntax:
`.DS.OBJ set OP=OV1:OV2 -range prop=RP[,src=SRC][,val=RV1:RV2][,clamp]`

A range is used to lineary map one property onto another. It is appended to a `set` statement, which must contain at least one property with a value-range. The value `OV` for each object element property `OP` is determined in the following way:

1. If `src` is omitted or set to `.DS` (the dataset the object belongs to), the value `RV` of the dataset property `RP` of the object element is obtained. Else, the value `RV` of the property `RP` of the dataset `SRC` *at the coordinates of the object element* is obtained (if necessary by interpolation).
2. `RV` is linearly mapped to `OV` with the following formula:

$$OV = \frac{(RV - RV1)}{(RV2 - RV1)} \cdot (OV2 - OV1) + OV1$$

3. If `clamp` was specified, `OV` will be clamped to lie within `OV1` and `OV2`.

Only those object elements will be affected which `OV` value lies within the specified range from `OV1` to `OV2`.

In the tables below, the symbol • in the *R*ange column denotes that this property may be used as RP, while × denotes that this property may be used as OP.

4.1.7 Transformations

A dataset can be globally transformed with respect to the scene (and the other datasets). This transformation is defined by a rotation matrix R, a translation vector T and a centering vector C. The transformation is applied to each object element coordinate v prior to rendering to yield the transformed coordinate v':

$$v' = R \cdot (v - C) + C + T$$

The rotation matrix defaults to the identity matrix, the translation vector to zero, and the centering vector to the geometric center of the dataset. The dataset commands and properties affecting the transformation are given in the generic dataset command section 4.3.1 on page 24 and in the dataset property tables 4.4 on page 30, 4.7 on page 35, 4.10 on page 38 and 4.13 on page 41.

4.2 Database Manager Commands

The database manager module handles the internal database containing all datasets and their objects.

delete

Syntax: **`delete DATASET`**

Removes `DATASET` and all its objects. Note that the name of the dataset is not preceded with a dot.

list

Syntax: **`list`**

Lists all loaded datasets on the terminal.

load

Syntax: **`load FILE [-name N] [-type T] [SPECIFIC PARAMS]`**

Probably the most important database manager command. Loads a file into the database, creating a new dataset and converting the file format into the internal format.

The name of the dataset can be specified with `-name`, otherwise the base of the filename[1] will be used. If a dataset of identical name already exists, consecutive numbers starting from 2 will be appended to the name until a unique one is found; the exisiting dataset will *not* be overwritten. Allowed characters are A-Z, a-z, 0-9, underscore and hyphen; other characters will be replaced with underscore.

The file type given with `-type` indicates both the kind of dataset to be created as well as the file format. If this parameter is omitted, the type is guessed from the extension. Table 4.1 on the next page lists the currently supported file formats, their recognized extension(s), the type and the kind of dataset that will be created. *NOTE:* Files compressed with `gzip` (extension `.gz`) are uncompressed on the fly. It is not event necessary to append the `.gz` extension to the filename in the load command.

Depending on the file type, some other parameters can be given:

Files in binary format might need to be byte-swapped with `-swap`, depending on the processor architecture they were generated on and the processor architecture DINO is running on. The byte order of the supported architectures is as follows: **Big Endian**: MIPS (irix), MOTOROLA (OSX), SPARC (sun), **Little Endian**: INTEL (linux-i386), ALPHA (osf1). *NOTE:* For most binary formats, DINO will try to detect wether byte-swapping is necessary, so this flag can be ommited in most cases.

For coordinate dataset file formats, the parameter `-conn CFLAG` will determine the connectivity rules applied upon startup (for more details see connectivity description on p.27).

The parameter `-conv` is specific for UHBD potentials - it causes a multiplication of each scalar grid value with the scale parameter contained within the header.

new

Syntax: **`new TYPE [-name N]`**

Creates a new dataset of `TYPE`. This is currently limited to `geom`. If no name is specified, `TYPE` is used.

rename

Syntax: **`rename OLD NEW`**

Renames dataset `OLD` into `NEW`, provided that `NEW` is not already used. Note that there is no dot predecing the dataset names.

[1] stripped of its path and extension

File Format	Ext	Type	Dataset	Mode
Protein Data Bank coordinate file www.rcsb.org	.pdb .ent	pdb	coord	asc
X-PLOR v3.x coordinate file atb.csb.yale.edu/xplor/	.xpl	xplorc	coord	asc
CNS v1.0 coordinate file cns.csb.yale.edu	.cnsc	cnsc	coord	asc
CHARMM coordinate file yuri.harvard.edu	.crd	charmmc	coord	asc
MEAD coordinate file www.scripps.edu/bashford	.pqr	pqr	coord	asc
Special BD Trajectory format	.bdtrj	bdtrj	coord	bin
GROMACS coordinate file	.gro	gromacs	coord	asc
electron density or mask from the CCP4 suite www.dl.ac.uk/CCP/CCP4/	.map .ccp4	ccp4	scal	bin
X-PLOR v3.x electron density or mask atb.csb.yale.edu/xplor/	.xmp .xmap	xplorb	scal	bin
CNS v1.0 electron density or mask cns.csb.yale.edu	.cmp .cmap	cnsb	scal	bin
UHBD grid file chemcca51.ucsd.edu/uhbd.html	.uhb .uhbd	uhbd	scal	bin
CHARMM electrostatic potential yuri.harvard.edu	.cpot	charmmb	scal	bin
MEAD electrostatic potential www.scripps.edu/bashford	.fld .mead	mead	scal	bin
DELPHI/INSIGHTII electrostatic potential trantor.bioc.columbia.edu/delphi/	.grd .ins	delphi	scal	bin
fixed (64^3) DELPHI potential (i.e. as output from GRASP)	-	delphig	scal	bin
SPIDER scalar field	.spi	.spider	scal	bin
simple DINO scalar field	.dgrd	dgrid	scal	bin
X-PLOR v3.x electron density or mask (ASCII) atb.csb.yale.edu/xplor/	-	xplora	scal	asc
CNS v1.0 electron density or mask (ASCII) cns.csb.yale.edu	-	cnsa	scal	asc
MSMS surface www.scripps.edu/pub/olson-web/people/sanner	.face .vert	msms	surf	asc
MSP surface www.biohedron.com	.msp .vet	msp	surf	asc
GRASP surface trantor.bioc.columbia.edu/grasp/	.grasp	grasp	surf	bin
ADS surface www.embl-heidelberg.de/~gabdoull/ads/	-	ads	surf	asc
greyscale TIFF image	.tiff	topo	topo	n.a.

Table 4.1: Supported File Formats and their Dataset Types (supported MD trajectory file formats are listed in table 4.3 on page 28)

4.3 Generic Commands

A number of dataset commands applying to all datasets or objects are described here. The specific commands are explained within the respective dataset section below.

4.3.1 Generic Dataset Commands

Ommiting a command will be interpreted as `.DS get center`

del

Syntax: **`.DS del OBJ`**

Removes the object named `OBJ` from the dataset. Note that there is no dot preceding the object name.

fix

Syntax: **`.DS fix`**

Applies the current transformation to the dataset, then resets the transformation to identity.

get

Syntax: **`.DS get PROPERTY`**

Returns the specified dataset property. (See tables 4.4 on page 30, 4.7 on page 35, 4.10 on page 38 and 4.13 on page 41).

grab

Syntax: **`.DS grab INPUTDEVICE`**

Grabs `INPUTDEVICE` (see section 2.4 on page 12). The transformations generated by this input device will be routed to the dataset. Use `fix` to actually apply the transformation to the dataset.

new

Syntax:
`.DS new [-name N][-type T][-set SET_EXPR][-sel SELECT_EXPR]`

Creates a new object from the dataset. If no name is given with `-name`, the dataset name will be used. If an object with the same name already exists, it will be deleted first. The object type - a specific way to convert the data into 3D primitives - can be specified with `-type`. Object properties as well as *shared* data-unit properties can be assigned using `-set`. Finally, a subset of the data-units used for object creation can be selected with `-sel`.

reset

Syntax: **`.DS reset (rot | trans | center | all)`**

Resets the dataset transformation. If no parameter is given, `all` is implied.

rot The rotation is set to identity.

trans The translation is set to `{0,0,0}`.

center The center of rotation is set to the geometric center of the dataset.

restrict

Syntax: **`.DS restrict SELECT_EXPR`**

All data-units that do *not* match the selection criteria in `SELECT_EXPR` are flagged as excluded and are ignored for all subsequent commands. The wildcard `*` will remove all restrictions. This command does *not* act cumulatively, ie the restriction is removed prior to each restrict command.

rotx roty rotz

Syntax: **.DS (rotx | roty | rotz) ANGLE**

Rotates around one of the major axis by ANGLE degrees. The major axis are oriented relativ to the current camera orientation, i.e. the x, y and z axis are horizontal, vertical and perpendicular to the screen, respectively.

set

Syntax: **.DS set SET_EXPR [-sel SELECT_EXPR]**

Sets dataset or data-units properties (depending on which appear in SET_EXPR). A selection can be added to specify which data-units are affected. *Shared* data-units (see section 4.1.4 on page 19 above) will be updated immediately in all objects, while *copied* data-units change the default values for subsequent object (re-)creations. The tables in the specific dataset sections below contain information about valid dataset properties (table 4.4 on page 30, table 4.7 on page 35, table 4.10 on page 38 and table 4.13 on page 41) as well as *shared* and *copied* data-units (table 4.5 on page 31, table 4.8 on page 36, table 4.11 on page 38 and table 4.14 on page 42).

The syntax of the selection is given in section 4.1.5 on page 20.

transx transy transz

Syntax: **.DS (transx | transy | transz) VALUE**

Translates the dataset VALUE Å along one of the major axis. The axis are defined in the same way as for the rotation (see above).

4.3.2 Generic Object Commands

Omiting a command will be interpreted as `.DS.OBJ get center`

get

Syntax: **.DS.OBJ get PROPERTY**

Returns the specified object property. (See tables 4.4 on page 30, 4.7 on page 35, 4.10 on page 38 and 4.13 on page 41).

hide

Syntax: **.DS.OBJ hide**

The object will no longer be displayed in the graphics window.

set

Syntax:
.DS.OBJ set SET_EXPR [-sel SEL_EXPR] [-range RANGE_EXPR]

A very flexible and powerful command; sets object properties or *copied* data-units properties (depending on which appear in SET_EXPR). The latter can be submitted to a selection and/or range. For a range statement to work, at least on property appearing in SET_EXPR must have a value-range.

The tables in the specific dataset sections below contain information about the valid object properties (table 4.4 on page 30, table 4.7 on page 35, table 4.10 on page 38 and table 4.13 on page 41) and *copied* data-units (table 4.5 on page 31, table 4.8 on page 36, table 4.11 on page 38 and table 4.14 on page 42). The update of properties might not take effect immediately: some object properties will only be evaluated during a renew (see below).

An explanation of the selection syntax is given in section 4.1.5 on page 20 and of the range syntax in section 4.1.6 on page 21.

show

Syntax: **.DS.OBJ show**

Displayes the object in the graphics window. Per default all objects are shown once created.

renew

Syntax: **.DS.OBJ renew [-set SET_EXPR] [-sel SELECT_EXPR]**

Renews an object. This is similar to the dataset command `new` (see above), except that the name and the type of the object are fixed and only new properties and/or a new selection can be applied. *Copied* data-unit properties will be regenerated from the default values, while object properties will only be modified if explicitely stated in `SET_EXPR`. If no selection is specified, the old one will be re-applied, otherwise the new selection is evaluated, *replacing* the old one.

render

Syntax: **.DS.OBJ render [RENDER_EXPR]**

Modifies render properties contained in `RENDER_EXPR`. If called without parameters the current rendering state will be renewed. Render properties are listed in tables 4.6 on page 32, 4.9 on page 36, 4.12 on page 39 and 4.15 on page 42.

material

Syntax: **.DS.OBJ material MATERIAL_EXPR**

Changes the surface material of an object through material properties (table 4.2) given in `MATERIAL_EXPR`, affecting the interaction between the light sources (see section 3.2 on page 17) and the objects. Without `MATERIAL_EXPR`, the current settings are shown.

Example:

```
.surf.obj material amb=0.2,spec=0.5,shin=64
```

Note: The diffuse material setting is determined by the color.

Property	Description
amb	Ambient light, either a scalar or explicit color `{r,g,b}`
spec	Specular hilights, either a scalar or explicit color `{r,g,b}`
shin	Shininess, integer value 1-128
emm	Emmission, either a scalar of explicit rgb color `{r,g,b}`

Table 4.2: Material Properties

4.4 Coordinate Dataset

Note: this used to be called structure dataset, but has been renamed to signify that it is based on atomic coordinates.

4.4.1 Basic Data-Unit

The coordinate dataset has individual atoms as its basic data-unit. In addition, it contains their organization in the molecular architecture and optionally several conformations.

4.4.2 Object Types

connect All selected atoms are connected by bonds based on their chemical connectivity. Two atoms are connected if they fullfill one of the criteria listed below. These rules are applied per default for the database manager command `load` and the dataset command `reconnect`. To change this default behaviour, a flag can be passed to these commands, indicating which rules are to be applied. The necessary value can be obtained by adding the desired flag numbers given after the rules.

1. Both atoms (defined by atom and residue name) are present and connected in the internal connectivity table (defined for the standard 20 amino and 5 nucleic acids) - *flag 0x1*
2. The two atoms were explicitely connected in the file (e.g. with a CONECT card) or by a `connect` command - *flag 0x2*
3. If *one or both* of the atoms is *not* present in the internal connectivity table, both atoms are connected if their distance is less than half the sum of their van der Waals radii - *flag 0x4*

trace All selected residues are sequentially connected by their central atom (CA for proteins, P for nucleic acids) if

1. Their residue numbers are continuous (n and n+1)
2. They belong to the same chain and model.

The selectable properties are listed in table 4.5 on page 31 under column L.

4.4.3 Dataset Commands

For the generic commands `del`, `fix`, `get`, `grab`, `new`, `reset`, `restrict`, `rot`, `set`, and `trans` see section 4.3.1 on page 24. Specific coordinate dataset commands follow:

connect

Syntax: **`.DS connect ATOM ATOM`**

Adds a covalent bond between the two atoms. The syntax for `ATOM` is described in section 4.4.5 on page 29.

load

Syntax: **`.DS load FILE [-type T]`**

Loads a trajectory file as an addon to the structure. The number of atoms in each trajectory frame must match the number of atoms in the dataset. The file type can be explicitely set with `-type`, otherwise it is guessed from the file extension. The table below lists the supported trajectory formats. All of these are binary, but DINO detects different endianess and performs byte-swapping if necessary.

type	ext	format
charmm	.trj .dcd	CHARMM trajectory
gromacs	.xtc	GROMACS trajectory
cns	.crd	CNS trajectory
dino	.dtrj	DINO trajectory
binpos	.binpos	BINPOS trajectory

Table 4.3: MD Trajectory formats supported.

play

Syntax:
.DS play [-b BEG][-e END][-w WAIT][-s STEP][-d DEL][-m MODE]

Commences trajectory playing, going from frame BEG to frame END (default all) in steps of STEP (default 1), waiting WAIT cycles between each step (default 0), adding a delay of DEL cycles at the end before continuing (default 0). MODE is one of loop (default, at END jump to BEG after DEL cycles), rock (going back and forth) or single (stop after one pass). The frame updates are implemented very efficiently with a single copy of a memory area containing the new coordinates for the atoms. This is fine for object types simple and cpk, but object type custom requires some more calculations and the dataset property tfast should be set to false to ensure proper rendering (see table 4.4 on page 30).

reconnect

Syntax: **.DS reconnect CFLAG**

Reruns connectivity algorithm, using connectivity rules described above (p.4.4.2). CFLAG defaults to 7, ie applying all three connectivity rules.

step

Syntax: **.DS step [N]**

Jumps to the next or +N trajectory step.

stop

Syntax: **.DS stop**

Halts a playing trajectory and resets the current frame to 1.

write

Syntax: **.DS write FILE [-type T]**

Writes all *unrestricted* atoms of the dataset into a file. The format can be explicitely set with -type, otherwise it is guessed from the extension. The table below lists the supported formats and their corresponding type and extension.

type	ext	format
pdb	.pdb	PDB coordinate file
cns xplorc	.xpl	CNS/X-PLOR coordinate file
charmmc	.crd	CHARMM coordinate file

4.4.4 Object Commands

For the generic commands get, hide, show, set, renew, render, and material see section 4.3.2 on page 25. Specific coordinate object commands follow:

clear

Syntax: **.DS.OBJ clear**

Removes all labels from this object.

write

Syntax: **.DS.OBJ write FILE [-type T]**

In most respects identical to the dataset command `write` (see above), except that all atoms in the object are written out.

4.4.5 Individual atoms

Individual atoms are addressed as

Syntax: **.DS:ATOM COMMAND PARAMETERS**

where `ATOM` is one of:

- `[MODEL.][CHAIN.]RESIDUE-NUMBER.ATOM-NAME`
- `#NUMBER`

`MODEL` and `CHAIN` are only required if the dataset contains several models and/or chains. `NUMBER` is the unique atom number read from the file. Chain and atom name are case sensitive.

Ommiting a command will be interpreted as `.DS:ATOM get xyz`

get

Syntax: **.DS:ATOM get PROP**

Retrieves atom property (see table 4.5 on page 31). Especially usefull after clicking on an atom, as its individual atom code is then stored in the shell variable `$CS`.

4.4.6 Dataset, Object and Data-Unit Properties

Tables 4.4 on the next page and 4.5 on page 31 list all the dataset, object and data-unit properties that are accessible with the dataset and object commands `set` and/or `get`, as well as available during selection or range statements.

4.4.7 Render Modes and Properties

Rendering properties mentioned below (in `typewriter` font) for coordinate objects are listed in table 4.6 on page 32.

Object type connect:

`simple` (default) Bonds are drawn as lines of width `lw`, colored depending on both atoms they connect. Non-bonded atoms are displayed as little crosses.

`cpk` Atoms are displayed as (hollow) spheres with their van der Waals radius. The circular subdivisions of the spheres are controlled by `detail`.

`custom` Bonds are drawn as cylinders of width `bw`, colored depending on the atoms they connect. Atoms are drawn as spheres of radius `sr`. Circular subdivisions of the cylinders and spheres are controlled by `detail`.

Object type trace:

simple (default) Same as for type connect.

custom Same as for type connect.

sline Smoothed spline with splines subdivisions passing *exactly* through the backbone centers; drawn as lines with width lw. The coloring is either smoothly interpolated along the spline segments or changed abruptly (intpol).

tube As sline, but a hollow tube is drawn, with the diameter tubew. If the flag userad is set, the diameter is multiplied with the radius of the central backbone atom (CA for proteins, P for nucleic acids). The axial ratio of the tube can be modified with tuber. The amount of circular subdivisions is set with detail.

hsc Similar to sline, displaying secondary structure cartoon for proteins and nucleic acids. Protein traces are rendered according to their residue type (see table 4.5 on the following page), with type *coil* displayed as a hollow tube of diameter tubew and axial ration of tuber, type *helix* displayed as a smooth helix with width helixw and thickness helixt, and type *strand* displayed as a pointed arrow with width strandw, thickness strandt and relative arrow size arrowt. Nucleic acid traces are composed of a spline running through the C3' positions of the sugar units (displayed as a tube with diameter tubew and axial ratio tuber) and one of two different sugar-base representations: either (nam=0) as a schematic sugar-purin or sugar-pyrimidine representation (of thickness sugart and baset) *or* (nam=1) as a hollow tube (of diameter bw) pointing towards the tip of the base. The amount of circular subdivisions is set with detail.

Property	*S*	*G*	Description
Dataset Properties			
cell	•	•	Crystallographic unit cell
center		•	Geometric center of dataset
frame	•	•	Current frame number (for trajectories)
rcen	•	•	Center of rotation, default is the geometric center
rot	•	•	3x3 rotation matrix, default is identity
rtc	•	•	4x4 compact matrix containing rotation, translation and center of rotation
smode	•	•	selection mode, can be either atom (default) or residue, in the latter case the complete residue will be selected if one of its atoms fulfills the selection criteria
tfast	•	•	Flag for fast trajectory update (default true). Disable when using render mode custom with trajectory playing
trans	•	•	Translation vector, default is {0,0,0}
Object Properties			
center		•	Geometric center of object

S: Property can be modified with set
G: Property can be obtained with get

Table 4.4: Coordinate Dataset and Object Properties

Property	S	G	L	R	Description
data-unit properties shared between dataset and objects					
`aname`		•	•		Name of atom
`anum`		•	•	•	Continuous number of atom
`bfac`		•	•	•	Crystallographic temperature factor of atom
`chain`		•	•		Chain name
`class`		•	•		Residue class, one of `protein`, `na` (nucleid acids) or `misc`
`ele`		•	•		Chemical element symbol of atom
`model`		•	•	•	Model number if coordinates read from a multi model file
`rname`		•	•		Name of the residue
`rnum`		•	•	•	Number of residue in the chain
`rtype`	•	•	•		Residue type, indicates secondary structure conformation this residue belongs to. One of `helix`, `strand` or `coil`
`weight`		•	•	•	Weight or crystallographic occupancy of atom
`xyz`		•			Atom position as `{x,y,z}`
`x`		•	•	•	x coordinate of atom position
`y`		•	•	•	y coordinate of atom position
`z`		•	•	•	z coordinate of atom position
data-unit properties copied from dataset to object type connect					
`color`	•			×	Color of atom
`vdwr`	•	•	•	•	Van der Waals radius of atom
data-unit properties copied from dataset to object type trace					
`color`	•			×	Color of central atom, upon setting will override `color1`, `color2` and `color3`
`color1`	•			×	Color of the phosphate backbone for nucleic acids `hsc` rendering mode
`color2`	•			×	Color of the sugar unit for nucleic acids `hsc` rendering mode
`color3`	•			×	Color of the base unit for nucleic acids `hsc` rendering mode
`rad`	•			×	Radius in Å of the tube in the equally named rendering mode

S Property can be modified with set
G Property can be retrieved with get
L Property can be used in a selection statement
R Property can be used in a range statement after `-range` (•) or as the property to be set (×)

Table 4.5: Coordinate Data-Unit Properties

Property	Description	Default
arrowt	1.0+arrowt is ratio of arrow width to strand width	0.0
baset	thickness in Å of nucleic acid base in hsc mode	0.5
bw	bond width in Å for mode custom	0.2
cull	flag to turn culling on or off	1 (on)
detail detail1	number of subdivisions for cylinders and spheres in modes cpk, custom, tube and hsc	3
splines detail2	number of interpolation steps in spline, used for rendering modes sline, tube and hsc	6
fast	mode: speeds up rendering with a loss of quality, opposite of nice	inactive
helixt	thickness in Å of alpha helix in hsc mode	0.3
helixw	width in Å of alpha helix in hsc mode	1.0
intpol	flag to turn color interpolation on or off for modes sline, tube and hsc	1 (on)
lw	line width in pixels for modes simple and sline, fractional values are supported on some graphic systems	1.0
nam	nucleic acid rendering method: 0 or 1	0
nice	mode: increases graphical quality with a decrease in speed, opposite if fast	active
sr	sphere radius in Å for mode custom	0.2
stipple	flag to turn stippling on or off	0 (off)
stipplei	length of stipple segment in Å	0.7
stippleo	length of stipple gap in Å	0.3
strandm	beta strand rendering method: 0 or 1	0
strandt	thickness in Å of beta strand in hsc mode	0.3
strandw	width in Å of beta strand in hsc mode	1.2
sugart	thickness in Å of nucleic acid sugar in hsc mode	0.5
t	transparency; 1.0 is fully opaque, 0.0 is fully transparent	1.0
tuber	axial ratio of tube for modes tube and hsc	1.0
tubew	diameter of tube for rendering modes tube and hsc	0.4
userad	flag to indicate that the atom radius is multiplied with the segment radius for mode tube	off

Table 4.6: Coordinate Object Render Modes and Properties

4.5 Scalar Field Dataset

4.5.1 Basic Data-Unit

The scalar field dataset is build up from individual grid points, each described by a point in space and a (scalar) value.

4.5.2 Object types

In contrast to the coordinate dataset, the objects of the scalar-field dataset are much more influenced by object properties:

contour Iso-contoured surface at a value of `level`. The object is centered on `center`, with dimension `size`. The color of each contour vertex is set with property `color`.

grid All scalar values on specified grid (defined by `center`, `size` and `step`) are displayed as points or spheres (see also rendering modes below). The color and radius of the spheres can be modified with the command `set`.

slab Planar slab - defined by `dir` and `center` - cutting through grid volume. The smallest rectangle that encompasses the intersection between plane and volume is constructed and subdivided into `size` x `size` points. The color of each point within the slab can be set with the `color` property.

4.5.3 Dataset Commands

For generic commands `del`, `fix`, `get`, `grab`, `new`, `reset`, `restrict`, `rot`, `set`, and `trans` see 4.3.1 on page 24. Specific scalar field dataset commands follow:

add

Syntax: **`.DS add .DS2`**

Adds scalar values from `DS2` to `DS`. The two grids must have equal dimensions.

mul

Syntax: **`.DS mul .DS2`**

Multiply scalar values from `DS2` with `DS`. The two grids must have equal dimensions.

sub

Syntax: **`.DS sub .DS2`**

Subtract scalar values of `DS2` from `DS`. The two grids must have equal dimensions.

4.5.4 Object Commands

For the generic commands `get`, `hide`, `show`, `set`, `renew`, `render`, and `material` see 4.3.2 on page 25. There are no scalar-field specific object commands.

4.5.5 Dataset, Object and Data-Unit Properties

Tables 4.7 on page 35 and 4.8 on page 36 list all the dataset, object and data-unit properties that are accessible with the dataset and object commands `set` and/or `get`, as well as available during selection or range statements.

4.5.6 Render Modes and Properties

The render properties (in `typewriter` font) mentioned below are listed in table 4.9 on page 36.

Object type contour:

dots Only the points on the unit cell edges, corresponding to the iso-contour value, are displayed, with a size of `ps`.

lines (default) The points on the unit cell edges are connected with lines of width `lw`.

fill A continuous surface, lit from both sides.

Object type grid:

on The grid points are shown as spheres, the radius depending on the data-unit property `rad` (see table 4.8 on page 36).

off (default) The grid points are shown as points of size `ps`.

Object type slab: *none*

Property	*S*	*G*	Description
Dataset Properties			
`{u,v,w}`		•	Returns scalar value at specified grid position
`center`	•	•	Geometric center of dataset
`edge`	•	•	Scalar value to use for areas outside grid, default 0.0
`rcen`	•	•	Center of rotation, default is the geometric center
`rot`	•	•	3x3 rotation matrix, default is identity
`rtc`	•	•	4x4 compact rotation, translation and center matrix
`scale`	•	•	Additional factor multiplied with the unit cell axis, default is 1.0
`trans`	•	•	Translation vector, default is `{0,0,0}`
`vm & vc`	•	•	scalar value at each grid point is (vm*VAL+vc) - default for vm is 1.0, for vc 0.0
Object Properties for type contour			
`center`	•	•	Center of object, default is the geometric center
`level`	•	•	Defines the contour level, if the letter `s` is appended the value is interpreted as standard deviation units. The default value is 1.0s for file formats CCP4, XPLOR and CNS, 0.0 for all others.
`size`	•	•	Size in grid-units, either a single number for a cubic extension or a triplet `{usize,vsize,wsize}`, default is 30
`step`	•	•	Stepsize along the grid units, default is 1 *not yet implemented*
Object Properties for type grid			
`center`	•	•	Center of object, default is the geometric center
`size`	•	•	Size in grid-units, either a single number for a cubic extension or a triplet `{usize,vsize,wsize}`, default 30
`step`	•	•	Stepsize along the grid units, default is 1
Object Properties for type slab			
`center`	•	•	Center of plane, default is `{0,0,0}`
`dir`	•	•	Direction of plane normal, default is `{0,0,1}`
`size`	•	•	Dimension of the internal texture mapped onto the rectangular plane, default is 64 - must be one of 8, 16, 32, 64, 128, 256 or 512

S: Property can be modified with `set`
G: Property can be obtained with `get`

Table 4.7: Scalar Field Dataset and Object Properties

Property	*S*	*G*	*L*	*R*	Description
data-unit properties shared between dataset and objects					
`v`			•	•	Scalar value
`x`			•	•	x coordinate of grid point
`y`			•	•	y coordinate of grid point
`z`			•	•	z coordinate of grid point
`dist`			•	•	distance from current center of rotation
data-unit properties copied from dataset to object type contour					
`color`	•			×	Color of vertex
data-unit properties copied from dataset to object type grid					
`color`	•			×	Color of grid point
`rad`	•	•		×	Radius in Å of spheres
data-unit properties copied from dataset to object type slab					
`color`	•			×	Color of each point on rectangle, defined by the intersection of the slab with the scalar field volume

S: Property can be modified with set
G: Property can be retrieved with get
L: Property can be used in a selection statement
R: Property can be used in a range statement after `-range` (•) or as the property to be set (×)

Table 4.8: Scalar Field Data-Unit Properties

Property	Description	Default
Render Properties		
`ps`	Point size for render mode dots, object type contour	1.0
`lw`	Line width for render mode lines, object type contour	1.0
`t`	transparency for object types contour and slab	0.7

Table 4.9: Scalar Field Object Render Modes and Properties

4.6 Surface Dataset

4.6.1 Basic Data-Unit

This dataset has a surface point as its basic data-unit, each characterized by a vertex, a normal and optionally some attributes. It also contains a list of point triplets which together form a triangle face. The faces have an inside and an outside, defined by the direction of the surface normal and the orientation of the three vertices.

4.6.2 Object Types

The surface dataset has only the default object type.

4.6.3 Dataset Commands

For the generic commands `del`, `fix`, `get`, `grab`, `new`, `reset`, `restrict`, `rot`, `set`, and `trans` see 4.3.1 on page 24. Special surface dataset commands follow:

attach

Syntax:
`.DS attach .COORD [-co DIST]`
`.DS attach none`

Attaches atomic information of a coordinate dataset to the surface: For each surface point, the atom of the coordinate dataset `.COORD` will be assigned to it - using the following rules:

1. The atom is unrestricted (see 4.3.1 on page 24)
2. It is closer in space than any other atom but not farther away than `DIST` Å (default 3Å).

If no atom fulfills the above criteria, the assignment is not modified; as a result, multiple attachment commands are cumulative, and it should be noted that mixing different coordinate datasets it possible. The assignements are removed by specifying the parameter `none` instead of a coordinate dataset.

As a consequence of the attachment, the properties of each surface point are extended to contain its attached atom properties, which in turn may be used for selection and range statements.

4.6.4 Object Commands

For the generic commands `get`, `hide`, `show`, `set`, `renew`, `render`, and `material` see 4.3.2 on page 25. Specific surface object commands follow:

reverse

Syntax: **`.DS.OBJ reverse`**

Inverses all surface normals. As a consequence, inside and outside for this surface object are swapped.

4.6.5 Dataset, Object and Data-Unit Properties

Tables 4.10 on the following page and 4.11 on the next page list all the dataset, object and data-unit properties that are accessible with the dataset and object commands `set` and/or `get`, as well as available during selection or range statements. Note that for attached surface datasets, the selectable properties of the coordinate data-units are also available (table 4.5).

Property	*S*	*G*	Description
Dataset Properties			
`center`	•	•	Geometric center of dataset
`rcen`	•	•	Center of rotation, default is the geometric center
`rot`	•	•	3x3 rotation matrix, default is identity
`rtc`	•	•	4x4 compact rotation, translation and center matrix
`smode`	•	•	Selection mode, can be either `any` (default) or `all`
`trans`	•	•	Translation vector, default is `{0,0,0}`
Object Properties			
`center`		•	Geometric center of object

S: Property can be modified with `set`
G: Property can be obtained with `get`

Table 4.10: Surface Dataset and Object Properties

Property	*S*	*G*	*L*	*R*	Description
data-unit properties shared between dataset and objects					
`x`			•	•	X coordinate of surface point
`y`			•	•	Y coordinate of surface point
`z`			•	•	Z coordinate of surface point
special			•	•	All *shared* coordinate datas-unit properties (table 4.5 on page 31) can be used with an attached surface.
data-unit properties copied from dataset to object					
`color`	•			×	Color of surface point

S: Property can be modified with set
G: Property can be retrieved with get
L: Property can be used in a selection statement
R: Property can be used in a range statement after `-range` (•) or as the property to be set (×)

Table 4.11: Surface Data-Unit Properties

4.6.6 Render Modes and Properties

The render properties (in `typewriter` font) mentioned below are listed in table 4.9 on page 36.

fill (default) The faces are rendered as filled and lit triangles.

lines Only the outlines of the triangular faces are drawn as lines of width `lw`.

dots Only the vertices are draw as points of size `ps`.

Property	Description	Default
`light1` `light2`	Mode that determines wether the surface is lit on the outside (`light1`) or from both sides (`light2`).	light1
`lw`	Linewidth for rendering mode `line`	1.0
`ps`	Pointsize for rendering mode `dots`	1.0
`solid`	Flag that determines wether the surface is considered solid, in which case the interior is filled with `solidc`.	false
`solidc`	Color of solid interior	{1,1,1}
`t`	transparency: 1.0 is fully opaque, 0.0 is fully transparent	1.0

Table 4.12: Surface Object Render Modes and Properties

4.7 Topograph Dataset

4.7.1 Basic Data-Unit

The topograph dataset consists of a two dimensional rectangular grid (dimensions u x v), with a scalar height value at each grid-point. The rectangular grid corners are initially located at positions $(-0.5\ -0.5\,0)$ and $(0.5\,0.5\,0)$ (independent of the number of grid points), the initial coordinates of the grid points calculated accordingly. The height is always in the range of 0.0 to 1.0, the internal scaling depends on the precision of the input file - for 8bit greyscale values, this is the range from 0 to 255.

Each grid point is assigned a 3D position: x- and y-coordinate are obtained from the initial position on the grid multiplied with `scalexy`, the z-coordinate equals the height multiplied with `scalez`.

4.7.2 Object Types

surface Continuous surface protruding from the grid plane, approximated with triangles, based on the 3D position of each grid point and its adjacent ones. The sampling of the grid is controlled with `step`.

contour Contour lines at equal heights, from `lstart` to `lend` with `lstep` sized steps, protruding from the grid plane. The sampling of the grid is controlled with `step`.

4.7.3 Dataset Commands

For the generic commands `del`, `fix`, `get`, `grab`, `new`, `reset`, `restrict`, `rot`, `set`, and `trans` see 4.3.1 on page 24. Special topograph dataset commands follow:

attach

Syntax:
`.DS attach .COORD [-co DIST]`
`.DS attach none`

Attaches a topograph dataset to a coordinate dataset. Funtionally identical to the equivalent command of the surface dataset (see page 37).

tex

Syntax: **`.DS tex FILE [-name TEXNAME]`**

Defines an image to be used as a surface texture. If the name is omitted then the base of the filename will be used.

4.7.4 Object Commands

For the generic commands `get`, `hide`, `show`, `set`, `renew`, `render`, and `material` see 4.3.2 on page 25. Specific topograph object commands follow:

map

Syntax: **`.DS.obj map TEXNAME`**

Maps texture named TEXNAME onto surface, only valid for object type `surface`. The texture is *modulated* with the underlying surface: The color components at each point will be multiplied, and the transparency is preserved. Only one texture can be mapped onto a `surface` object at a time.

unmap

Syntax: **`.DS.obj unmap`**

Removes the current texture from the object.

4.7.5 Dataset, Object and Data-Unit Properties

Tables 4.13 and 4.14 on the next page list all the dataset, object and data-unit properties that are accessible with the dataset and object commands `set` and/or `get`, as well as available during selection or range statements. Note that for attached topograph datasets, the selectable properties of the coordinate data-units are also available (table 4.5).

4.7.6 Render Modes and Properties

Object type surface:

fill (default) The faces are rendered as filled and lit triangles.

lines Only the outlines of the triangular faces are drawn as lines of width `lw`.

dots Only the vertices are draw as points of size `ps`.

Object type contour:

lines (default) The contour lines are displayed, with a width of `lw`.

dots Only the endpoints of the lines are displayed, with a size of `ps`.

Property	S*	G*	Description
Dataset Properties			
`center`	•	•	Center of rotation, default is the geometric center
`rot`	•	•	3x3 rotation matrix, default is identity
`rtc`	•	•	4x4 compact rotation, translation and center matrix
`scalexy`	•	•	Scaling factor in xy plane
`scalez`			Scaling factor in z
`trans`	•	•	Translation vector, default is `{0,0,0}`
Object Properties for type surface			
`center`		•	Geometric center of object
`step`	•	•	
Object Properties for type contour			
`center`		•	Geometric center of object
`level`	•	•	Contour level (single level contouring), ranges from 0 to 1
`lstep`	•	•	For multi-level contouring, stepsize of contour levels
`lstart`	•	•	For multi-level contouring, start of contour levels
`lend`	•	•	For multi-level contouring, end of contour levels
`step`	•	•	Stepsize within grid, default is 1 (every grid point)

*S: Property can be modified with `set`
*G: Property can be obtained with `get`

Table 4.13: Topograph Dataset and Object Properties

Property	S*	G*	L*	R*	Description
data-unit properties shared between dataset and objects					
h		•	•		height between 0 and 1
x		•	•		x-coordinate
y		•	•		y-coordinate
z		•	•		z-coordinate
data-unit properties copied from dataset to object					
color	•			×	color as name or rgb-triplet

*S Property can be modified with set
*G Property can be retrieved with get
*L Property can be used in a selection statement
*R Property can be used in a range statement

Table 4.14: Topograph Data-Unit Properties

Render Properties		
Property	**Description**	**Default**
ps	Point size for render mode dots, object type contour	1.0
lw	Line width for render mode lines, object type contour	1.0
polyf	polygon-offset parameter 1	0
polyu	polygon-offset paramater 2	0
t	transparency for object types contour and slab	1.0

Table 4.15: Topograph Object Render Properties

4.8 Geometric Primitives

4.8.1 Creating dataset and object

This dataset provides the ability to display arbitrary points and lines. It is created with the database manager command `new`:

Syntax: **`new -type geom [-name NAME]`**

If `NAME` is omitted, `geom` is used. There is only one object type that is again created with the dataset command `new`:

Syntax: **`.GEOM new [-name NAME]`**

If `NAME` is omitted, the dataset-name will be used.

4.8.2 Adding primitives

An object of a geometric dataset can hold an arbitrary amount of primitives.

4.8.2.1 Point

add point

Syntax:

`.GEOM.GEOM add point p={x,y,z} [,c=COLOR] [,r=RADIUS] [,t=TRANSP]`

A position must be supplied, a color, radius and transparency can be also given.

Example:

```
.geom.geom add point p={2,1,0},c=red,r=2.0,t=0.5
```

4.8.2.2 Line

add line

Syntax:

`.GEOM.GEOM add line p={{x1,y1,z1},{x2,y2,z2}} [,c=COLOR] [,t=TRANSP]`

A start- and end-position must be given, color and transparency can also be set.

Example:

```
.geom.geom add line p={{0,0,0},{1,0,0}},c=green
```

4.8.3 Other object commands

For the generic commands `get`, `hide`, `show`, `set`, `render`, and `material` see 4.3.2 on page 25. Specific geometric object commands follow:

del

Syntax: **`.GEOM.GEOM del SELECTION`**

Remove primitives from object based on `SELECTION`. This expression is special to the geometric dataset: the individual primitives are named in a particular way, and SELECTION is a list of names to be removed. Points start with `p`, lines with `l`, and then both carry a consecutive number.

list

Syntax: **`.GEOM.GEOM list`**

Lists all primitives with their names and attributes

4.8.4 Properties

In table 4.16, object and render properties are listed that apply to geometric objects.

Object Properties			
Property	Description	Default	Applies to[x]
`c`	color	white	all
`r`	radius	1.0	all
`t`	transparency	1.0	all
Render Properties			
Property	Description	Default	Applies to[x]
`on`	renders points as spheres, lines as cylinders		all
`off`	simple points and lines	X	all
`detail`	spherical and tubular subdivision	3	all
`stipple`	flag to turn stippling on or off	off	L
`stipplei`	amount of visible line segment	0.2	L
`stippleo`	amount of invisible line segment	0.2	L
`tube`	consecutively connects all points as spline (flag)	off	P
[x]P: points, L:lines			

Table 4.16: Properties of geometric objects

5 Exporting the Scene

write

Exports the current scene as a PNG raster image, as a PostScript file or as a POVray scene. While actually a `scene` command, its aliased so `scene` can be ommitted.

5.1 Raster Image

Syntax: **`write FILE.png [-s (SIZE | WxH | VAL%)] [-a ACC]`**

Creates a raster image in png format[1]. The dimensions of the offline rendering area can be set with `-s`: a single value denotes a square area (in pixels), two numbers interspaced with `x` stand for a rectangular area (in pixels), and a percentage value refers to the corresponding relative size of the graphics window. The maximum possible size depends on the X-Windows system, 2000x2000 should work in most cases. If this option is utilized, an offline rendering context is created and the current scene is rendered into it, otherwise the pixel values are taken directly form the OpenGL context of the gfx-window.

On OpenGL systems supporting accumulation buffers, the scene can be subjected to scene antialiasing[2]: The final image will be a blend of `ACC` images. Possible values for `ACC` are 2,3,4,5,6,8,9,12 or 16. The higher the number the better the final image will look, but rendering time is increased accordingly.

5.2 PostScript

Syntax: **`write FILE.ps`**

Converts the current scene into a PostScript file. The 3D primitives (points, lines and triangles) are displayed on the screen - after transformation and projection - with x and y coordinates in screen pixels and z coordinate normalized between 0 and 1. These transformed primitives are collected, sorted from back to front and then written as PostScript elements: again points, lines and triangles, just 2D. Several noteworthy points:

1. The resulting scene is described by vectors and hence resolution independent (but see note about triangles below).
2. A complicated scene might take a long time to render because every PostScript element is drawn, even if finally completely covered by other ones.
3. Transparency is not possible with PostScript, any transparent object will be opaque.
4. Depth cueing does not work.
5. Postscript triangles are unicolored. DINO converts its gouraud shaded trianges into several unicolored ones[3]. This approximation might become apparent at high magnification.
6. The BoundingBox is crudely set to the the window dimensions and might need adjustment.
7. Several object types cannot be exported.

PostScript export has been largely superseded by the POVray format (see below).

[1] PNG provides an open-source, royalty-free, efficient graphics format. See `http://www.libpng.org/`

[2] from the OpenGL Programming Guide 3rd Edition

[3] using the freely available `gouraudtriangle` macro written by Frederic Delhoume (delhoume@ilog.fr)

5.3 POVray

Syntax: **write FILE.pov [(-patch | -smooth | -v35)] [-nocolor]**

Exports the current scene as a POVray scene. The *Persistence of Vision Raytracer* (`www.povray.org`) is an extremely powerful tool to create stunningly beautiful raytraced pictures. Its freely available for all major platforms and is constantly being upgraded and improved upon by a large community of enthusiastic users and developers. An extensive tutorial on the DINO homepage (`www.dino3d.org`) is devoted to POVray output and a duplication here would exceed the scope of this reference manual.

A note on the options. The first three concern color-interpolation across a triangular face:

-patch use a custom color-triangle that requires a patched version of POVray to be parsed correctly.

-smooth simulate color-interpolation across triangle by using a special texture.

-v35 export for POVray version 3.5 (default is 3.1g), which includes support for color-interpolation across triangles.

The flag **-nocolor** will write uni-colored objects that can be colored by setting an explicit color in the generated output file.