Contents

Index

Alphabetical

1 Modules

1.1 basic

NAME

basic

DESCRIPTION

This module implements the basic object handling functions. The module depends and builds on top of the "property" module, where the object property handling is implemented. However most of the functions in this module handle the validation, creation and deletion of the basic objects. The basic objects are: points, lines or links, triangles, quads, tetrahedrons, blocks and info objects. All of the objects are graphical objects except the info objects. Although info objects can also be displayed. The graphical objects depend on points. This means that to define a link object first two points must be defined. For triangles three for quads four points must be define, and so on. By default the points are visible but they can be switched off, so they do not disturb the view of the model. The objects can define other dependencies as well, for example when a triangle is formed from three link objects, then it is possible to specify that the link objects cannot be deleted until the triangle object exists.

Object referencing

When objects are referencing each other they use a two element list, a pair. The first element in the list identifies the type of the object and the second element is the identification number of the object. The first element is a symbol. For example '(p 2) denotes a point with ID number 2. This kind of object reference will be called as **object descriptor**.

1.2 command

NAME

command

DESCRIPTION

This module implements the highest level of functions. The functions in the module are the basic commands that are available to the user. Generally the functions here setup some kind of selection mechanism, however these functions do not act upon the user input directly. The user response is processed by functions in the "operation" module.

1.3 entry

NAME

entry

DESCRIPTION

The functions in this module handle the text input of the user.

1.4 error

NAME

error

DESCRIPTION

The functions in this module can be used to print error and warning messages. The function ensures that these messages are uniform.

1.5 gui-gl

NAME

gui-gl

DESCRIPTION

This module is an OpenGL binding layer. The module only contains the functions required by the program. The use of this module is necessary, since every scheme implementation creates a slightly different binding for the OpenGL library.

1.6 gui-widgets

NAME

gui-widgets

DESCRIPTION

This module contains all the required GUI functions. The implementation of these functions are very much dependent on the scheme implementation therefore it should be adapted for scheme. The module also contains some scheme specific functions, where for example the standard does not specify any behaviour, but every scheme implementation does.

This file contains the PLT-Scheme specific implementation of the functions.

1.7 io-native

NAME

io-native

DESCRIPTION

This module implements the handling of the native file format of SX. The main reason for the creation of this file format, instead of using some other file format is to have the internal object database in a very simple format which is easy to debug or correct.

The comment character is the hash ('#') on the first non-white space position of the line. The format is forgiving in one respect, that once the required data has been read in anything in the rest of the line is ignored. For example it is valid to say in the header:
`*point\aaUnder{}unique*  \aaSmaller{}integer\aaLarger{}  hello  world`

EXAMPLE

```
Format description:

<header>
<object-list>

where :

<header> =
  sx-1.0-format
    <parameters>
  end

<parameters> =
  *point_unique*  <integer>
  *link_unique*   <integer>
  *triang_unique* <integer>
  *quad_unique*   <integer>
  *tetrah_unique* <integer>
  *block_unique*  <integer>
  *info_unique*   <integer>
and optionally any of the following:
  *gl_zoom*                <double>
  *gl_rotation_quaternion* <double> <double> <double> <double>
  *gl_rotation_x*          <double>
  *gl_rotation_y*          <double>
  *gl_rotation_z*          <double>
  *gl_pan_x*               <double>
  *gl_pan_y*               <double>
  *gl_center*              <double> <double> <double>


<object-list> = <object> <object> ... <eof>


<object> =
  <point> |
  <link> | <triang> | <quad> | <tetrah> | <block> |
  <info>


<point>  =
  p <integer>
    child <object-desc> ...
    parent <object-desc> ...
    color <colors>
    style <style>
    coords <double> <double> <double>
  end


<link> | <triang> | <quad> | <tetrah> | <block> =
  <object-name> <integer>
    child <object-desc> ...
    parent <object-desc> ...
```

```
    color <colors>
    style <style>
    nodes <integer> <integer> ...
  end


<info> =
  i <integer>
    child <object-desc> ...
    parent <object-desc> ...
    color <colors>
    style <style>
    <name> <something-list>
    ...
  end

<object-name> = l | t | q | e | b

<object-desc> =
  p <integer> |
  l <integer> |
  t <integer> |
  q <integer> |
  e <integer> |
  b <integer> |
  i <integer>

<colors> =
  <integer> |
  <double> <double> <double> <double>
    red     green     blue     alpha

<style> =
  <style-keyword> |
  <style-keyword> <double>

<style-keyword> =
  hide |
  opaq |
  sides |
  transparent |
  lines

<name> = a name to identify this property

<something-list> =
  <somthing> <something-list>

<something> =
  "sztring"
  <symbol>
  <integer>
  <double>
```

1.8 main

NAME

main

DESCRIPTION

This is the main module of the SX modeller. The module exports the "sx-main" function, which carries out all the necessary setup and initialisation processes required by any program based on the SX modeller.

1.9 menu

NAME

menu

DESCRIPTION

This module implements the handling of menus. The module exports only two functions. One of the functions is used to initialise the menu bar, the other function loads a menu file.

Format of the menu file.

A comment line starts with the '#' character as the first, non whitespace character in the line. The order of the definition of the menus specifies the order of appereance, not only in the menu bar but in the popdown menu list as well. The menu bar holds several menu lists. A menu list is defined as:

```
*MENU name
[item1] command1 option1 option2 ...
[item2] command2 option3 option4 ...
*ENDMENU
```

A menu list must start with a line whose first word is '`*MENU`'. The menu list is terminated by a line containing the '`*ENDMENU`' word and nothing else (no extra whitespaces). `name` must be a single word, without any whitespace, denoting the menu list in the menu bar. A menu item, in the menu list has a name and after the name some commands and options. The menu item name must be enclosed by square brackets and it must contain at least one character. If the menu name contains only two minus signs then a separator is generated in the menu. The commands and options are separated by white spaces (any number of them). It is assumed that after every word in the command and option list an ENTER is issued. If only an ENTER should be issued then use the ';' character. It is possible that no command is specified after the menu item name. In this case the command will be a simple ENTER.

For example to set every element to opaq rendering style:

```
[Opaq]          style all ; opaq ;
```

This menu line uses the 'style' command which requires that first a number of elements is selected to which the style change is applied. In the example line all of the elements are selected by the 'all' keyword. Then to tell the program that we have stopped the selection of elements a simply ENTER must be issued. This is carried out in the menu

by the ';' character. It is important that there are spaces around the ';' character, otherwise the program would interpret it as part of the command. Finally we set the style to opaq and again to terminate the style selection an ENTER must be issued.

1.10 mouse

NAME

mouse

DESCRIPTION

This module implements the handling of mouse actions, like clicking a button or moving the mouse. The module exports only three functions which are the callback functions for the different mouse actions, like pressing or releasing a mouse button and moving the mouse.

1.11 optmenu

NAME

optmenu

DESCRIPTION

This module implements the functions which can create and handle the optional menus in the program. Optional menus appear to the right of the graphical window and they provide optional keywords and choices as buttons. The module also provides a set of default optional menus.

1.12 property

NAME

property

DESCRIPTION

This module contains the basic property handling functions. All objects are represented by several properties. Some of the properties are mandatory and some are optional. All objects types must have 'child', 'parent', 'color' and 'style' properties. Points must also have the 'coords' property. Links, triangles, quadrilaterals, terahedras and blocks (the other geometric objects) must have the 'nodes' property. It is an error when a point has the 'nodes' property or an object which is not a point to have the 'coords' property. The info object, apart from the mandatory properties can have an unlimited number of other properties. The only requirement is that each property is identified by a unique name. It is also important that the values of the properties are stored in a list, even for single value properties.

The properties of an object are stored in a "table". The table could be a vector an association list or a hash table, which depends on the scheme implementation. These property tables are the objects and they are also stored in an object table. One object table stores only one type of object. For example point objects are stored in the *POINTS* table. There is another important variable for each object type, which

describes what was the largest unique number assigned to an object. For example for points the '*POINT_UNIQUE*' variable stores this value. The reason for this variable is that when an element is deleted its position in the table is "kept". So when an undo occurs, the object can be restored directly and references to other objects can be established immediately. Of course this also assumes that deletion is tightly controlled and allowed only when no other object references the object to be deleted. This also means that when the object database is written out, there can be "gaps" in the tables, which are places of the deleted elements. Obviously the database can be compressed and renumbered to remove these gaps if necessary.

There are three sets of functions in this module. The first set of functions is used mostly in the program, which directly manipulates the properties. The second set of functions is built on top of the first set of functions and it can be used when the type of the object and the identification number is available. The third set of functions are capable to record their actions with the undo system, so these changes can be undone later if necessary. High level commands should use this third set of functions.

The functions are implemented in such a way that they rely on a thin application layer, therefore the underlying data structure can be replaced!

Object referencing

Objects can reference each other in a hierarchical way. It is important that there is always a hierarchy between these objects otherwise it is not possible to delete the circularly referenced objects! When an object refers to another object, so this object is at higher level, then it stores the object descriptor of the lower level object in its 'child' property. However at the same time the lower level object stores the higher level object in its 'parent' property. In this way the objects make up several tree structures which can be walked in both direction, from bottom to top or from top to bottom. Several tree structures are mentioned, since separate object hierarhies are allowed.

1.13 render

NAME

render

DESCRIPTION

This module implements all the rendering functions required by the SX program. The rendering of the model is carried out by OpenGL and the selection from screen is also performed by OpenGL.

glSelectBuffer interface

To create the selection buffer use the "sx-select-buffer-create" function. The buffer is stored in the *GL_SELECT_BUFFER*. However the size of the buffer is also stored explicitly in the *GL_SELECT_BUFFER_SIZE* variable. The reason for this explicit storage is that some scheme does not use vectors or lists directly as selection buffers. For example in guile a uniform vector is used, however in bigloo a C array is allocated which cannot be handled directly. Similarly PLT-scheme allocates a gl-unit-vector for the selection buffer and its size must be known by some other means since the vector-length function does not work. For the same reason to access the contents of the selection buffer use the "sx-select-buffer-get" function which returns a vector. The "sx-select-buffer-get" function has three arguments:

- the first denotes the number of picked elements
- the second denotes the size of the selection buffer and
- the third argument stores the selection buffer itself.

The reason for the first argument is that in bigloo (and maybe in other scheme implementations) the returned vector is created on the fly from the selection buffer, therefore in this case the memory allocation should be minimal and the returned vector will have the size of the number of picked objects.

There is one more very important function: "object-gl-selection-buffer". This function is called every time when a graphical selection is to be made, since this function increases the selection buffer as required. The size of the selection buffer should be so large that it is possible to hold all of the objects.

The selection buffer contains hit records of five numbers for each selected object:

- 0 - the number of names in this record, this is always two for the SX program since a record stores the type and the id of the object;
- 1 - minimum projected z coordinate (depth) of the object;
- 2 - maximum projected z coordinate (depth) of the object;
- 3 - number to identify the type of the object (point, line, etc) and
- 4 - the id number of the object

Rendering styles and colors

The rendering style and color are mainly governed by the object itself through the 'color' and 'style' properties. The rendering style can be:

- opaq: same color on both sides, no see through
- sides: ONLY for triangles and quads, different color on the different sides of the object
- transparent: same color, but it is possible to see through the element like glass
- lines: only outline is drawn.

The available rendering styles are stored in the "*GL_STYLE_LIST*" variable. On top of this it is also possible to render the object fully or as tiled. In the later case the size of the object is reduced by a certain percentage towards the center of the object. Rendering is possible with lights and without lights.

The color of an object can be an indexed color stored in the "*GL_COLOR_LIST*" variable or it can be an RGBA value.

1.14 selection

NAME

selection

DESCRIPTION

This module provides functions for selection. In the SX program selection means user input, other than mouse handling, for example: input of an integer, a real number,

a string, a keyword, a point or three coordinate values, an object, a filename or a color.

1.15 trackball

NAME

trackball

DESCRIPTION

This module implements a trackball rotation model.

1.16 undo

NAME

undo

DESCRIPTION

This module provides undo capabilities for a program. The system is similar to the undo facility which can be found in AutoCAD systems. This means:

- unlimited undo;
- one operation can be undone in one step, for example the copying of several points;
- user marks can be placed to mark stages and it is possible to return to them by undoing everything up to that point.

However there is one drawback of this undo system at the moment, that it provides only one level of redo. If several steps has been undone not all of them can be redone only the last one. On the other hand a redone operation can be undone again, if necessary.

From the programmers' point of view there is one more very important feature. To ensure that one command can be undone as one step the "undo-add-op-mark" function must be called at the beginning of the command. This is a special mark, different from the user placable marks. In this case "one command" means all the operations that are performed by a user initiated command. For example if the user copies three points from one place to another, then undoing this command will remove all three points, not only the last of the three points. **It is very important that the operation marks are placed properly, since a single undo ends when the process encounters a user or an operational mark or the undo list becomes empty!**

1.17 view

NAME

view

DESCRIPTION

This module contains functions for basic viewing operations. Functions in this module can reset the rotation angle, the rotation centre, the zooming scale and the panning distance of the model. Moreover the functions can modify the rendering style and the color of the objects.

2 Variables

2.1 *OBJECT_LAST*

NAME

OBJECT_LAST

DESCRIPTION

This variable stores the reference to the last created object as an object descriptor.

2.2 *ENTRY_CALLBACK*

NAME

ENTRY_CALLBACK

DESCRIPTION

This variable stores the callback function for the entry widget. This is a second level of callback function, since actually the "entry-callback-execute" function is tied to the graphical widget. However not all graphical toolkits can replace the callback function with another one, therefore the "entry-callback-execute" function calls the function which is stored in this variable.

2.3 *COMMAND-ABBREV*

NAME

COMMAND-ABBREV

DESCRIPTION

This variable stores the command abbreviation. When a string is executed by the default callback function then first it checks whether the string is a command abbreviation found in this list. The variable contains pairs of strings, where the first string is the abbreviation and the second string is the name of the command. For example the "p" letter can be the abbreviation for the command "point". Any abbreviation can be defined for a command. Thus "point" can be abbreviated as "pp" as well.

2.4 *COMMAND-LIST*

NAME

COMMAND-LIST

DESCRIPTION

This variable stores the available commands. The "entry-add-command" should be used to add a new command to the system. A command must be a pair where the first element is the name of the command and it is a string, while the second element in the list is a procedure with zero arity (no arguments).

2.5 *COMMAND_CACHE*

NAME

COMMAND_CACHE

DESCRIPTION

This variable can store a series of commands which are executed after another by the entry. Using this variable the menus can issues several consequtive commands. The variable can be set by the "entry-command-cache-set!" function. Its default value is an empty list. When it is not an empty list then all elements must string.

2.6 *COMMAND_LAST*

NAME

COMMAND_LAST

DESCRIPTION

This variable stores the last executed command. If no command was executed yet then ist value is #f. NOTE This is a private variable, the user cannot modify directly

2.7 sx-version

NAME

sx-version

DESCRIPTION

This variable stores the version number of the SX modeller.

2.8 *MENU_BAR*

NAME

MENU_BAR

DESCRIPTION

This variable stores the graphical menubar.

NOTES

This is a private variable and cannot be modified by the user directly.

2.9 *MENU_LIST*

NAME

MENU_LIST

DESCRIPTION

This variable stores the list of menus attached to the menubar. The list can be used to remove the menus from the menubar when a new menu is loaded.

NOTES

This is a private variable and cannot be modified by the user directly.

2.10 *MOUSE-START-X*

NAME

MOUSE-START-X

DESCRIPTION

This variable stores the x position of the mouse when the mouse button has been pressed. NOTE This is a private variable and cannot be modified by the user directly.

2.11 *MOUSE-START-Y*

NAME

MOUSE-START-Y

DESCRIPTION

This variable stores the y position of the mouse when the mouse button has been pressed. NOTE This is a private variable and cannot be modified by the user directly.

2.12 *MOUSE-CURSOR-PICK-SIZE*

NAME

MOUSE-CURSOR-PICK-SIZE

DESCRIPTION

This variable specifies the size of the picking region when the mouse cursor is used for picking. NOTE This is a private variable and cannot be modified by the user directly.

2.13 *MOUSE-DRAGGING-HANDLER*

NAME

MOUSE-DRAGGING-HANDLER

DESCRIPTION

This variable stores a function which is called when the mouse is moving with one of the mouse buttons pressed. In other words, the stored function handles the mouse dragging. If no dragging is allowed this variable stores #f. NOTE This is a private variable and cannot be modified by the user directly.

2.14 *ROTATION-DIRECTION*

NAME

ROTATION-DIRECTION

DESCRIPTION

The rotation of the model can be performed by quaternions or by x and y rotations. In the first case, when quaternions are used, the rotation of the model can be restricted, so the model will only rotate around the x or the y axis. This variable stores the symbol 'x' when in quaternion mode the only rotation allowed is around the x axis, and it stores the 'y' symbol when the only rotation allowed is around the y axis. In all other cases, when there is no restriction on the rotation, the variable stores #f. NOTE This is a private variable and cannot be modified by the user directly.

2.15 *OPTION_<menu>*

NAME

OPTION_<menu>

DESCRIPTION

This is the description for: *OPTION_MENU*, *OPTION_CURRENT*, *OPTION_EMPTY* *OPTION_SELECT*, *OPTION_POINT*, *OPTION_YES_NO*, *OPTION_VIEW*, *OPTION_CENTER*, *OPTION_RESET*, *OPTION_STYLE*, *OPTION_APPEAR_ALL*. These variables store the created, graphical widgets for the specific optional menu.

2.16 *PROP_INTERNAL*

NAME

PROP_INTERNAL

DESCRIPTION

This variable stores the names of the internal properties. These properties should not be modified by the user explicitly, only through provided functions.

NOTES

This is a private variable, cannot be modified in the program.

2.17 *<object>_UNIQUE*

NAME

<object>_UNIQUE

DESCRIPTION

This description relates to the following variables: *POINT_UNIQUE* *LINK_UNIQUE*, *TRIANG_UNIQUE*, *QUAD_UNIQUE*, *TETRAH_UNIQUE*, *BLOCK_UNIQUE* and *INFO_UNIQUE*. These variables store the identification number of an object which can be assigned to a new object. These numbers are never decreased.

2.18 *<object>_DB*

NAME

<object>_DB

DESCRIPTION

These variables store the object databases. In the databases the objects are identified by a key. The key is an identification number. The value related to a key is the object itself.

2.19 *POINT_MAX*

NAME

POINT_MAX

DESCRIPTION

This variable stores the maximum extent of the current 3D model in the absolute sense. In other words, considering all points and all coordinate directions this variable stores the maximum absolute value of all values.

2.20 *GL_SELECT_BUFFER*

NAME

GL_SELECT_BUFFER

DESCRIPTION

This variable stores the OpenGL selection buffer. It is implementation dependent what is the value of the variable. In guile it stores a vector, in bigloo it stores a C array and in PLT-scheme it stores gl-uint-vector.

2.21 *GL_SELECT_BUFFER_SIZE*

NAME

GL_SELECT_BUFFER_SIZE

DESCRIPTION

This variable stores the size of the OpenGL selection buffer. Its default value is zero.

2.22 *GL_SELECT_BUFFER_ALLOC_UNIT*

NAME

GL_SELECT_BUFFER_ALLOC_UNIT

DESCRIPTION

This variable stores the allocation unit used for the allocation of the OpenGL selection buffer. Basically the size of the buffer is increased by the amount specified by this variable.

2.23 *GL_VIEWPORT_WIDTH*

NAME

GL_VIEWPORT_WIDTH

DESCRIPTION

This variable stores the width of the graphical area. It is always updated, when the window is resized.

2.24 *GL_VIEWPORT_HEIGHT*

NAME

GL_VIEWPORT_HEIGHT

DESCRIPTION

This variable stores the height of the graphical area. It is always updated, when the window is resized.

2.25 *GL_CENTER*

NAME

GL_CENTER

DESCRIPTION

This variable stores the centre of rotation during viewing. Initially it stores the coordinates of the origo.

2.26 *GL_ROTATION_MODE*

NAME

GL_ROTATION_MODE

DESCRIPTION

This variable controls that during viewing how the mouse movement influences the rotation of the model. The possible values are: "quaternion" and "xy".

2.27 *GL_ROTATION_X*

NAME

GL_ROTATION_X

DESCRIPTION

When "*GL_ROTATION_MODE*" is set to "xy" then this variable stores the rotation around the x axis.

2.28 *GL_ROTATION_Y*

NAME

GL_ROTATION_Y

DESCRIPTION

When "*GL_ROTATION_MODE*" is set to "xy" then this variable stores the rotation around the y axis.

2.29 *GL_ROTATION_Z*

NAME

GL_ROTATION_Z

DESCRIPTION

When "*GL_ROTATION_MODE*" is set to "xy" then this variable stores the rotation around the z axis.

2.30 *GL_ROTATION_QUATERNION*

NAME

GL_ROTATION_QUATERNION

DESCRIPTION

When "*GL_ROTATION_MODE*" is set to "quaternion" then this variable stores the current quaternion. Its initial value is (0.0 0.0 0.0 1.0).

2.31 *GL_ZOOM*

NAME

GL_ZOOM

DESCRIPTION

This variable controls the zooming factor. Initially its value is 1.0.

2.32 *GL_PAN_X*

NAME

GL_PAN_X

DESCRIPTION

This variable controls the panning (displacement of the model during viewing) in the x direction.

2.33 *GL_PAN_Y*

NAME

GL_PAN_Y

DESCRIPTION

This variable controls the panning (displacement of the model during viewing) in the y direction.

2.34 *GL_RENDER_LIGHTING*

NAME

GL_RENDER_LIGHTING

DESCRIPTION

This variable controls whether lighting is used during rendering or not. Its default value is #t, to indicate that the light is on.

2.35 *GL_COLOR_SELECTION*

NAME

GL_COLOR_SELECTION

DESCRIPTION

This variable stores a color description (in RGBA format) which is used when selected elements are rendered. Its initial value is red. NOTE At the moment there is no way to change this during execution.

2.36 *GL_COLOR_BACK_FACE*

NAME

GL_COLOR_BACK_FACE

DESCRIPTION

This variable stores a color description (in RGBA format) which is used when triangle and quad objects are rendered with the "sides" style. This color is used to render the backface of the elements. NOTE At the moment there is no way to change this during execution.

2.37 *GL_COLOR_LIST*

NAME

GL_COLOR_LIST

DESCRIPTION

This vector contains the actual RGBA color representation of all the available indexed colors.

2.38 *GL_STYLE_LIST*

NAME

GL_STYLE_LIST

DESCRIPTION

This variable stores the list of currently available rendering styles. The values in the list are:

- hide - the object is not rendered
- opaq - the object is rendered as solid
- sides - the two sides of triangle and quad objects are rendered with different colors
- transparent - the object is rendered as transparent, it is possible to see through the object, like through a glass surface
- lines - the object outline is rendered as a set of lines

2.39 *OBJECT_<object>_ID*

NAME

OBJECT_<object>_ID

DESCRIPTION

This is the description of the following variables: *OBJECT_POINT_ID* *OBJECT_LINK_ID*, *OBJECT_TRIANG_ID*, *OBJECT_QUAD_ID*, *OBJECT_TETRAH_ID*, *OBJECT_BLOCK_ID*, *OBJECT_INFO_ID*. These variables store a unique integer number for each object type. The integer numbers are loaded as OpenGL names when rendering the objects in selection mode.

2.40 *ENTRY_DEFAULT_TEXT*

NAME

ENTRY_DEFAULT_TEXT

DESCRIPTION

This variable stores the default text that is written out as a prompt.

2.41 *DIALOG_WINDOWS*

NAME

DIALOG_WINDOWS

DESCRIPTION

All dialog windows that are opened should register in this variable. When this variable is not an empty list then the ESC button cannot stop the selection.

2.42 *PICK_MODE*

NAME

PICK_MODE

DESCRIPTION

This variable determines how the program reacts to the clicking of a mouse button. The possible values are:

- view - Program is in viewing mode, by clicking the button the model can be rotated, moved and zoomed
- select_coord - Program is in coordinate selection mode, by clicking the mouse button a point coordinate can be selected
- select_obj - Program is in object selection mode, by clicking the mouse button objects can be selected
- select_cross - Program is in object selection mode, by clicking the mouse button objects can be selected by a rectangular area

2.43 *SELECT_KEYWORDS*

NAME

SELECT_KEYWORDS

DESCRIPTION

When the user does not enter the expected type of input (for example a point) then the entered text is checked against the keywords stored in this variable as a list. These keywords apply only to the next user-input function call. A keyword is stored as a two element list, where the first element in the list is the full keyword while the second element is the shortcut. For example: ("Origo" "O")

2.44 *SELECT_INIT*

NAME

SELECT_INIT

DESCRIPTION

This variable stores control "bits" in a list that specify the behaviour of the next user-input function. The possible values are:

- no_negative - No negative number can be entered
- no_zero - No zero number can be entered
- no_empty - An input must be specified
- no_multiple - Only a single object can be selected
- type_point - Only point object can be selected
- type_link - Only link object can be selected
- type_triang - Only triangle object can be selected
- type_quad - Only quad object can be selected
- type_tetrah - Only tetrahedron object can be selected
- type_block - Only block object can be selected
- file_open - File selection dialog to open a file
- file_save - File selection dialog to save a file

2.45 *SELECT_STATE*

NAME

SELECT_STATE

DESCRIPTION

This variable stores the result of the selection. Possible values are:

- init - the selection mechanism has been started
- cancel - nothing is selected, cancel the selection
- quit - quit the selection
- objects - one or more objects are selected, they are stored in the *SELECT_LIST* variable
- point - a point coordinate is selected, the x, y and z coordinates are stored in the *SELECT_X*, *SELECT_Y* and *SELECT_Z* variables
- integer - an integer is entered, the number is stored in the *SELECT_ANSWER* variable
- real - a real number is entered, the number is stored in the *SELECT_ANSWER* variable

- keyword - a keyword is entered, the string is stored in the *SELECT_ANSWER* variable
- dist - a distance is entered or selected, the number is stored in the *SELECT_ANSWER* variable

2.46 *SELECT_OBJ_MODE*

NAME

SELECT_OBJ_MODE

DESCRIPTION

This variable controls the object selection mode. The default value is "invert" as a symbol. It means that by selecting an unselected object the object becomes selected and by selecting an already selected object the object becomes unselected. Thus it inverts the selection. Other possible values for this variable are: add and remove. When the value of this variable is "add" then by selecting an unselected object the object becomes selected, however by selecting an already selected object the object remains selected. When the value is "remove" the objects become unselected.

2.47 *SELECT_PREFIX*

NAME

SELECT_PREFIX

DESCRIPTION

This variable stores a one letter symbol which is used during object selection. When only a number is entered by the user then the number is taken as an object identification number and the type of the object is determined by this variable. The default value is: p, for point selection.

2.48 *SELECT_TEXT_PREFIX*

NAME

SELECT_TEXT_PREFIX

DESCRIPTION

This variable stores the text that appears in the prompt. This text guides the user during the selection. For example: Select object(s). The text is printed repeatedly until the user specifies a valid input.

2.49 *SELECT_CLOSEST*

NAME

SELECT_CLOSEST

DESCRIPTION

This variable controls whether to select only the closest object or select all objects which are "penetrated" by the line under the cursor. The variable is accessed by the render, mouse and selection modules.

2.50 *SELECT_WIN_CORNER_1*

NAME

SELECT_WIN_CORNER_1

DESCRIPTION

When objects are selected by a rectangular area this variable stores the coordinates of the first corner of the area.

2.51 *SELECT_WIN_CORNER_2*

NAME

SELECT_WIN_CORNER_2

DESCRIPTION

When objects are selected by a rectangular area this variable stores the coordinates of the second corner of the area. This variable is contiuously changing as the mouse is moving around the screen.

2.52 *SELECT_MOUSE_OBJ*

NAME

SELECT_MOUSE_OBJ

DESCRIPTION

This variable is used to pass the objects selected by the mouse to the entry widget.

2.53 *SELECT_X*,*SELECT_Y*,*SELECT_Z*

NAME

SELECT_X*,*SELECT_Y*,*SELECT_Z

DESCRIPTION

These variables store the selected point coordinates.

2.54 *SELECT_BASE*

NAME

SELECT_BASE

DESCRIPTION

The variable stores the base point or the first point of the distance.

2.55 *SELECT_ANSWER*

NAME

SELECT_ANSWER

DESCRIPTION

This variable stores the result of the user input. It may store an integer, a real number, a string or a keyword. The value depends on the *SELECT_STATE* variable.

2.56 *SELECT_LIST*

NAME

SELECT_LIST

DESCRIPTION

The variable stores the selected objects at the end of the selection.

2.57 *SELECT_PREVIOUS*

NAME

SELECT_PREVIOUS

DESCRIPTION

The variable stores the previous, accepted selection of objects. During the next selection of objects this previous selection can be reused.

2.58 *SELECT_ACTION*

NAME

SELECT_ACTION

DESCRIPTION

This variable stores the function that is called when the user has finished the input.

2.59 *SELECT_USER_TYPE*

NAME

SELECT_USER_TYPE

DESCRIPTION

This variable also modifies the behaviour of the next user-input function as the *SELECT_INIT* variable. However this variable can only restrict the selectable objects. To allow to select any of the object types the value of the variable must be "any". To allow the selection of only one type of object the value can be: p, l, t, q, e, b, i. The variable is a further restriction that is controlled by the *SELECT_INIT* variable.

2.60 *SELECT_USER_SINGLE*

NAME

SELECT_USER_SINGLE

DESCRIPTION

The value of the variable can be #t or #f. When the value is #t it means that the user has restricted the selection to a single object. Otherwise there is no restriction and any number of objects can be selected.

2.61 *SELECT_STACK*

NAME

SELECT_STACK

DESCRIPTION

When a selection mechanism calls another selection, it can push its state variables on a stack. This variable stores the stack.

2.62 *SELECT_VIEW_SETUP*

NAME

SELECT_VIEW_SETUP

DESCRIPTION

This variable stores the function that is called when the selection mechanism wants to setup viewing.

2.63 *KEYWORDS_OBJECT_SELECTION*

NAME

KEYWORDS_OBJECT_SELECTION

DESCRIPTION

This variable stores the internal keywords that are defined by the object selection mechanism. Other, external keywords can be defined through the "select-init" function.

2.64 *KEYWORDS_POINT_SELECTION*

NAME

KEYWORDS_POINT_SELECTION

DESCRIPTION

This variable stores the internal keywords that are defined by the coordinate selection mechanism. Other, external keywords can be defined through the "select-init" function.

2.65 *UNDO*

NAME

UNDO

DESCRIPTION

This variable stores all of the operations registered with the undo system. The variable basically stores a function name and arguments which if called will undo the effect of the corresponding command. The list also contains the user and operation marks. NOTE This is a private variable and cannot be modified directly outside of the undo module only through functions.

2.66 *REDO*

NAME

REDO

DESCRIPTION

This variable stores the operations which can be redone after an undo operation. The variable stores only one level of operations. This means for example, that if a copy and a move operations are undone, then it possible to redo only the last move operation and not the copying. NOTE This is a private variable and cannot be modified directly outside of the undo module only through functions.

2.67 *CHANGED*

NAME

CHANGED

DESCRIPTION

This variable stores a flag indicating whether the model has changed or not. The variable is also used by the quit command determining whether to ask the user, since the changed model is not saved, or simply quit. NOTE This is a private variable and cannot be modified directly outside of the undo module only through functions.

3 Functions

3.1 object-clear-all

NAME

object-clear-all

SYNOPSIS

```
(basic-object-clear-all)
```

DESCRIPTION

This function removes all objects from the model, clears the undo and the redo lists and reset the *OBJECT_LAST* variable.

ARGUMENTS

None

RETURN VALUE

None

3.2 basic-object-valid?

NAME

basic-object-valid?

SYNOPSIS

```
(basic-object-valid? obj)
```

DESCRIPTION

This function validates an object. An object is valid if its ID number is smaller than the maximum ID number for the specific object type, defined by *<object>_UNIQUE* variables and if the object exists in the database.

ARGUMENTS

obj – the object to validate. The object is identified by a pair, a symbol of the type and an integer id number, for example '(l 4) denoting link 4 object.

RETURN VALUE

It returns #t if the object is valid otherwise it returns #f.

3.3 object-property-print

NAME

object-property-print

SYNOPSIS

```
(basic-property-print obj)
```

DESCRIPTION

This function assembles a string whic describes the specified object. For all objects it puts the name and the identification number of the object into the string. For points it also attaches the coordinates to the string. For links, triangles, quads and tetrahedrons it also attaches the node numbers to the string. This function is based on the "object-type-string" function.

ARGUMENTS

obj – it is an object descriptor, e.g. '(l 4)

RETURN VALUE

It returns a string describing the object.

3.4 basic-property-get

NAME

basic-property-get

SYNOPSIS

```
(basic-property-get type id)
```

DESCRIPTION

This is an auxiliary function for the "basic-object-duplicate". The function returns the user defined properties, the "style" and the "color" properties of an object. These are the properties that are duplicated for an object.

ARGUMENTS

type – the type of the object, as a symbol
id – the identification number of the object

RETURN VALUE

It returns a property list

NOTES

This is not a public function. It cannot be used outside of the module.

3.5 basic-object-duplicate-aux

NAME

basic-object-duplicate-aux

SYNOPSIS

```
(basic-object-duplicate-aux type id node_assoc)
```

DESCRIPTION

This is an auxiliary function for the "basic-object-duplicate". When this function is called the new duplicate points are already created and their numbers are stored in a hash table, where the key is the old node number and the value is the new node number. In this way when this function wants to create the new, duplicate elements using the new nodes but the old topology the function can easily determine the node number.

ARGUMENTS

type – the type of the object, as a symbol
id – the identification number of the object
node_assoc – the node hash table

RETURN VALUE

It returns an object descriptor of the newly created element

NOTES

This is not a public function. It cannot be used outside of the module.

3.6 basic-object-duplicate

NAME

basic-object-duplicate

SYNOPSIS

```
(basic-object-duplicate proc objects)
```

DESCRIPTION

This function can be used to duplicate a set of objects by higher level functions. The coordinates of the points in the selected objects are modified by a user specified function. The identification number of the modified points are stored in a has table where the key is the old node number and the value is the new node number. Once the modified points are created then all other non-point objects are created by the "basic-object-duplicate-aux" function. The "basic-object-duplicate-aux" function preserves the original topology, but uses the new points in the creation of the non-point elements.

ARGUMENTS

proc – this argument defines a function which will operate on the points in the object set. For example when points are copied by a distance, then this function adds the displacement vector to all of the points in the set. The function has one argument which is the coordinate of an original point and it returns the coordinates of the new point.

objects – a hash table of object descriptors to duplicate

RETURN VALUE

None.

3.7 basic-point-list-centre

NAME

basic-point-list-centre

SYNOPSIS

```
(basic-point-list-centre points)
```

DESCRIPTION

This point determines the gravity centre of a set of points. It is assumed that all points have the same mass.

ARGUMENTS

points – a list of point identification numbers

RETURN VALUE

If the argument is an empty list it returns the origo, otherwise it returns the gravity centre of the points.

3.8 basic-point-all-centre

NAME

basic-point-all-centre

SYNOPSIS

```
(basic-point-all-centre)
```

DESCRIPTION

This function determines the gravity centre of all points in the model. It is assumed that all points have the same mass.

ARGUMENTS

None.

RETURN VALUE

If there are no points in the model it returns the origo, otherwise it returns the gravity centre of the points.

3.9 basic-point-valid?

NAME

basic-point-valid?

SYNOPSIS

```
(basic-point-valid? id)
```

DESCRIPTION

This function validates a point. A point is valid if its ID number is smaller than *POINT_UNIQUE* and if the point exists in the database.

ARGUMENTS

id – the identification number of the point

RETURN VALUE

It returns #t if the point is valid otherwise it returns #f.

3.10 basic-point-add

NAME

basic-point-add

SYNOPSIS

```
(basic-point-add id coords properties undo)
```

DESCRIPTION

This function creates a point and adds it to the current model.

ARGUMENTS

id – a point identification number, however it can be #f when the function will use the current value of the *POINT_UNIQUE* variable as the identification number. In this later case the function also increases the *POINT_UNIQUE* variable.
coords – the coordinates of the point. It is a list of three numbers.
properties – properties, other than coords to set for the point. If there is no extra property then its value is #f.
undo – its value is #t when the adding of the point should be registered with the undo system and therefore it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created point.

3.11 basic-point-del

NAME

basic-point-del

SYNOPSIS

```
(basic-point-del id undo)
```

DESCRIPTION

This function deletes a point from the model. Generally when a point is deleted then its place is kept in the *POINTS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is a point identification number
undo – its value is #t when the deletion of the point should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.12 basic-link-valid?

NAME

basic-link-valid?

SYNOPSIS

```
(basic-link-valid? id)
```

DESCRIPTION

This function validates a link object. A link object is valid if its ID number is smaller than *LINK_UNIQUE* and if the link object exists in the database.

ARGUMENTS

id – the identification number of the link object

RETURN VALUE

It returns #t if the link is valid otherwise it returns #f.

3.13 basic-link-add

NAME

basic-link-add

SYNOPSIS

```
(basic-link-add id nodes properties undo)
```

DESCRIPTION

This function creates a link object and adds it to the current model.

ARGUMENTS

id – a link identification number, however it can be #f when the function will use the current value of the *LINK_UNIQUE* variable as the identification number. In this later case the function also increases the *LINK_UNIQUE* variable.

nodes – the nodes of the link object. It is a list of two integer numbers, which are also valid, point identification numbers.

properties – list of properties, other than nodes, to set for the link object. For example: ((name "elem-1") (comment "generated")) If there is no extra property then its value should be #f.

undo – its value is #t when the adding of the link should be registered with the undo system and therefore it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created link object.

3.14 basic-link-del

NAME

basic-link-del

SYNOPSIS

```
(basic-link-del id undo)
```

DESCRIPTION

This function deletes a link object from the model. Generally when a link object is deleted then its place is kept in the *LINKS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is a link identification number
undo – its value is #t when the deletion of the link should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.15 basic-triang-valid?

NAME

basic-triang-valid?

SYNOPSIS

```
(basic-triang-valid? id)
```

DESCRIPTION

This function validates a triangle object. A triangle object is valid if its ID number is smaller than *TRIANG_UNIQUE* and if the triangle object exists in the database.

ARGUMENTS

id – the identification number of the triangle object

RETURN VALUE

It returns #t if the triangle object is valid otherwise it returns #f.

3.16 basic-triang-add

NAME

basic-triang-add

SYNOPSIS

```
(basic-triang-add id nodes properties undo)
```

DESCRIPTION

This function creates a triangle object and adds it to the current model.

ARGUMENTS

id – a triangle identification number, however it can be #f when the function will use the current value of the *TRIANG_UNIQUE* variable as the identification number. In this later case the function also increases the *TRIANG_UNIQUE* variable.

nodes – the nodes of the triangle object. It is a list of three integer numbers, which are also valid, point identification numbers.

properties – list of properties, other than nodes, to set for the triangle object. For example: ((name "elem-1") (comment "generated")) If there is no extra property then its value should be #f.

undo – its value is #t when the adding of the triangle should be registered with the undo system and therefore the it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created triangle object.

3.17 basic-triang-del

NAME

basic-triang-del

SYNOPSIS

```
(basic-triang-del id undo)
```

DESCRIPTION

This function deletes a triangle object from the model. Generally when a triangle object is deleted then its place is kept in the *TRIANGS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is a triangle identification number

undo – its value is #t when the deletion of the triangle should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.18 basic-quad-valid?

NAME

basic-quad-valid?

SYNOPSIS

```
(basic-quad-valid? id)
```

DESCRIPTION

This function validates a quad object. A quad object is valid if its ID number is smaller than *QUAD_UNIQUE* and if the quad object exists in the database.

ARGUMENTS

id – the identification number of the quad object

RETURN VALUE

It returns #t if the quad is valid otherwise it returns #f.

3.19 basic-quad-add

NAME

basic-quad-add

SYNOPSIS

```
(basic-quad-add id nodes properties undo)
```

DESCRIPTION

This function creates a quad object and adds it to the current model.

ARGUMENTS

id – a quad identification number, however it can be #f when the function will use the current value of the *QUAD_UNIQUE* variable as the identification number. In this later case the function also increases the *QUAD_UNIQUE* variable.

nodes – the nodes of the quad object. It is a list of four integer numbers, which are also valid, point identification numbers. The points can be listed in either a clockwise or a counterclockwise order.

properties – list of properties, other than nodes, to set for the quad object. For example: ((name "elem-1") (comment "generated")) If there is no extra property then its value should be #f.

undo – its value is #t when the adding of the quad should be registered with the undo system and therefore the it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created quad object.

3.20 basic-quad-del

NAME

basic-quad-del

SYNOPSIS

```
(basic-quad-del id undo)
```

DESCRIPTION

This function deletes a quad object from the model. Generally when a quad object is deleted then its place is kept in the *QUADS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is a quad identification number
undo – its value is #t when the deletion of the quad should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.21 basic-tetrah-valid?

NAME

basic-tetrah-valid?

SYNOPSIS

```
(basic-tetrah-valid? id)
```

DESCRIPTION

This function validates a tetrahedra object. A tetrahedra object is valid if its ID number is smaller than *TETRAH_UNIQUE* and if the tetrahedra exists in the database.

ARGUMENTS

id – the identification number of the tetrahedra

RETURN VALUE

It returns #t if the tetrahedra is valid otherwise it returns #f.

3.22 basic-tetrah-add

NAME

basic-tetrah-add

SYNOPSIS

```
(basic-tetrah-add id nodes properties undo)
```

DESCRIPTION

This function creates a tetrahedra object and adds it to the current model.

ARGUMENTS

id – a tetrahedra identification number, however it can be #f when the function will use the current value of the *TETRAH_UNIQUE* variable as the identification number. In this later case the function also increases the *TETRAH_UNIQUE* variable.

nodes – the nodes of the tetrahedra object. It is a list of four integer numbers, which are also valid, point identification numbers. The points can be defined in any order.

properties – list of properties, other than nodes, to set for the tetrahedra. For example: ((name "elem-1") (comment "generated")) If there is no extra property then its value should be #f.

undo – its value is #t when the adding of the tetrahedra should be registered with the undo system and therefore the it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created tetrahedra object.

3.23 basic-tetrah-del

NAME

basic-tetrah-del

SYNOPSIS

```
(basic-tetrah-del id undo)
```

DESCRIPTION

This function deletes a tetrahedra object from the model. Generally when a tetrahedra object is deleted then its place is kept in the *TETRAHS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is a tetrahedra identification number
undo – its value is #t when the deletion of the tetrahedra should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.24 basic-block-valid?

NAME

basic-block-valid?

SYNOPSIS

```
(basic-block-valid? id)
```

DESCRIPTION

This function validates a block object. A block object is valid if its ID number is smaller than *BLOCK_UNIQUE* and if the block object exists in the database.

ARGUMENTS

id – the identification number of the block

RETURN VALUE

It returns #t if the block is valid otherwise it returns #f.

3.25 basic-block-add

NAME

basic-block-add

SYNOPSIS

```
(basic-block-add id nodes properties undo)
```

DESCRIPTION

This function creates a block object and adds it to the current model.

ARGUMENTS

id – a block identification number, however it can be #f when the function will use the current value of the *BLOCK_UNIQUE* variable as the identification number. In this later case the function also increases the *BLOCK_UNIQUE* variable.

nodes – the nodes of the block object. It is a list of eight integer numbers, which are also valid, point identification numbers. The first four numbers define the top side of the block and the last four points define the bottom side of the block. The first and the fifth, the second and the sixth, the third and the seventh, the fourth and eighth points are connected by a side

properties – list of properties, other than nodes, to set for the block object. For example: ((name "group-1") (comment "imported from file")) If there is no extra property then its value should be #f.

undo – its value is #t when the adding of the block should be registered with the undo system and therefore the it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created block object.

3.26 basic-block-del

NAME

basic-block-del

SYNOPSIS

```
(basic-block-del id undo)
```

DESCRIPTION

This function deletes a block object from the model. Generally when a block object is deleted then its place is kept in the *BLOCKS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is a block identification number

undo – its value is #t when the deletion of the block should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.27 basic-info-valid?

NAME

basic-info-valid?

SYNOPSIS

```
(basic-info-valid? id)
```

DESCRIPTION

This function validates an info object. An info object is valid if its ID number is smaller than *INFO_UNIQUE* and if the info object exists in the database.

ARGUMENTS

id – the identification number of the info object

RETURN VALUE

It returns #t if the info object is valid otherwise it returns #f.

3.28 basic-info-add

NAME

basic-info-add

SYNOPSIS

```
(basic-info-add id properties undo)
```

DESCRIPTION

This function creates an info object and adds it to the current model. An info object may not have a graphical representation, but if it has than the object must have the "render-func" property with a symbol value. For example '(render-func boundary). The value of the render-func property and an actual rendering function must also be registered with the rendering module by using the "info-render-add" function.

ARGUMENTS

id – an info identification number, however it can be #f when the function will use *INFO_UNIQUE* as the identification number. In this later case the function, of course, also increases *INFO_UNIQUE*.

properties – properties to set for the info object. If there is no extra property then its value should be #f.

undo – its value is #t when the adding of the info object should be registered with the undo system and therefore the it can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

It returns the identification number of the created info object.

3.29 basic-info-del

NAME

basic-info-del

SYNOPSIS

```
(basic-info-del id undo)
```

DESCRIPTION

This function deletes an info object from the model. Generally when an info object is deleted then its place is kept in the *INFOS* database, so it can be easily undone at a later stage if necessary.

ARGUMENTS

id – it is an info object identification number
undo – its value is #t when the deletion of the info object should be registered with the undo system and therefore the deletion can be reversed or undone at a later stage. Otherwise its value is #f.

RETURN VALUE

None.

3.30 command-reset

NAME

command-reset

SYNOPSIS

```
(command-reset)
```

DESCRIPTION

This function is the "reset" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.31 command-center

NAME

command-center

SYNOPSIS

```
(command-center)
```

DESCRIPTION

This function is the "center" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.32 command-style

NAME

command-style

SYNOPSIS

```
(command-style)
```

DESCRIPTION

This function is the "style" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.33 command-color

NAME

command-color

SYNOPSIS

```
(command-color)
```

DESCRIPTION

This function is the "color" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.34 command-zoom

NAME

command-zoom

SYNOPSIS

```
(command-zoom)
```

DESCRIPTION

This function is the "zoom" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.35 command-light

NAME

command-light

SYNOPSIS

```
(command-light)
```

DESCRIPTION

This function is the "ligh" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.36 command-new

NAME

command-new

SYNOPSIS

```
(command-new)
```

DESCRIPTION

This function is the "new" command which creates a new model. This function starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.37 command-open

NAME

command-open

SYNOPSIS

```
(command-open)
```

DESCRIPTION

This function is the "open" command which can open a new model in the native format. This function starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.38 command-save

NAME

command-save

SYNOPSIS

```
(command-save)
```

DESCRIPTION

This function is the "save" command which can save the current model in the native format. This function starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.39 command-quit

NAME

command-quit

SYNOPSIS

```
(command-quit)
```

DESCRIPTION

This function is the "quit" command.

ARGUMENTS

None.

RETURN VALUE

None.

3.40 command-u

NAME

command-u

SYNOPSIS

```
(command-u)
```

DESCRIPTION

This function is the "u" command and it undoes the last command or in other words it reverses the effect of the last command.

ARGUMENTS

None.

RETURN VALUE

None.

3.41 command-redo

NAME

command-redo

SYNOPSIS

```
(command-redo)
```

DESCRIPTION

This function is the "redo" command and it redoes the previously undone command.

ARGUMENTS

None.

RETURN VALUE

None.

3.42 command-undo

NAME

command-undo

SYNOPSIS

```
(command-undo)
```

DESCRIPTION

This function is the "undo" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.43 command-menuload

NAME

command-menuload

SYNOPSIS

```
(command-menuload)
```

DESCRIPTION

This function is the "menuload" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.44 command-select

NAME

command-select

SYNOPSIS

```
(command-select)
```

DESCRIPTION

This function is the "select" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.45 command-background

NAME

command-background

SYNOPSIS

```
(command-background)
```

DESCRIPTION

This function is the "background" command.

ARGUMENTS

None.

RETURN VALUE

None.

3.46 command-getcoord

NAME

command-getcoord

SYNOPSIS

```
(command-getcoord)
```

DESCRIPTION

This function is the "getcoord" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.47 command-getdist

NAME

command-getdist

SYNOPSIS

```
(command-getdist)
```

DESCRIPTION

This function is the "getdist" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.48 command-getcolor

NAME

command-getcolor

SYNOPSIS

```
(command-getcolor)
```

DESCRIPTION

This function is the "getcolor" command and it starts the command by setting up the selection mechanism.

ARGUMENTS

None.

RETURN VALUE

None.

3.49 command-list

NAME

command-list

SYNOPSIS

```
(command-list)
```

DESCRIPTION

This function is the "list" command and it lists all available commands in the program. The command prints not only the name of the function, but a description if it exists.

ARGUMENTS

None.

RETURN VALUE

None.

3.50 text-char-cond

NAME

text-char-cond

SYNOPSIS

```
(text-char-cond start line func)
```

DESCRIPTION

This function is a general character finder. It can be used to find or skip certain characters in a string.

ARGUMENTS

start – the starting position from which to start the scanning
line – the string
func – the function which returns true when the scanning for certain characters can stop

RETURN VALUE

It returns the position of the first character that satisfies the given function or #f if the end of string is reached while scanning.

3.51 text-skip-whitespace

NAME

text-skip-whitespace

SYNOPSIS

```
(text-skip-whitespace start line)
```

DESCRIPTION

This function skips the white spaces, like space or the tab characters, until it reaches a non-white space character or the end of string.

ARGUMENTS

start – the starting position from which to start the scanning for white spaces
line – the string

RETURN VALUE

It returns the position of the first non-white space character or #f if the end of string is reached while scanning.

3.52 text-find-char

NAME

text-find-char

SYNOPSIS

```
(text-find-char start line ch)
```

DESCRIPTION

This function tries to find a character in the string starting from the specified character position.

ARGUMENTS

start – the starting character position in the "line" argument
line – the string
ch – the character to find

RETURN VALUE

It returns the position of the character or #f if the end of string is reached while scanning for the character.

3.53 text-split-with-empty

NAME

text-split-with-empty

SYNOPSIS

```
(text-split-with-empty str ch empty)
```

DESCRIPTION

This function splits a string into substrings. The separation positions are specified by a given character. The function can store or ignore created empty strings. The idea of this function is taken from the "string-split" function of guile 1.5.

ARGUMENTS

str – the string to split
ch – character or character list where the text splitting should occur
empty – when #f it ignores empty, created strings, otherwise the empty strings are also stored

RETURN VALUE

It returns the list of split string. Each element in the list is a string.

3.54 entry-add-command

NAME

entry-add-command

SYNOPSIS

```
(entry-add-command command)
```

DESCRIPTION

This function adds a new command to the list of available commands. There is no check whether the command already exists, but the function checks the validity of the command. The argument must be a list where the first element is the name of the command and it is a string, while the second element in the list is a procedure with zero arity (no arguments). (The arity of the procedure is not checked.)

ARGUMENTS

command – a list containing the name of the command, the function and an optional description of the command

RETURN VALUE

None.

3.55 entry-list-command

NAME

entry-list-command

SYNOPSIS

```
(entry-list-command)
```

DESCRIPTION

This function lists all, currently available commands. It prints the name and the description of the command if available.

ARGUMENTS

None.

RETURN VALUE

None.

3.56 entry-add-abbrev

NAME

entry-add-abbrev

SYNOPSIS

```
(entry-add-abbrev abbrev)
```

DESCRIPTION

This function adds a new abbreviation to the list of available, abbreviated commands. There is no check whether the abbreviation already exist. If the abbreviation already exist then only the last definition is used!

ARGUMENTS

abbrev – the abbreviation to add. It must be a two element list, where the first string is the abbreviation itself and the second string is the actual command to invoke (full name of the command).

RETURN VALUE

None.

3.57 entry-command-cache-set!

NAME

entry-command-cache-set!

SYNOPSIS

```
(entry-command-cache-set list)
```

DESCRIPTION

This function sets the command cache to a list of commands which will be executed by the entry callback function. The list can be empty, when no extra execution is done.

ARGUMENTS

list – a list commands, where every element of the list is a string

RETURN VALUE

None.

3.58 entry-callback-get

NAME

entry-callback-get

SYNOPSIS

```
(entry-callback-get)
```

DESCRIPTION

This function returns the current active callback function of the entry.

ARGUMENTS

None.

RETURN VALUE

It returns the current callback function of the entry.

3.59 entry-callback-execute

NAME

entry-callback-execute

SYNOPSIS

```
(entry-callback-execute)
```

DESCRIPTION

This is the callback function for the entry widget. This function is an extra layer between the graphical widget and the user defined callback functions. This layer is required due to the fact, that the user defined callback can change and not all GUI widget may allow to change the callback function.

The function executes/calls the user defined callback function. Moreover if the *COMMAND_CACHE* variable is not an empty list then for each element in the list it also calls the user defined callback function.

ARGUMENTS

None.

RETURN VALUE

None.

3.60 entry-callback-replace

NAME

entry-callback-replace

SYNOPSIS

```
(entry-callback-replace callback-function)
```

DESCRIPTION

This function replaces the current callback function with a new one.

ARGUMENTS

callback-function – the new callback function. There is no check that the new function is a valid function!

RETURN VALUE

None.

3.61 entry-callback-default

NAME

entry-callback-default

SYNOPSIS

```
(entry-callback-default)
```

DESCRIPTION

This function is the standard or default callback function associated with the entry widget. The function is also the standard command processor. As a command processor it understands full and abbreviated commands. The abbreviated command can be user defined or simple abbreviation. In the case of user abbreviated command the abbreviation can be anything, for example: "tt" for "line". In the case of simple abbreviation it must be the first non-ambiguous part of a command, for example: "co" for "copy" is correct but "c" can be "copy" or "center". In the case of ambiguous abbreviation the system notifies the user. Moreover just by pressing ENTER or SPACE at an empty entry the last command is called again. It is important to note, that not the whole command is repeated, only the command is called again.

ARGUMENTS

None.

RETURN VALUE

None.

3.62 entry-execute-menu

NAME

entry-execute-menu

SYNOPSIS

```
(entry-execute-menu command)
```

DESCRIPTION

This function executes one or several commands. The function does not change the callback function, therefore the command is processed by the current callback function!

ARGUMENTS

command – a list of commands

RETURN VALUE

None. NOTE

3.63 entry-get-text

NAME

entry-get-text

SYNOPSIS

```
(entry-get-text)
```

DESCRIPTION

This function reads back the user's command and post-process it if necessary. Post-processing means for example to remove trailing newline or space character. The function also clears the entry widget! This function is not really necessary, if SPACE and ENTER key do not reach the graphical entry widget, thus in that case they would not appear in the string. However, since there can be some problems with the graphical toolkit on some systems, this function is kept and used.

ARGUMENTS

None.

RETURN VALUE

It returns the text from the entry widget.

3.64 sx-error-internal

NAME

sx-error-internal

SYNOPSIS

```
(sx-error-internal who message)
```

DESCRIPTION

This function print an error message notifying the user that there is an internal problem in the program.

ARGUMENTS

who – it is symbol that denotes the name of the function which generated the error
message – extra message to print

RETURN VALUE

It returns #f.

EXAMPLE

```
*INTERNAL ERROR: Invalid object in delete-object*
```

3.65 sx-error

NAME

sx-error

SYNOPSIS

```
(sx-error message)
```

DESCRIPTION

This function print an error message.

ARGUMENTS

message – error message to print

RETURN VALUE

It returns #f.

EXAMPLE

```
*ERROR: Cannot delete object*
```

3.66 sx-warning

NAME

sx-warning

SYNOPSIS

```
(sx-warning message)
```

DESCRIPTION

This function print a warning message.

ARGUMENTS

message – warning message to print

RETURN VALUE

It returns #f.

EXAMPLE

```
*WARNING: Cannot delete object*
```

3.67 sx-normal3d

NAME

sx-normal3d

SYNOPSIS

```
(sx-normal3d x y z)
```

DESCRIPTION

The function creates a binding for the glNormal3d OpenGL function.

ARGUMENTS

x – X component of the normal
y – Y component of the normal
z – Z component of the normal

RETURN VALUE

None.

3.68 sx-normal3dv

NAME

sx-normal3dv

SYNOPSIS

```
(sx-normal3dv normal)
```

DESCRIPTION

The function creates a binding for the glNormal3d OpenGL function.

ARGUMENTS

normal – a three element list

RETURN VALUE

None.

3.69 sx-vertex2d

NAME

sx-vertex2d

SYNOPSIS

```
(sx-vertex2d x y)
```

DESCRIPTION

The function creates a binding for the glVertex2d OpenGL function.

ARGUMENTS

x – X component of the vertex
y – Y component of the vertex

RETURN VALUE

None.

3.70 sx-vertex3d

NAME

sx-vertex3d

SYNOPSIS

```
(sx-vertex3d x y z)
```

DESCRIPTION

The function creates a binding for the glVertex3d OpenGL function.

ARGUMENTS

x – X component of the vertex
y – Y component of the vertex
z – Z component of the vertex

RETURN VALUE

None.

3.71 sx-vertex3dv

NAME

sx-vertex3dv

SYNOPSIS

```
(sx-vertex3dv vertex)
```

DESCRIPTION

The function creates a binding for the glVertex3d OpenGL function.

ARGUMENTS

vertex – a three element list

RETURN VALUE

None.

3.72 sx-color4d

NAME

sx-color4d

SYNOPSIS

```
(sx-color4d r g b a)
```

DESCRIPTION

The function creates a binding for the glColor4d OpenGL function.

ARGUMENTS

r – red component of the color
g – green component of the color
b – blue component of the color
a – alpha component of the color

RETURN VALUE

None.

3.73 sx-color4dv

NAME

sx-color4dv

SYNOPSIS

```
(sx-color4dv color)
```

DESCRIPTION

The function creates a binding for the glColor4d OpenGL function.

ARGUMENTS

color – a four element list containing the color components

RETURN VALUE

None.

3.74 sx-color3d

NAME

sx-color3d

SYNOPSIS

```
(sx-color3d r g b)
```

DESCRIPTION

The function creates a binding for the glColor3d OpenGL function.

ARGUMENTS

r – red component of the color
g – green component of the color
b – blue component of the color

RETURN VALUE

None.

3.75 sx-color3dv

NAME

sx-color3dv

SYNOPSIS

```
(sx-color3dv color)
```

DESCRIPTION

The function creates a binding for the glColor3d OpenGL function.

ARGUMENTS

color – a three element list

RETURN VALUE

None.

3.76 sx-begin

NAME

sx-begin

SYNOPSIS

```
(sx-begin obj)
```

DESCRIPTION

The function creates a binding for the glBegin OpenGL function.

ARGUMENTS

obj – the type of the object, possible values are: sx-points, sx-lines, sx-triangles, sx-quads, sx-line-loop

RETURN VALUE

None.

3.77 sx-end

NAME

sx-end

SYNOPSIS

```
(sx-end)
```

DESCRIPTION

The function creates a binding for the glEnd OpenGL function.

ARGUMENTS

None.

RETURN VALUE

None.

3.78 sx-init-names

NAME

sx-init-names

SYNOPSIS

```
(sx-init-names)
```

DESCRIPTION

The function creates a binding for the glInitNames OpenGL function.

ARGUMENTS

None.

RETURN VALUE

None.

3.79 sx-load-name

NAME

sx-load-name

SYNOPSIS

```
(sx-load-name n)
```

DESCRIPTION

The function creates a binding for the glLoadName OpenGL function.

ARGUMENTS

n – an integer

RETURN VALUE

None.

3.80 sx-push-name

NAME

sx-push-name

SYNOPSIS

```
(sx-push-name n)
```

DESCRIPTION

The function creates a binding for the glPushName OpenGL function.

ARGUMENTS

n – an integer

RETURN VALUE

None.

3.81 sx-pop-name

NAME

sx-pop-name

SYNOPSIS

```
(sx-pop-name)
```

DESCRIPTION

The function creates a binding for the glPopName OpenGL function.

ARGUMENTS

None.

RETURN VALUE

None.

3.82 sx-enable

NAME

sx-enable

SYNOPSIS

```
(sx-enable const)
```

DESCRIPTION

The function creates a binding for the glEnable OpenGL function.

ARGUMENTS

const – an OpenGL constant, possible values are: sx-lighting, sx-blend, sx-depth-test, sx-light0

RETURN VALUE

None.

3.83 sx-disable

NAME

sx-disable

SYNOPSIS

```
(sx-disable const)
```

DESCRIPTION

The function creates a binding for the glDisable OpenGL function.

ARGUMENTS

const – an OpenGL constant, possible values are: sx-lighting, sx-blend, sx-depth-test, sx-light0

RETURN VALUE

None.

3.84 sx-clear

NAME

sx-clear

SYNOPSIS

```
(sx-clear mask)
```

DESCRIPTION

The function creates a binding for the glClear OpenGL function.

ARGUMENTS

mask – an OpenGL constant

RETURN VALUE

None.

3.85 sx-materialfv

NAME

sx-materialfv

SYNOPSIS

```
(sx-materialfv face pname param)
```

DESCRIPTION

The function creates a binding for the glMaterialfv OpenGL function.

ARGUMENTS

face – an OpenGL contant to denote the face: sx-front-and-back, sx-front, sx-back

pname – an openGL constant to denote the material property: sx-diffuse, sx-ambient, sx-specular, sx-emission, sx-shininess, sx-ambient-and-diffuse

param – the material property as a list, the size of the list depends on the material property

RETURN VALUE

None.

3.86 sx-light-modelf

NAME

sx-light-modelf

SYNOPSIS

```
(sx-light-modelf pname param)
```

DESCRIPTION

The function creates a binding for the glLightModelf OpenGL function.

ARGUMENTS

pname – an OpenGL constant: sx-light-model-two-side, sx-light-model-local-viewer
param – it is 0.0 or 1.0

RETURN VALUE

None.

3.87 sx-lightfv

NAME

sx-lightfv

SYNOPSIS

```
(sx-lightfv light pname param)
```

DESCRIPTION

The function creates a binding for the glLightfv OpenGL function.

ARGUMENTS

light – an OpenGL contant to denote the light: sx-light0
pname – an openGL constant to denote the light property: sx-diffuse, sx-ambient, sx-specular, sx-emission, sx-shininess, sx-ambient-and-diffuse, sx-position
param – the light property as a list, the size of the list depends on the light property

RETURN VALUE

None.

3.88 sx-depth-mask

NAME

sx-depth-mask

SYNOPSIS

```
(sx-depth-mask flag)
```

DESCRIPTION

The function creates a binding for the glDepthMask OpenGL function.

ARGUMENTS

flag – #t or #f

RETURN VALUE

None.

3.89 sx-shade-model

NAME

sx-shade-model

SYNOPSIS

```
(sx-shade-model mode)
```

DESCRIPTION

The function creates a binding for the glShadeModel OpenGL function.

ARGUMENTS

mode – an OpenGL constant, e.g. sx-smooth

RETURN VALUE

None.

3.90 sx-hint

NAME

sx-hint

SYNOPSIS

```
(sx-hint target mode)
```

DESCRIPTION

The function creates a binding for the glHint OpenGL function.

ARGUMENTS

target – an OpenGL constant, e.g. sx-polygon-smooth
mode – an OpenGL constant, e.g. sx-fastest

RETURN VALUE

None.

3.91 sx-polygon-mode

NAME

sx-polygon-mode

SYNOPSIS

```
(sx-polygon-mode face mode)
```

DESCRIPTION

The function creates a binding for the glPolygonMode OpenGL function.

ARGUMENTS

face – an OpenGL constant, sx-front-and-back, sx-front, sx-back
mode – an OpenGL constant, e.g. sx-fill

RETURN VALUE

None.

3.92 sx-point-size

NAME

sx-point-size

SYNOPSIS

```
(sx-point-size size)
```

DESCRIPTION

The function creates a binding for the glPointSize OpenGL function.

ARGUMENTS

size – a number

RETURN VALUE

None.

3.93 sx-clear-color

NAME

sx-clear-color

SYNOPSIS

```
(sx-clear-color r g b a)
```

DESCRIPTION

The function creates a binding for the glClearColor OpenGL function.

ARGUMENTS

r – red component of the color
g – green component of the color
b – blue component of the color
a – alpha component of the color

RETURN VALUE

None.

3.94 sx-new-list

NAME

sx-new-list

SYNOPSIS

```
(sx-new-list lst mode)
```

DESCRIPTION

The function creates a binding for the glNewList OpenGL function.

ARGUMENTS

lst – an integer number
mode – an OpenGL constant, sx-compile, sx-compile-and-execute

RETURN VALUE

None.

3.95 sx-end-list

NAME

sx-end-list

SYNOPSIS

```
(sx-end-list)
```

DESCRIPTION

The function creates a binding for the glEndList OpenGL function.

ARGUMENTS

None.

RETURN VALUE

None.

3.96 sx-call-list

NAME

sx-call-list

SYNOPSIS

```
(sx-call-list lst)
```

DESCRIPTION

The function creates a binding for the glCallList OpenGL function.

ARGUMENTS

lst – an integer number

RETURN VALUE

None.

3.97 sx-rotatef

NAME

sx-rotatef

SYNOPSIS

```
(sx-rotatef angle x y z)
```

DESCRIPTION

The function creates a binding for the glRotated OpenGL function.

ARGUMENTS

angle – a number
x – X component of the axis, 1 or 0
y – Y component of the axis, 1 or 0
z – Z component of the axis, 1 or 0

RETURN VALUE

None.

3.98 sx-translatef

NAME

sx-translatef

SYNOPSIS

```
(sx-translatef x y z)
```

DESCRIPTION

The function creates a binding for the glTranslatef OpenGL function.

ARGUMENTS

x – X component of the translation
y – Y component of the translation
z – Z component of the translation

RETURN VALUE

None.

3.99 sx-mult-matrixf

NAME

sx-mult-matrixf

SYNOPSIS

```
(sx-mult-matrixf matrix)
```

DESCRIPTION

The function creates a binding for the glMultMatrixd OpenGL function.

ARGUMENTS

matrix – a 4 by 4 matrix given as a 16 element list

RETURN VALUE

None.

3.100 sx-select-buffer-init

NAME

sx-select-buffer-init

SYNOPSIS

```
(sx-select-buffer-init size buf)
```

DESCRIPTION

The function creates a binding for the glSelectBuffer OpenGL function.

ARGUMENTS

size – an integer number, denoting the length of the buffer
buf – a buffer created by the "sx-select-buffer-create" function

RETURN VALUE

None.

NOTES

For further documentation see the "render" module.

3.101 sx-select-buffer-get

NAME

sx-select-buffer-get

SYNOPSIS

```
(sx-select-buffer-get npick size buf)
```

DESCRIPTION

The function returns the buffer as a vector.

ARGUMENTS

npick – an integer number
size – an integer number
buf – a buffer created by the "sx-select-buffer-create" function

RETURN VALUE

None.

NOTES

For further documentation see the "render" module.

3.102 sx-select-buffer-create

NAME

sx-select-buffer-create

SYNOPSIS

```
(sx-select-buffer-create nn)
```

DESCRIPTION

The function creates a OpenGL selection buffer.

ARGUMENTS

nn – an integer number

RETURN VALUE

None.

NOTES

For further documentation see the "render" module.

3.103 sx-render-mode

NAME

sx-render-mode

SYNOPSIS

```
(sx-render-mode mode)
```

DESCRIPTION

The function creates a binding for the glRenderMode OpenGL function.

ARGUMENTS

mode – an OpenGL constant, sx-select, sx-render

RETURN VALUE

None.

3.104 sx-matrix-mode

NAME

sx-matrix-mode

SYNOPSIS

```
(sx-matrix-mode mode)
```

DESCRIPTION

The function creates a binding for the glMatrixMode OpenGL function.

ARGUMENTS

mode – an OpenGL constant, sx-projection, sx-modelview

RETURN VALUE

None.

3.105 sx-load-identity

NAME

sx-load-identity

SYNOPSIS

```
(sx-load-identity)
```

DESCRIPTION

The function creates a binding for the glLoadIdentity OpenGL function.

ARGUMENTS

None.

RETURN VALUE

None.

3.106 sx-ortho

NAME

sx-ortho

SYNOPSIS

```
(sx-ortho left right bottom top near far)
```

DESCRIPTION

The function creates a binding for the glOrtho OpenGL function.

ARGUMENTS

left – a number
right – a number
bottom – a number
top – a number
near – a number
far – a number

RETURN VALUE

None.

3.107 sx-blend-func

NAME

sx-blend-func

SYNOPSIS

```
(sx-blend-func source destination)
```

DESCRIPTION

The function creates a binding for the glBlendFunc OpenGL function.

ARGUMENTS

source – an OpenGL constant, e.g. sx-src-alpha
destination – an OpenGL constant, e.g. sx-dst-alpha

RETURN VALUE

None.

3.108 sx-push-attrib

NAME

sx-push-attrib

SYNOPSIS

```
(sx-push-attrib attrib)
```

DESCRIPTION

The function creates a binding for the glPushAttrib OpenGL function.

ARGUMENTS

attrib – an OpenGL constant, e.g. sx-viewport-size

RETURN VALUE

None.

3.109 sx-pop-attrib

NAME

sx-pop-attrib

SYNOPSIS

```
(sx-pop-attrib)
```

DESCRIPTION

The function creates a binding for the glPopAttrib OpenGL function.

ARGUMENTS

None.

RETURN VALUE

None.

3.110 sx-viewport

NAME

sx-viewport

SYNOPSIS

```
(sx-viewport x y w h)
```

DESCRIPTION

The function creates a binding for the glViewport OpenGL function.

ARGUMENTS

x – a number
y – a number
w – a number
h – a number

RETURN VALUE

None.

3.111 sx-line-width

NAME

sx-line-width

SYNOPSIS

```
(sx-line-width w)
```

DESCRIPTION

The function creates a binding for the glLineWidth OpenGL function.

ARGUMENTS

w – a number

RETURN VALUE

None.

3.112 sx-read-error

NAME

sx-read-error

SYNOPSIS

```
(sx-read-error text)
```

DESCRIPTION

This is the error reporting function when reading a model file in native format.

ARGUMENTS

text – the error message

RETURN VALUE

It returns #f.

3.113 sx-read-data-line?

NAME

sx-read-data-line?

SYNOPSIS

```
(sx-read-data-line? line)
```

DESCRIPTION

This function checks that there is some data in the line.

ARGUMENTS

line – a string, not including the new line, linefeed ot return character.

RETURN VALUE

It returns #f for comment and empty lines, otherwise #t for line with data.

3.114 sx-read-line-string

NAME

sx-read-line-string

SYNOPSIS

```
(sx-read-line-string port)
```

DESCRIPTION

This function reads in a line which contains some data. The function ignores comments and empty lines. The function also keeps track of the number of read lines for error reporting.

ARGUMENTS

port – a port open for reading

RETURN VALUE

It returns the line as a string or an end of file object.

3.115 sx-read-line-tokenize

NAME

sx-read-line-tokenize

SYNOPSIS

```
(sx-read-line-tokenize line)
```

DESCRIPTION

This function tokenizes a string line. It basically transforms the line to a list where each element in the list is a string which is enclosed by white spaces in the line. For example: "aa bb cc" -> ("aa" "bb" "bb").

ARGUMENTS

line – the string line

RETURN VALUE

If the argument is a string then it returns the tokenised line otherwise it returns the argument unchanged.

3.116 sx-read-line

NAME

sx-read-line

SYNOPSIS

```
(sx-read-line port)
```

DESCRIPTION

This function reads in a line which contains some data. The function ignores comment and empty lines. The function also keeps track of the number of read lines for error reporting.

ARGUMENTS

port – a port open for reading

RETURN VALUE

It returns the tokenised line or an end of file object.

3.117 sx-read-token-integer

NAME

sx-read-token-integer

SYNOPSIS

```
(sx-read-token-integer tokens)
```

DESCRIPTION

It parses in an integer number after a keyword with error checking.

ARGUMENTS

tokens – a list of string tokens

RETURN VALUE

It returns a list of the keyword and the parsed number or #f in the case of an error.

3.118 sx-read-token-double

NAME

sx-read-token-double

SYNOPSIS

```
(sx-read-token-double tokens)
```

DESCRIPTION

It parses in a real number after a keyword with error checking.

ARGUMENTS

tokens – a list of string tokens

RETURN VALUE

It returns a list of the keyword and the parsed number or #f in the case of an error.

3.119 sx-read-header-token

NAME

sx-read-header-token

SYNOPSIS

```
(sx-read-header-token tokens)
```

DESCRIPTION

This function parses in a header component depending on the first keyword in the line.

ARGUMENTS

tokens – a list of string tokens

RETURN VALUE

It returns the read header component or #f in the case of an error.

3.120 sx-read-header-check

NAME

sx-read-header-check

SYNOPSIS

```
(sx-read-header-check header)
```

DESCRIPTION

This function checks that all mandatory components of the header has been read in.

ARGUMENTS

header – the association list of the header

RETURN VALUE

It returns the association list of the header or #f in the case of an error.

3.121 sx-read-header

NAME

sx-read-header

SYNOPSIS

```
(sx-read-header port)
```

DESCRIPTION

This function reads in the header of the native file format for type "sx-1.0-format". The "sx-1.0-format" text identifies the type of the file format.

ARGUMENTS

port – a port open for reading

RETURN VALUE

It returns the header list or #f in the case of an error.

3.122 sx-read-object-init

NAME

sx-read-object-init

SYNOPSIS

```
(sx-read-object-init header)
```

DESCRIPTION

This function initializes the temporary database where the read in elements are stored.

ARGUMENTS

header – an association list with all the header information, e.g. the maximum number of objects

RETURN VALUE

None

3.123 sx-read-object-get

NAME

sx-read-object-get

SYNOPSIS

```
(sx-read-object-get type id)
```

DESCRIPTION

This function returns the read in element from the temporary database according to the requested type.

ARGUMENTS

type – the type of the object, a symbol
id – the id number of the object

RETURN VALUE

It returns the object or #f if the element does not exist.

3.124 sx-read-object-set!

NAME

sx-read-object-set!

SYNOPSIS

```
(sx-read-object-set! type id obj)
```

DESCRIPTION

This function sets a new object in the temporary database according to the type and id number

ARGUMENTS

type – the type of the object, a string
id – the id number of the object
obj – the object database

RETURN VALUE

None

3.125 sx-read-object-desc

NAME

sx-read-object-desc

SYNOPSIS

```
(sx-read-object-desc header obj id)
```

DESCRIPTION

This function reads in one object descriptor or object reference. An object reference must have a valid object type and a number which does not exceed the maximum number that is specified in the header of the file.

ARGUMENTS

header – an association list with all the header information, e.g. the maximum number of objects
obj – a string representing the type of the object
id – a string representing the number of the object

RETURN VALUE

It returns an object descriptor, a two element list.

3.126 sx-read-object-desc-list

NAME

sx-read-object-desc-list

SYNOPSIS

```
(sx-read-object-desc-list header line)
```

DESCRIPTION

This function reads in a series of object references. An object reference is like an object header, it has a type and a number. For example: ((p 20) (l 51) (t 14)). By default the 'child' and 'parent' properties contain these object references.

ARGUMENTS

header – an association list with all the header information, e.g. the maximum number of objects
line – a list of string tokens

RETURN VALUE

It returns the list of object descriptions or #f in the case of an error.

3.127 sx-read-object-color

NAME

sx-read-object-color

SYNOPSIS

```
(sx-read-object-color line)
```

DESCRIPTION

This function reads in the color specification for an object. The color specification can be an integer or four real numbers. The integer number denotes the indexed color. The four real numbers represent the red, green, blue and alpha components of the color.

ARGUMENTS

line – a list of string tokens

RETURN VALUE

It returns the color specification or #f in the case of an error.

3.128 sx-read-object-style

NAME

sx-read-object-style

SYNOPSIS

```
(sx-read-object-style line)
```

DESCRIPTION

This function reads in the style specification for an object. The style specification is an integer number.

ARGUMENTS

line – a list of string tokens

RETURN VALUE

It returns the style specification or #f in the case of an error.

3.129 sx-read-object-coords

NAME

sx-read-object-coords

SYNOPSIS

```
(sx-read-object-coords line)
```

DESCRIPTION

This function reads in three coordinates.

ARGUMENTS

line – a list of string tokens

RETURN VALUE

It returns the list of coordinates of #f in the case of an error.

3.130 sx-read-object-nodes

NAME

sx-read-object-nodes

SYNOPSIS

```
(sx-read-object-nodes line n)
```

DESCRIPTION

This function parses the required number of nodes from a line. All nodes must be integer and their order is important.

ARGUMENTS

line – a list of string tokens
n – the required number of nodes

RETURN VALUE

It returns the node list or #f in the case of an error.

3.131 sx-read-info-string

NAME

sx-read-info-string

SYNOPSIS

```
(sx-read-info-string line ns)
```

DESCRIPTION

This function tries to find the double-quote character, denoting the end of the string. The function considers the escaped double-quotes and handles them properly.

ARGUMENTS

line – a string line
ns – starting position to scan for an end of string character

RETURN VALUE

It returns the position of the double-quote character or #f in the case of an error.

3.132 sx-read-info-field

NAME

sx-read-info-field

SYNOPSIS

```
(sx-read-info-field line)
```

DESCRIPTION

This function reads in the fields, properties for an info object. A property may contain numbers, strings, symbols.

ARGUMENTS

line – a string line

RETURN VALUE

It returns the property list or #f in the case of an error.

3.133 sx-read-object-check

NAME

sx-read-object-check

SYNOPSIS

```
(sx-read-object-check type id obj)
```

DESCRIPTION

This function checks that all the mandatory property fields have been read in for an object.

ARGUMENTS

type – a string to identify the type of the object
id – a string token storing the number of the object
obj – an object database containing the properties

RETURN VALUE

It returns #t if all mandatory fields are present in the object and it returns #f in the case of an error.

3.134 sx-read-object-default-prop?

NAME

sx-read-object-default-prop?

SYNOPSIS

```
(sx-read-object-default-prop? type prop)
```

DESCRIPTION

This function determines whether a property is a valid, default property.

ARGUMENTS

type – a string to identify the type of the object
prop – a string name of the property

RETURN VALUE

It returns #t if the property is a default property otherwise it returns #f

3.135 sx-read-object-default-property

NAME

sx-read-object-default-property

SYNOPSIS

```
(sx-read-object-default-property header type line)
```

DESCRIPTION

This function reads in a default property. The function has error checking.

ARGUMENTS

header – an association list with all the header information, e.g. the maximum number of objects
type – a string to identify the type of the object
line – a list of string tokens

RETURN VALUE

It returns when a valid, default property is read from the line and it returns #f in the case of an error.

3.136 sx-read-object

NAME

sx-read-object

SYNOPSIS

```
(sx-read-object header type id port)
```

DESCRIPTION

This function reads in all the properties of an object.

ARGUMENTS

header – an association list with all the header information, e.g. the maximum number of objects
type – a string to identify the type of the object
id – a string token storing the number of the object
port – a port open for reading

RETURN VALUE

It returns a full object representation or #f in the case of a reading error.

3.137 sx-read-object-id

NAME

sx-read-object-id

SYNOPSIS

```
(sx-read-object-id line header)
```

DESCRIPTION

This function checks that the header of an object definition is valid. It checks the type of the object, the number after the type and the valid range of the number. The function also checks that the object is not defined twice.

ARGUMENTS

line – a list of string tokens
header – an association list with all the header information, e.g. the maximum number of objects

RETURN VALUE

It returns #t or #f in the case of an error.

3.138 sx-read-object-list

NAME

sx-read-object-list

SYNOPSIS

```
(sx-read-object-list header port)
```

DESCRIPTION

This function reads in all the objects for the 3d model.

ARGUMENTS

header – an association list with all the header information, e.g. the maximum number of objects
port – an open port to read in

RETURN VALUE

It returns the list of objects with all data or #f in the case of a reading error.

3.139 sx-read-db

NAME

sx-read-db

SYNOPSIS

```
(sx-read-db filename)
```

DESCRIPTION

This function reads in the object database of a 3D model in the native file format. The reading process has full error checking and error reporting capability. After a successfull reading the function also checks the consistency of the database.

ARGUMENTS

filename – the name of the file to read

RETURN VALUE

It returns the objects and the header or #f if there was an error while reading from the port or checking the database.

3.140 sx-write-header

NAME

sx-write-header

SYNOPSIS

```
(sx-write-header port)
```

DESCRIPTION

This function writes out the header of the file in the format of 'sx-1.0-format'.

ARGUMENTS

port – a port opened for writing

RETURN VALUE

None

3.141 sx-write-desc-list

NAME

sx-write-desc-list

SYNOPSIS

```
(sx-write-desc-list lst port)
```

DESCRIPTION

This function writes out a list of object descriptors.

ARGUMENTS

lst – a list of object descriptors
port – a port open for writing

RETURN VALUE

none

3.142 sx-write-numbers

NAME

sx-write-numbers

SYNOPSIS

```
(sx-write-numbers lst port)
```

DESCRIPTION

This function writes out a series of numbers into the port.

ARGUMENTS

lst – a list of numbers
port – a port open for writing

RETURN VALUE

None

3.143 sx-write-info-field

NAME

sx-write-info-field

SYNOPSIS

```
(sx-write-info-field properties port)
```

DESCRIPTION

This function writes out the fields of an info field. The function consideres numbers, strings and symbols. Strings are enclosed between double quotes.

ARGUMENTS

lst – a list of info fields or properties
port – a port open for writing

RETURN VALUE

None

3.144 sx-write-object

NAME

sx-write-object

SYNOPSIS

```
(sx-write-object type port)
```

DESCRIPTION

This function writes out the same type of objects according to the native file format specification

ARGUMENTS

type – a string denoting the type of object, for example "p", "l", ...
port – a port open for writing

RETURN VALUE

None

3.145 sx-write-db

NAME

sx-write-db

SYNOPSIS

```
(sx-write-db filename)
```

DESCRIPTION

This function writes out the full database of a model according to the native file format specification.

ARGUMENTS

port – a port open for writing

RETURN VALUE

None

3.146 sx-check-error

NAME

sx-check-error

SYNOPSIS

```
(sx-check-error text)
```

DESCRIPTION

This is the error handler function for the checking process.

ARGUMENTS

text – the error message

RETURN VALUE

It always returns #f.

3.147 sx-check-references

NAME

sx-check-references

SYNOPSIS

```
(sx-check-references type id chk-list ref-prop)
```

DESCRIPTION

This function checks that all references are correct, the referenced object exists and the referenced object references backward.

ARGUMENTS

type – a symbol denoting the type of the object
id – the identification number of the object
chk-list – the list of object references to check
ref-prop – the name of the property which references backward; when 'chk-list' contains elements from the 'child' property list then 'ref-prop' is 'parent' and vica versa.

RETURN VALUE

It returns the number of errors or zero if there was no error.

3.148 sx-check-nodes

NAME

sx-check-nodes

SYNOPSIS

```
(sx-check-nodes type id nodes child)
```

DESCRIPTION

This function checks that objects with nodes (e.g. lines, triangles, etc) are properly reference the points in nodes and child properties.

ARGUMENTS

type – a symbol denoting the type of the object
id – the identification number of the object
props – all the properties of the object

RETURN VALUE

It returns the number of errors or zero.

3.149 sx-check-db

NAME

sx-check-db

SYNOPSIS

```
(sx-check-db type db)
```

DESCRIPTION

This function performs consistency checks of a type of object. The function mainly checks that all the references are correct.

ARGUMENTS

type – the type of the objects
db – the object database (vector)

RETURN VALUE

It returns #t if the database is correct and #f otherwise

3.150 command-version

NAME

command-version

SYNOPSIS

```
(command-version)
```

DESCRIPTION

This function is the "version" command and it prints the version number of the SX modeller.

ARGUMENTS

None.

RETURN VALUE

None.

3.151 sx-main

NAME

sx-main

SYNOPSIS

```
(sx-main menu)
```

DESCRIPTION

This is the main function of the SX modeller and it must be called to carry out the initialisation.

ARGUMENTS

menu – the name of the menu file

RETURN VALUE

None.

3.152 menu-callback-func

NAME

menu-callback-func

SYNOPSIS

```
(menu-callback-func func)
```

DESCRIPTION

This is the callback function which is called when a menu item is selected.

ARGUMENTS

func – a list of strings to execute through the entry widget

RETURN VALUE

None.

3.153 menu-error

NAME

menu-error

SYNOPSIS

```
(menu-error text)
```

DESCRIPTION

This is a simple error handler while reading in a menu file. The function prints the error message.

ARGUMENTS

text – the error message to print

RETURN VALUE

It always returns #f

3.154 menu-read-data-line?

NAME

menu-read-data-line?

SYNOPSIS

```
(menu-read-data-line? line)
```

DESCRIPTION

This is an auxiliary function for the "menu-read-line" function and it determines whether the read line contains any data. It ignores empty or comment lines.

ARGUMENTS

line – the read line as a string

RETURN VALUE

It returns #t if the line contains data otherwise it returns #f.

3.155 menu-read-line

NAME

menu-read-line

SYNOPSIS

```
(menu-read-line port)
```

DESCRIPTION

This function reads in a data line. It ignores empty and comment lines.

ARGUMENTS

port – a port open for reading

RETURN VALUE

It returns the data line or the end of file object if the end of the file has been reached.

3.156 menu-parse-command-line

NAME

menu-parse-command-line

SYNOPSIS

```
(menu-parse-command-line line)
```

DESCRIPTION

This function tries to read in a line corresponding to a menu item. The line must contain a menu name enclosed by square brackets. After the name the line may contain a series of commands to be executed when the menu item is selected.

ARGUMENTS

line – the read line as a string

RETURN VALUE

It returns a two element list where the first element is the name of the menu item and the second element is a list of strings. When no command is specified in the line then the second element is a list containing an empty string. This is equivalent to the action when a simple ENTER is pressed.

3.157 menu-read-section

NAME

menu-read-section

SYNOPSIS

```
(menu-read-section menu fp)
```

DESCRIPTION

This function reads in a menu section or a series of menu items until the *ENDMENU is encountered in the file. If the menu name contains two minus signs then a separator is inserted into the menu.

ARGUMENTS

menu – the parent widget, to which the read menu items should be attached
fp – a port open for reading

RETURN VALUE

The function returns #t if it could successfully read all the menu items otherwise it returns #f.

3.158 menu-read-header

NAME

menu-read-header

SYNOPSIS

```
(menu-read-header fp)
```

DESCRIPTION

It reads in a menu header line.

ARGUMENTS

fp – a port open for reading

RETURN VALUE

It returns the end of file object if the end of the file is reached, otherwise it returns the tokenised line, a list of string tokens.

3.159 menu-read-file

NAME

menu-read-file

SYNOPSIS

```
(menu-read-file menubar fp)
```

DESCRIPTION

This function performs the actual reading in of the menu file.

ARGUMENTS

menubar – the menubar widget to which the menus should be attached
fp – a port open for reading

RETURN VALUE

The function returns the list of created menus or it returns #f in the case of an error.

3.160 menu-load

NAME

menu-load

SYNOPSIS

```
(menu-load file)
```

DESCRIPTION

This function reads in a new menu. The old menu is always deleted. If no menu could be read in then no menu will be displayed. On the other hand when a menu could be read in successfully then it is attached to the menubar and made available to the user.

ARGUMENTS

file – the name of the menu file, including the directory path

RETURN VALUE

none

3.161 menu-init

NAME

menu-init

SYNOPSIS

```
(menu-init parent)
```

DESCRIPTION

This function creates a graphical menubar and attaches to the main window. The created menu is stored in the *MENU_BAR* variable.

ARGUMENTS

parent – the widget which will contain the menubar, for example the main window

RETURN VALUE

None.

3.162 list->command-selection

NAME

list->command-selection

SYNOPSIS

```
(list->command-selection sellist)
```

DESCRIPTION

This function converts an element selection list, where all elements are specified by object descriptors, into a textual command which can be passed to the command processor. For example:

```
((p 1) (p 2) (q 4)) -\aaLarger{} "point,1,point,2,quad,4"
```

ARGUMENTS

sellist – the list of object descriptors

RETURN VALUE

It returns a string

3.163 list->object-selection

NAME

list->object-selection

SYNOPSIS

```
(list->object-selection objlist)
```

DESCRIPTION

This function converts an element selection list, where all elements are specified by object descriptors, into a list of strings which can be processed by the object selection mechanism. For example:

```
((p 1) (p 2) (q 4)) -\aaLarger{} ("point" "1" "2" "quad" "4")
```

ARGUMENTS

sellist – the list of object descriptors

RETURN VALUE

It returns a string

3.164 mouse-dragging-panning

NAME

mouse-dragging-panning

SYNOPSIS

```
(mouse-dragging-panning x y w h state)
```

DESCRIPTION

This function handles the dragging case when the left mouse button is pressed. The effect of the function is model panning.

ARGUMENTS

x – the current x position of the mouse cursor
y – the current y position of the mouse cursor
w – the current width of the graphical context, window
h – the current height of the graphical context, window
state – the state of the keyboar modifiers, like shift, ctrl, alt.

RETURN VALUE

None

3.165 mouse-dragging-rotation

NAME

mouse-dragging-rotation

SYNOPSIS

```
(mouse-dragging-rotation x y w h shift-state)
```

DESCRIPTION

This function handles the dragging case when the middle mouse button is pressed. The effect of the function is model rotation.

ARGUMENTS

x – the current x position of the mouse cursor
y – the current y position of the mouse cursor
w – the current width of the graphical context, window
h – the current height of the graphical context, window
shift-state – the state of the keyboar modifiers, like shift, ctrl, alt.

RETURN VALUE

None

3.166 mouse-dragging-zoom

NAME

mouse-dragging-zoom

SYNOPSIS

```
(mouse-dragging-zoom x y w h state)
```

DESCRIPTION

This function handles the dragging case when the right mouse button is pressed. The effect of the function is model zooming.

ARGUMENTS

x – the current x position of the mouse cursor
y – the current y position of the mouse cursor
w – the current width of the graphical context, window
h – the current height of the graphical context, window
shift-state – the state of the keyboar modifiers, like shift, ctrl, alt.

RETURN VALUE

None

3.167 mouse-button-press-callback

NAME

mouse-button-press-callback

SYNOPSIS

```
(mouse-button-press-callback pressed_button x y)
```

DESCRIPTION

This function is the general callback function which is called when a mouse button is pressed. The function behaves differently depending on the value of the *PICK_MODE* variable. The function can be in a simple viewing mode when panning, rotation and zooming is allowed or in a selection mode. There are two kind of selections, coordinate and object selections. The two selections implemented differently, since in the case of the coordinate selection the user wants coordinates of a point, while in the case object selection a series of objects are required by the user. Moreover in object selection it is possible to use a single cursor selection or a window selection.

ARGUMENTS

pressed_button – a symbol denoting which mouse button has been pressed. The valid values are symbols of: left, middle, right.
x – the current x position of the mouse cursor
y – the current y position of the mouse cursor

RETURN VALUE

None

3.168 mouse-button-motion-callback

NAME

mouse-button-motion-callback

SYNOPSIS

```
(mouse-button-motion-callback x y state)
```

DESCRIPTION

This function is the general callback function which is executed when the mouse is moved. The function executes the specific dragging function according to the value of the "*MOUSE-DRAGGING-HANDLER*" variable.

ARGUMENTS

x – the current x position of the mouse cursor
y – the current y position of the mouse cursor
state – the state of the keyboar modifiers, like shift, ctrl, alt.

RETURN VALUE

None

3.169 mouse-button-release-callback

NAME

mouse-button-release-callback

SYNOPSIS

```
(mouse-button-release-callback released_button x y)
```

DESCRIPTION

This function is the general callback function which is executed when a mouse button is released. The function is mainly used to reset the dragging handling.

ARGUMENTS

released_button – the name of the released button as a symbol. The valid valid values are: left, middle, right
x – the current x position of the mouse cursor
y – the current y position of the mouse cursor

RETURN VALUE

None

3.170 operation-new

NAME

operation-new

SYNOPSIS

```
(operation-new)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-new" function. If the user accepts that a new model is to be created then this function clears the object database and resets the viewing parameters.

ARGUMENTS

None.

RETURN VALUE

None.

3.171 operation-open-native-format

NAME

operation-open-native-format

SYNOPSIS

```
(operation-open-native-format)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-open" function. The function loads model which is saved in the native format.

ARGUMENTS

None.

RETURN VALUE

None.

3.172 operation-save-native-format

NAME

operation-save-native-format

SYNOPSIS

```
(operation-save-native-format)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-save" function. The function saves the current model in a native mode.

ARGUMENTS

None.

RETURN VALUE

None.

3.173 operation-quit

NAME

operation-quit

SYNOPSIS

```
(operation-quit)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-quit" function. If there was no modifications made to the model since the last time it has been saved then this function simply quits. Otherwise it sets up a new selection mechanism to ask the user whether the program should really quit.

ARGUMENTS

None.

RETURN VALUE

None.

3.174 operation-quit-internal

NAME

operation-quit-internal

SYNOPSIS

```
(operation-quit-internal)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-quit" function. Depending on the user's answer, the program quits or continues the execution.

ARGUMENTS

None.

RETURN VALUE

None.

3.175 operation-u

NAME

operation-u

SYNOPSIS

```
(operation-u)
```

DESCRIPTION

This function undoes one operation.

ARGUMENTS

None.

RETURN VALUE

None.

3.176 operation-redo

NAME

operation-redo

SYNOPSIS

```
(operation-redo)
```

DESCRIPTION

This function redoes an undone operation.

ARGUMENTS

None.

RETURN VALUE

None.

3.177 operation-undo

NAME

operation-undo

SYNOPSIS

```
(operation-undo)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-undo" function. The function can place a user mark, undo one operation, undo several operations or undoe everything.

ARGUMENTS

None.

RETURN VALUE

None.

3.178 operation-menuload

NAME

operation-menuload

SYNOPSIS

```
(operation-menuload)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-menuload" function. If a filename is selected the function calls the menuload function defined in the "menu.scm" file.

ARGUMENTS

None.

RETURN VALUE

None.

3.179 operation-select

NAME

operation-select

SYNOPSIS

```
(operation-select)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-select" function. The function does not do any specific thing, it basically only stores the current selection as previous selection.

ARGUMENTS

None.

RETURN VALUE

None.

3.180 operation-background

NAME

operation-background

SYNOPSIS

```
(operation-background)
```

DESCRIPTION

This function is the callback function for the "command-background" function. The function sets the color of the background of the rendering screen.

ARGUMENTS

None.

RETURN VALUE

None.

3.181 operation-getcoord

NAME

operation-getcoord

SYNOPSIS

```
(operation-getcoord)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-getcoord" function. If a point has been selected it prints coordinate of the selected point. However it is possible that the coordinate of the point has been combined from other points using the .x, .xz, etc filters.

ARGUMENTS

None.

RETURN VALUE

None.

3.182 operation-getdist

NAME

operation-getdist

SYNOPSIS

```
(operation-getdist)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-getdist" function. If two points have been selected it determines the distance between the points and prints.

ARGUMENTS

None.

RETURN VALUE

None.

3.183 operation-getcolor

NAME

operation-getcolor

SYNOPSIS

```
(operation-getcolor)
```

DESCRIPTION

This function is the callback function for the selection mechanism setup by the "command-getcolor" function. If an element has been selected it prints either the indexed color or the RGBS color of the selected element.

ARGUMENTS

None.

RETURN VALUE

None.

3.184 option-menu-init

NAME

option-menu-init

SYNOPSIS

```
(option-menu-init panel)
```

DESCRIPTION

This function initializes the optional menus. The optional menus appear on the right hand side of the screen and they consists of buttons. Usually the optional menus inform the user about the available keywords.

ARGUMENTS

panel – the new optional menu will be inserted into the panel widget

RETURN VALUE

None

3.185 option-menu-new

NAME

option-menu-new

SYNOPSIS

```
(option-menu-new options)
```

DESCRIPTION

This function creates the submenus for an optional menu. These optional menus are the children of the *OPTION_MENU* (created by the "gui-option-menu-init" function), however they are not added and they are not visible initially.

ARGUMENTS

options – a list of options to appear in the option menu. The list contains pairs, where the first element is the name of the option (a string) and the second element is #t or #f indicating whether the option is executed immediately or it is only shown in the entry widget. For example: (("last" #t) ("previous" #t) ...)

RETURN VALUE

None

3.186 option-menu-callback

NAME

option-menu-callback

SYNOPSIS

```
(option-menu-callback name exec)
```

DESCRIPTION

This is the callback function for the option menu items. The function basically writes the name of the item into the command entry and depending on the second argument of the function it executes it immediately or simply returns.

ARGUMENTS

name – the name of the item, which is also the command
exec – flag to indicate whether the command should be executed immediately

RETURN VALUE

None

3.187 option-menu-get

NAME

option-menu-get

SYNOPSIS

```
(option-menu-get)
```

DESCRIPTION

This function returns the current optional menu. The returned value is the value of one of the *OPTION_<name>* menus.

ARGUMENTS

None

RETURN VALUE

It returns the current optional menu.

3.188 option-menu-set

NAME

option-menu-set

SYNOPSIS

```
(option-menu-set menu)
```

DESCRIPTION

This function sets a new optional menu which will replace the previously shown menu.

ARGUMENTS

menu – the optional menu, one of the *OPTION_<name>* menus

RETURN VALUE

None

3.189 delete

NAME

delete

SYNOPSIS

```
(delete elem oldlist)
```

DESCRIPTION

This is an auxiliary function. Some scheme implementations did not have it defined, but it is required in this program.

The function deletes all occurences of an element from a list. The comparison function is equal?, so every element is fully evaluated and lists inside list can be also deleted.

ARGUMENTS

elem – the element to delete, can be any expression, atom, list, etc.
oldlist – the list from which the element should be deleted.

RETURN VALUE

It returns a new list which does not contain the specified element.

3.190 object-db-new

NAME

object-db-new

SYNOPSIS

```
(object-db-new)
```

DESCRIPTION

This function creates a new object database.

ARGUMENTS

None.

RETURN VALUE

It returns the newly created database.

3.191 object-db-length

NAME

object-db-length

SYNOPSIS

```
(object-db-length db)
```

DESCRIPTION

This function returns the number of objects in the database.

ARGUMENTS

db – an object database

RETURN VALUE

It returns the number of objects.

3.192 object-db-get

NAME

object-db-get

SYNOPSIS

```
(object-db-get db key)
```

DESCRIPTION

The function returns the object identified by a key.

ARGUMENTS

db – an object database
key – the key value to identify the object

RETURN VALUE

It returns the object or #f if no object can be found with key.

3.193 object-db-set!

NAME

object-db-set!

SYNOPSIS

```
(object-db-set! db key val)
```

DESCRIPTION

This function creates a new object and stores it in the database which can be identified by a key.

ARGUMENTS

db – an object database
key – the key value to identify the object
val – the object to store

RETURN VALUE

None.

3.194 object-db-del!

NAME

object-db-del!

SYNOPSIS

```
(object-db-del! db key)
```

DESCRIPTION

This function deletes an object from the database. The object is identified by a key.

ARGUMENTS

db – an object database
key – the key value to identify the object

RETURN VALUE

None.

3.195 object-db-map

NAME

object-db-map

SYNOPSIS

```
(object-db-map proc db)
```

DESCRIPTION

The function goes through all objects of the database in some order. The objects are passed to the "proc" function for processing and the return values are collected into a list.

ARGUMENTS

proc – a function with two arguments, the first argument is the object identifier and the second argument is the object itself
db – an object database

RETURN VALUE

It returns the list of the return values of the "proc" function.

3.196 object-db-for-each

NAME

object-db-for-each

SYNOPSIS

```
(object-db-for-each proc db)
```

DESCRIPTION

The function goes through all objects of the database in some order. The objects are passed to the "proc" function for processing, however the return values are ignored.

ARGUMENTS

proc – a function with two arguments, the first argument is the object identifier and the second argument is the object itself
db – an object database

RETURN VALUE

None.

3.197 get-point-max

NAME

get-point-max

SYNOPSIS

```
(get-point-max)
```

DESCRIPTION

Generally the function simply returns the maximum extent of the current 3D model. However if it is not set, therefore no point is defined then it returns 1.0. If only one point is defined, but that one point is at the origo, thus the extent would be zero, it also returns 1.0.

ARGUMENTS

None

RETURN VALUE

It returns the maximum extent of the current 3D model.

3.198 set-point-max!

NAME

set-point-max!

SYNOPSIS

```
(set-point-max! val)
```

DESCRIPTION

This function sets a new maximum extent if the argument, which may represent any of the coordinate directions, is larger than the current maximum extent.

ARGUMENTS

None

RETURN VALUE

It returns the maximum extent of the current 3D model.

3.199 object-type-string

NAME

object-type-string

SYNOPSIS

```
(object-type-string type)
```

DESCRIPTION

This function simply returns the name of the object type as a string, for example when the "type" argument is "p" then the returned string is "Point".

ARGUMENTS

type – the type of the object as a symbol

RETURN VALUE

It returns a string.

3.200 object-all-clear!

NAME

object-all-clear!

SYNOPSIS

```
(object-all-clear!)
```

DESCRIPTION

This function clears the object databases for all object types. The function also resets the "*POINT_MAX*" variable.

ARGUMENTS

None.

RETURN VALUE

None.

3.201 object-all-set!

NAME

object-all-set!

SYNOPSIS

```
(object-all-set! point-max  point-db
                 link-max   link-db
                 triang-max triang-db
                 quad-max   quad-db
                 tetrah-max tetrah-db
                 block-max  block-db
                 info-max   info-db)
```

DESCRIPTION

This function resets the object databases for all types of objects. It resets the maximum ID of the object types and it also stores new databases for each object type. All, old data is lost. This function is also used by the native file reader.

ARGUMENTS

point-max – the new maximum id for points
link-max – the new maximum id for links
triang-max – the new maximum id for triangles
quad-max – the new maximum id for quads
tetrah-max – the new maximum id for tetrahedrons
block-max – the new maximum id for blocks
info-max – the new maximum id for infos
object-db – the object database

RETURN VALUE

None

3.202 object-unique-get

NAME

object-unique-get

SYNOPSIS

```
(object-unique-get type)
```

DESCRIPTION

This function returns the maximum identification number of a given object type. It returns one of the *<object>_UNIQUE* variables.

ARGUMENTS

type – the type of object as a symbol

RETURN VALUE

It returns a number or if the type is not recognised it returns #f.

3.203 object-unique-set!

NAME

object-unique-set!

SYNOPSIS

```
(object-unique-set! type id)
```

DESCRIPTION

This function sets a new maximum identification number for a given object type. If the identification number is smaller than the current maximum ID then the function does not do anything.

ARGUMENTS

type – the type of the object as a symbol
id – the new, maximum identification number

RETURN VALUE

None.

3.204 object-length

NAME

object-length

SYNOPSIS

```
(object-length type)
```

DESCRIPTION

The function returns the number of objects of a given type.

ARGUMENTS

type – the type of the object as a symbol

RETURN VALUE

It returns the number of objects.

3.205 object-get

NAME

object-get

SYNOPSIS

```
(object-get type id)
```

DESCRIPTION

The function returns the object of a given type from the object database. The function may return #f when the object does not exist.

ARGUMENTS

type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

It returns the object or #f.

3.206 object-set!

NAME

object-set!

SYNOPSIS

```
(object-set! type id elem)
```

DESCRIPTION

The function stores a new object of a given type in the database. If an object was already stored in the database then after the execution of this function that object will be lost.

ARGUMENTS

type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

None.

3.207 object-del!

NAME

object-del!

SYNOPSIS

```
(object-del! type id)
```

DESCRIPTION

The function deletes an object of a given type from the database.

ARGUMENTS

type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

None.

3.208 object-map

NAME

object-map

SYNOPSIS

```
(object-map proc type)
```

DESCRIPTION

The function passes one-by-one all objects of a given type to a user specified function. The return values of the "proc" function are collected into a list.

ARGUMENTS

proc – a function with two arguments. The first argument is an object identification and the second argument is the object.
type – the type of the object as a symbol

RETURN VALUE

It returns a list.

3.209 object-for-each

NAME

object-for-each

SYNOPSIS

```
(object-for-each proc type)
```

DESCRIPTION

The function passes one-by-one all objects of a given type to a user specified function. The return values of the "proc" function are discarded.

ARGUMENTS

proc – a function with two arguments. The first argument is an object identification and the second argument is the object.
type – the type of the object as a symbol

RETURN VALUE

None.

3.210 object-new

NAME

object-new

SYNOPSIS

```
(object-new)
```

DESCRIPTION

The function creates a new object, a set of properties.

ARGUMENTS

None.

RETURN VALUE

It returns the new object.

3.211 property-set!

NAME

property-set!

SYNOPSIS

```
(property-set! elem prop_id prop allow-internal)
```

DESCRIPTION

The function sets a property in an object to a new value. If there was already a property in the object then the old value of the property will be lost after the execution of this function.

ARGUMENTS

elem – the object
prop_id – the identifier of the property
prop – the values of the property
allow-internal – #t or #f denoting whether the function is allowed to set internal property

RETURN VALUE

None.

3.212 property-del!

NAME

property-del!

SYNOPSIS

```
(property-del! elem prop_id allow-internal)
```

DESCRIPTION

The function deletes a property from an object.

ARGUMENTS

elem – the object
prop_id – the identifier of the property
allow-internal – #t or #f denoting whether the function is allowed to delete an internal property

RETURN VALUE

None.

3.213 property-get

NAME

property-get

SYNOPSIS

```
(property-get elem prop_id)
```

DESCRIPTION

The function returns a property from an object.

ARGUMENTS

elem – the object
prop_id – the identifier of the property

RETURN VALUE

It returns the value of the property.

3.214 property-add-comp!

NAME

property-add-comp!

SYNOPSIS

```
(property-add-comp! elem prop_id comp allow-internal)
```

DESCRIPTION

The function adds a new component to a property of an object.

ARGUMENTS

elem – the object
prop_id – the identifier of the property
comp – the property component
allow-internal – #t or #f denoting whether the function is allowed to add a component to a property

RETURN VALUE

#t

3.215 property-del-comp!

NAME

property-del-comp!

SYNOPSIS

```
(property-del-comp! elem prop_id comp allow-internal)
```

DESCRIPTION

The function deletes a component from a property of an object. The rest of the property is unchanged in the object.

ARGUMENTS

elem – the object
prop_id – the identifier of the property
comp – the property component
allow-internal – #t or #f denoting whether the function is allowed to delete an internal property component

RETURN VALUE

#t

3.216 property-get-list

NAME

property-get-list

SYNOPSIS

```
(property-get-list elem)
```

DESCRIPTION

The function collects all the properties from an object into a list.

ARGUMENTS

elem – the object

RETURN VALUE

It returns the list of properties.

3.217 property-get-user

NAME

property-get-user

SYNOPSIS

```
(property-get-user elem)
```

DESCRIPTION

The function collects all non-internal properties from an object into a list.

ARGUMENTS

elem – the object

RETURN VALUE

It returns the list of properties.

3.218 property-map

NAME

property-map

SYNOPSIS

```
(property-map proc elem)
```

DESCRIPTION

The function passes one-by-one all properties of an object to the "proc" function and it collects the return values of the "proc" function into a list.

ARGUMENTS

proc – a function with two arguments, where the first argument is the name of the property and the second argument is the property itself
elem – the object

RETURN VALUE

It returns a list.

3.219 property-for-each

NAME

property-for-each

SYNOPSIS

```
(property-for-each proc elem)
```

DESCRIPTION

The function passes one-by-one all properties of an object to the "proc" function. The return values of the "proc" function are discarded

ARGUMENTS

proc – a function with two arguments, where the first argument is the name of the property and the second argument is the property itself
elem – the object

RETURN VALUE

None.

3.220 property-type-set!

NAME

property-type-set!

SYNOPSIS

```
(property-type-set! type id prop_id prop allow-internal)
```

DESCRIPTION

This function sets the property in an object to a new value. The object is specified by type and identification number. The function can modify internal properties or restrict the modification of them. The function is based on the "property-set!" function.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object
prop_id – the name of the property as a symbol
prop – the new value for the property
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns #t when it could carry out the operation and #f when there was an error.

3.221 property-type-del!

NAME

property-type-del!

SYNOPSIS

```
(property-type-del! type id prop_id allow-internal)
```

DESCRIPTION

This function deletes the a property from an object. The object is specified by type and identification number. The function can modify internal properties or restrict the modification of them. Even when all the properties are deleted (which is actually not a valid object) the object itself is not deleted! The function is based on the "property-del!" function.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object
prop_id – the name of the property, as a symbol
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns #t when it could carry out the operation and #f when there was an error.

3.222 property-type-get

NAME

property-type-get

SYNOPSIS

```
(property-type-get type id prop_id)
```

DESCRIPTION

The function returns a property of an object. The object is specified by type and identification number. If the property does not exist then it returns #f. The function is based on the "property-get" function.

ARGUMENTS

type – the type of the object as a symbol
id – the identification number of the object
prop_id – the name of the property, as a symbol

RETURN VALUE

It returns the property of #f.

3.223 property-type-add-comp!

NAME

property-type-add-comp!

SYNOPSIS

```
(property-type-add-comp! type id prop_id comp allow-internal)
```

DESCRIPTION

This function adds a property component to a property in an object. The object is specified by type and identification number. If the property does not exist the function creates and adds the property. If the property exists then the function adds a new component to the property. The function can modify internal properties or restrict the modification of them. The function is based on the "property-add-comp!" function.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object
prop_id – the name of the property, as a symbol
comp – the component of the property to be added. It can be any expression, number, string, symbol, list, etc. However it is important to note that if the same component occurs several times in the property even then the component is added.
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns #t when it could carry out the operation and #f when there was an error.

3.224 property-type-del-comp!

NAME

property-type-del-comp!

SYNOPSIS

```
(property-type-del-comp! type id prop_id comp allow-internal)
```

DESCRIPTION

This function deletes a component of a property from an object. The object is specified by type and identification number. If all components of a property are deleted the property itself is not deleted from the object! To fully delete a property from an object use the "property-type-del!" function. The function can modify internal properties or restrict the modification of them. The function is based on the "property-del-comp!" function.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object
prop_id – the name of the property, as a symbol
comp – the component of the property to be deleted. It can be any expression, number, string, symbol, list, etc. However it is important to note that if the same component occurs several times in the property then all occurences of the property component will be deleted. For example: before it is (numbers 1 2 3 4 1 6 1 7) and after deleting number "1" as a property component the result is (numbers 2 3 4 6 7).
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns #t when it could carry out the operation and #f when there was an error.

3.225 property-type-get-list

NAME

property-type-get-list

SYNOPSIS

```
(property-type-get-list type id)
```

DESCRIPTION

This function returns all properties of an object. The object is specified by type and identification number. The function is based on the "property-get-list" function.

ARGUMENTS

type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

It returns the list of properties.

3.226 property-type-get-user

NAME

property-type-get-user

SYNOPSIS

```
(property-type-get-user type id)
```

DESCRIPTION

This function returns all non-internal, user defined properties of an object. If there are no user defined properties then it returns an empty list. The object is specified by type and identification number. The function is based on the "property-get-user" function.

ARGUMENTS

type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

It returns the list of user defined properties or an empty list.

3.227 property-type-map

NAME

property-type-map

SYNOPSIS

```
(property-type-map proc type id)
```

DESCRIPTION

The function passes one-by-one all properties of an object to the "proc" function and it collects the return values of the "proc" function into a list. The object is specified by type and identification number. The function is based on the "property-map" function.

ARGUMENTS

proc – a function with two arguments, where the first argument is the property identifier and the second argument is the property
type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

It returns the list of values returned by the "proc" function.

3.228 property-type-for-each

NAME

property-type-for-each

SYNOPSIS

```
(property-type-for-each proc type id)
```

DESCRIPTION

The function passes one-by-one all properties of an object to the "proc" function. The returned values of the "proc" function are discarded. The object is specified by type and identification number. The function is based on the "property-for-each" function.

ARGUMENTS

proc – a function with two arguments, where the first argument is the property identifier and the second argument is the property
type – the type of the object as a symbol
id – the identification number of the object

RETURN VALUE

None.

3.229 property-undo-del!

NAME

property-undo-del!

SYNOPSIS

```
(property-undo-del! type id prop_id undo allow-internal)
```

DESCRIPTION

This function deletes a property using the "property-del!" function however the deletion is also registered with the undo system and therefore it can be undone at a later stage. If the property does not exist in the object no changes occur in the object and no action is registered with the undo system.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object, which is also the position of the object in the database
prop_id – the name of the property, as a symbol
undo – it is #t when the action of the function should be registeres with the undo system otherwise #f
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns the result of the "property-del!" function.

3.230 property-undo-set!

NAME

property-undo-set!

SYNOPSIS

```
(property-undo-set! type id prop_id prop undo allow-internal)
```

DESCRIPTION

This function sets the property in an object to a new value using the "property-set!" function, however the setting is also registered with the undo system and therefore it can be undone at a later stage. The function handle two different cases. In the first case the property already exists in the object, thus it is changed to something else, therefore the undo action should be setting the property back to its original value. In the second case the property does not exists in the object, therefore the undo action should be deletion of the property.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object, which is also the position of the object in the database
prop_id – the name of the property as a symbol
prop – the new value for the property
undo – it is #t when the action of the function should be registeres with the undo system otherwise #f
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns the result of the "object-property-set" function.

3.231 property-undo-del-comp!

NAME

property-undo-del-comp!

SYNOPSIS

```
(property-undo-del-comp! type id prop_id comp undo allow-internal)
```

DESCRIPTION

This function deletes a property component using the "property-del-comp!" function, however the deletion is also registered with the undo system and therefore it can be undone at a later stage. If the property component does not exist in the object no changes occur in the object and no action is registered with the undo system.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object, which is also the position of the object in the database
prop_id – the name of the property, as a symbol
comp – the component of the property to be deleted. For further comments see the "property-del-comp!" function.
undo – it is #t when the action of the function should be registeres with the undo system otherwise #f
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns the result of the "property-del-comp!" function.

3.232 property-undo-add-comp!

NAME

property-undo-add-comp!

SYNOPSIS

```
(property-undo-add-comp! type id prop_id comp undo allow-internal)
```

DESCRIPTION

This function adds the property to an object. The result of the function is undoable. The function relies on the "property-undo-add-comp!" function, therefore handles both cases: when prop_id exists in the object and when prop_id does not exist in the object.

ARGUMENTS

type – the type of the object, as a sybmbol, to select the appropriate object database
id – the identification number of the object, which is also the position of the object in the database
prop_id – the name of the property as a symbol
comp – a new value to add to the property
undo – it is #t when the action of the function should be registeres with the undo system otherwise #f
allow-internal – it is #t when the function can modify internal properties otherwise its value is #f

RETURN VALUE

It returns the result of the "property-add-comp!" function.

3.233 render-background-color-set!

NAME

render-background-color-set!

SYNOPSIS

```
(render-background-color-set! color)
```

DESCRIPTION

The function sets the background color of the OpenGL area.

ARGUMENTS

color – a list of red, green, blue components

RETURN VALUE

None.

3.234 render-background-color-get

NAME

render-background-color-get

SYNOPSIS

```
(render-background-color-get)
```

DESCRIPTION

The function returns the background color of the OpenGL area.

ARGUMENTS

None.

RETURN VALUE

None.

3.235 render-background-color-inv

NAME

render-background-color-inv

SYNOPSIS

```
(render-background-color-inv)
```

DESCRIPTION

The function inverts the background color of the OpenGL area. Since all color components are a number between 0.0 and 1.0, therefore the inversion means that each color component is substracted from 1.0. In this case 0.0 becomes 1.0 and 1.0 becomes 0.0, or black becomes white and vica versa.

ARGUMENTS

None.

RETURN VALUE

None.

3.236 render-pan-x-set!

NAME

render-pan-x-set!

SYNOPSIS

```
(render-pan-x-set! x)
```

DESCRIPTION

This is a mutator function, to set the *GL_PAN_X* variable to a new value.

ARGUMENTS

x – the new value for *GL_PAN_X*

RETURN VALUE

None.

3.237 render-pan-y-set!

NAME

render-pan-y-set!

SYNOPSIS

```
(render-pan-y-set! y)
```

DESCRIPTION

This is a mutator function, to set the *GL_PAN_Y* variable to a new value.

ARGUMENTS

y – the new value for *GL_PAN_Y*

RETURN VALUE

None.

3.238 render-quaternion-set!

NAME

render-quaternion-set!

SYNOPSIS

```
(render-quaternion-set! q)
```

DESCRIPTION

This is a mutator function, to set the *GL_ROTATION_QUATERNION* variable to a new value.

ARGUMENTS

q – the new quaternion for *GL_ROTATION_QUATERNION*

RETURN VALUE

None.

3.239 render-rotation-x-set!

NAME

render-rotation-x-set!

SYNOPSIS

```
(render-rotation-x-set! x)
```

DESCRIPTION

This is a mutator function, to set the *GL_ROTATION_X* variable to a new value.

ARGUMENTS

x – the new rotation around the x axis

RETURN VALUE

None.

3.240 render-rotation-y-set!

NAME

render-rotation-y-set!

SYNOPSIS

```
(render-rotation-y-set! y)
```

DESCRIPTION

This is a mutator function, to set the *GL_ROTATION_Y* variable to a new value.

ARGUMENTS

y – the new rotation around the y axis

RETURN VALUE

None.

3.241 render-rotation-z-set!

NAME

render-rotation-z-set!

SYNOPSIS

```
(render-rotation-z-set! z)
```

DESCRIPTION

This is a mutator function, to set the *GL_ROTATION_Z* variable to a new value.

ARGUMENTS

z – the new rotation around the z axis

RETURN VALUE

None.

3.242 render-zoom-set!

NAME

render-zoom-set!

SYNOPSIS

```
(render-zoom-set! z)
```

DESCRIPTION

This is a mutator function, to set the *GL_ZOOM* variable to a new value.

ARGUMENTS

z – the new zooming factor

RETURN VALUE

None.

3.243 render-center-set!

NAME

render-center-set!

SYNOPSIS

```
(render-center-set! c)
```

DESCRIPTION

This is a mutator function, to set the *GL_CENTER* variable to a new value.

ARGUMENTS

c – the new center of rotation for viewing

RETURN VALUE

None.

3.244 render-lighting-set!

NAME

render-lighting-set!

SYNOPSIS

```
(render-lighting-set! flag)
```

DESCRIPTION

This is a mutator function, to set the *GL_RENDER_LIGHTING* variable to a new value.

ARGUMENTS

flag – the new value for the *GL_RENDER_LIGHTING*, #t or #f

RETURN VALUE

None.

3.245 object-center

NAME

object-center

SYNOPSIS

```
(object-center nodes)
```

DESCRIPTION

This function determines the centre of an object. It takes all nodes of the object, adds the coordinates up and divides the components by the number of nodes.

ARGUMENTS

nodes – a list of point identification numbers

RETURN VALUE

It returns the coordinates of the object centre.

3.246 render-with-style

NAME

render-with-style

SYNOPSIS

```
(render-with-style nodes xcenter tiling lighting)
```

DESCRIPTION

This function generates the vertices of an object considering the rendering style of the object. If tiling is allowed it reduces the size of the element before rendering. The function should be called between the glBegin and the glEnd OpenGL commands. If lighting is enabled the function also generates the normals.

ARGUMENTS

nodes – the list of point identification numbers
xcenter – the coordinates of the centre of the object
tiling – it is #f when the object should be rendered in full size, otherwise it is a number between 0.0 and 1.0 when the element is rendered with reduced size.
lighting – it is #t when there the light is on otherwise it is #f

RETURN VALUE

None.

3.247 render-basic-color

NAME

render-basic-color

SYNOPSIS

```
(render-basic-color req-style type render-id render-func)
```

DESCRIPTION

This function renders all those object of the same type whose rendering style is the same as the requested rendering style. For example it renders all triangles which has the "lines" rendering style. However the function only uses simple color definition, not material. In other words it only uses the glColor function. If an object is selected it is rendered with the *GL_COLOR_SELECTION* color, otherwise the object is rendered with its own color.

The function is used for info objects as well, when the "render-func" argument is #f. In this case the rendering function is different for different info objects.

ARGUMENTS

req-style – the requested style as a symbol. The valid values are the same as in the *GL_STYLE_LIST* variable

type – the type fo the object to render, as a symbol

render-id – the rendering identification number, the value of one of the *OBJECT_<object>_ID* variables

render-func – this is the rendering function to render only one of the objects. The function has two arguments: the object with properties and the tiling parameter. It can be #f for info objects when the object itself contains a property refering to the name of the rendering function. This rendering function must be registered in the render module.

RETURN VALUE

None.

3.248 render-basic-material

NAME

render-basic-material

SYNOPSIS

```
(render-basic-material req-style type render-id render-func)
```

DESCRIPTION

This function renders all those object of the same type whose rendering style is the same as the requested rendering style. For example it renders all triangles which has the "opaq" rendering style. The function uses OpenGL materials to define the color of the objects except when the lighting is switched off and when the requested rendering style is "lines". If an object is selected it is rendered with the *GL_COLOR_SELECTION* color, otherwise the object is rendered with its own color.

ARGUMENTS

req-style – the requested style as a symbol. The valid values are the same as in the *GL_STYLE_LIST* variable
type – the type fo the object to render, as a symbol
render-id – the rendering identification number, the value of one of the *OBJECT_<object>_ID* variables
render-func – this is the rendering function to render only one of the objects. The function has three arguments: the object, the rendering style and the tiling parameter.

RETURN VALUE

None.

3.249 point-one-render-only

NAME

point-one-render-only

SYNOPSIS

```
(point-one-render-only obj tiling)
```

DESCRIPTION

This function renders one point.

ARGUMENTS

obj – the object
tiling – tiling parameter, not used by this function

RETURN VALUE

None.

3.250 points-render-basic

NAME

points-render-basic

SYNOPSIS

```
(points-render-basic req-style)
```

DESCRIPTION

This function renders all the points which has the requested rendering style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.251 link-one-render-only

NAME

link-one-render-only

SYNOPSIS

```
(link-one-render-only obj tiling)
```

DESCRIPTION

This function renders one link object.

ARGUMENTS

obj – the object
tiling – tiling parameter. If it is #f the object is rendered in full size otherwise it is a number between 0.0 and 1.0 denoting how much the size of the object should be reduced.

RETURN VALUE

None.

3.252 links-render-basic

NAME

links-render-basic

SYNOPSIS

```
(links-render-basic req-style)
```

DESCRIPTION

This function renders all the links which has the requested rendering style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.253 triang-one-render-only

NAME

triang-one-render-only

SYNOPSIS

```
(triang-one-render-only obj style tiling)
```

DESCRIPTION

This function renders one triangle object.

ARGUMENTS

elem – the object
style – the rendering style of the object
tiling – tiling parameter. If it is #f the object is rendered in full size otherwise it is a number between 0.0 and 1.0 denoting how much the size of the object should be reduced.

RETURN VALUE

None.

3.254 triangs-render-basic

NAME

triangs-render-basic

SYNOPSIS

```
(triangs-render-basic req-style)
```

DESCRIPTION

This function renders all the triangles which has the requested rendering style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.255 quad-one-render-only

NAME

quad-one-render-only

SYNOPSIS

```
(quad-one-render-only obj style tiling)
```

DESCRIPTION

This function renders one quad object.

ARGUMENTS

elem – the object
style – the rendering style of the object
tiling – tiling parameter. If it is #f the object is rendered in full size otherwise it is a number between 0.0 and 1.0 denoting how much the size of the object should be reduced.

RETURN VALUE

None.

3.256 quads-render-basic

NAME

quads-render-basic

SYNOPSIS

```
(quads-render-basic req-style)
```

DESCRIPTION

This function renders all the quads which has the requested rendering style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.257 tetrah-one-render-only

NAME

tetrah-one-render-only

SYNOPSIS

```
(tetrah-one-render-only obj style tiling)
```

DESCRIPTION

This function renders one tetrahedron object.

ARGUMENTS

elem – the object
style – the rendering style of the object
tiling – tiling parameter. If it is #f the object is rendered in full size otherwise it is a number between 0.0 and 1.0 denoting how much the size of the object should be reduced.

RETURN VALUE

None.

3.258 tetrahs-render-basic

NAME

tetrahs-render-basic

SYNOPSIS

```
(tetrahs-render-basic req-style)
```

DESCRIPTION

This function renders all the tetrahedra which has the requested rendering style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.259 block-one-render-only

NAME

block-one-render-only

SYNOPSIS

```
(block-one-render-only obj style tiling)
```

DESCRIPTION

This function renders one block object. In the case of opaq and transparent rendering style the faces are rendered as triangles and not as quads.

ARGUMENTS

elem – the object
style – the rendering style of the object
tiling – tiling parameter. If it is #f the object is rendered in full size otherwise it is a number between 0.0 and 1.0 denoting how much the size of the object should be reduced.

RETURN VALUE

None.

3.260 blocks-render-basic

NAME

blocks-render-basic

SYNOPSIS

```
(blocks-render-basic req-style)
```

DESCRIPTION

This function renders all the blocks which has the requested rendering style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.261 info-render-add

NAME

info-render-add

SYNOPSIS

```
(info-render-add name func)
```

DESCRIPTION

This function registers an info object rendering function.

ARGUMENTS

name – the property identifier to identify an info object
func – a rendering function

RETURN VALUE

None

3.262 info-render-get

NAME

info-render-get

SYNOPSIS

```
(info-render-get name)
```

DESCRIPTION

This function returns an info object rendering function.

ARGUMENTS

name – the property identifier to identify an info object

RETURN VALUE

None

3.263 infos-render-basic

NAME

infos-render-basic

SYNOPSIS

```
(infos-render-basic req-style)
```

DESCRIPTION

This function renders all the info objects.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None

3.264 render-expose

NAME

render-expose

SYNOPSIS

```
(render-expose compile)
```

DESCRIPTION

This function is called whenever the graphical window requires a refresh and the contents should be redrawn.

ARGUMENTS

compile – flag to indicate whether the model should be compiled again or it should be simply redrawn

RETURN VALUE

None.

3.265 render-viewport

NAME

render-viewport

SYNOPSIS

```
(render-viewport width height)
```

DESCRIPTION

This function is called whenever the graphical window is resized.

ARGUMENTS

width – the new width of graphical window
height – the new height of graphical window

RETURN VALUE

None.

3.266 render-init

NAME

render-init

SYNOPSIS

```
(render-init)
```

DESCRIPTION

This function initializes some of the OpenGL parameters. This function must be called when the graphical window is created for the first time.

ARGUMENTS

None.

RETURN VALUE

None.

3.267 render-all-objects

NAME

render-all-objects

SYNOPSIS

```
(render-all-objects req-style)
```

DESCRIPTION

This function calls the rendering function for all object types with the requested style.

ARGUMENTS

req-style – the requested rendering style, see *GL_STYLE_LIST*

RETURN VALUE

None.

3.268 render-modelview-rotation

NAME

render-modelview-rotation

SYNOPSIS

```
(render-modelview-rotation mode)
```

DESCRIPTION

This function handles the rotation of the model in viewing mode.

ARGUMENTS

mode – the rotation modeis either "xy" or "quaternion"

RETURN VALUE

None.

3.269 object-gl-selection-buffer

NAME

object-gl-selection-buffer

SYNOPSIS

```
(object-gl-selection-buffer)
```

DESCRIPTION

This function increases the OpenGL selection buffer if necessary. If no object has been added to the model since the last call of this function or all objects still fit into the current selection buffer then the function does not change anything

ARGUMENTS

None.

RETURN VALUE

None.

3.270 render-scene

NAME

render-scene

SYNOPSIS

```
(render-scene mode x y pick_x pick_y compile)
```

DESCRIPTION

This function renders the model. The function has multiple purposes. It can simply redraw the model or it can be used for selection. When the function is used for redrawing the model, then it provides two modes. In the first case the model is compiled into an OpenGL buffer and it is redrawn at the same time. In the second mode it uses the compiled model to redraw the model. Using the compiled model greatly speeds up the refreshing of the window. However when the compiled model is used for redrawing no part of the model can be changed, for example no element can change its color to indicate that it is selected. (This is the reason to also use the compilation mode.)

The second purpose of the function is to be able to select objects from the screen. The function uses the OpenGL provided selection mechanism, where the model is properly rendered, but not shown, however the selection area is scanned which element fall into it. The selected elements are returned.

ARGUMENTS

mode – the rendering mode of the model. When it is "select" the model is rendered for selection, when it is "render" then it redraws the model on the screen.
x – the x coordinate of the bottom left corner of the selection area
y – the y coordinate of the bottom left corner of the selection area
pick_x – the width of the selection area
pick_y – the height of the selection area
compile – a flag to specify whether the model should be compiled into an OpenGL buffer or it should use the already existing OpenGL buffer and redraw the model without compilation. Its value can be #t or #f.

RETURN VALUE

When "mode" is "render" it returns #f. When "mode" is "select" the function either returns #f when no element was selected or it returns a list of object descriptors, which are already filtered according to the object selection mechanism, for example only points or the closest link, etc.

3.271 sellist-make

NAME

sellist-make

SYNOPSIS

```
(sellist-make)
```

DESCRIPTION

The function creates a new set for object selection.

ARGUMENTS

None.

RETURN VALUE

None.

3.272 sellist-length

NAME

sellist-length

SYNOPSIS

```
(sellist-length table)
```

DESCRIPTION

The function determines and returns the number objects in the selection set.

ARGUMENTS

table – an object selection set

RETURN VALUE

It returns the number objects in the selection set.

3.273 sellist-null?

NAME

sellist-null?

SYNOPSIS

```
(sellist-null? table)
```

DESCRIPTION

The function determines whether the selection set is empty or not.

ARGUMENTS

table – an object selection set

RETURN VALUE

It returns #t if the selection set is empty otherwise it returns #f.

3.274 sellist-add

NAME

sellist-add

SYNOPSIS

```
(sellist-add table obj)
```

DESCRIPTION

The function adds a new object to the selection set.

ARGUMENTS

table – an object selection set
obj – an object descriptor as described in the "property" and "basic" modules.

RETURN VALUE

None.

3.275 sellist-member

NAME

sellist-member

SYNOPSIS

```
(sellist-member obj table)
```

DESCRIPTION

The function determines whether an object is in the selection set.

ARGUMENTS

obj – an object descriptor as described in the "property" and "basic" modules.
table – an object selection set

RETURN VALUE

It returns #t when the object is in the selection set, otherwise #f.

3.276 sellist-del

NAME

sellist-del

SYNOPSIS

```
(sellist-del table obj)
```

DESCRIPTION

The function deletes an object from the selection set.

ARGUMENTS

table – an object selection set
obj – an object descriptor as described in the "property" and "basic" modules.

RETURN VALUE

none.

3.277 sellist-map

NAME

sellist-map

SYNOPSIS

```
(sellist-map proc table)
```

DESCRIPTION

The function passes one-by-one all objects in the selection set to the user specified function, "proc". The return values of the "proc" function are collected into a list.

ARGUMENTS

proc – a function with one argument, where the argument is an object descriptor
table – an object selection set

RETURN VALUE

It returns a list.

3.278 sellist-for-each

NAME

sellist-for-each

SYNOPSIS

```
(sellist-for-each proc table)
```

DESCRIPTION

The function passes one-by-one all objects in the selection set to the user specified function, "proc". The return values of the "proc" function are discarded.

ARGUMENTS

proc – a function with one argument, where the argument is an object descriptor
table – an object selection set

RETURN VALUE

None.

3.279 selection-init-internal

NAME

selection-init-internal

SYNOPSIS

```
(selection-init-internal)
```

DESCRIPTION

The function sets all selection variables to their default values.

ARGUMENTS

None.

RETURN VALUE

None.

3.280 selection-clear-internal

NAME

selection-clear-internal

SYNOPSIS

```
(selection-clear-internal)
```

DESCRIPTION

The function clears the state of the selection mechanism. It resets all selection variables, resets the entry callback, the selection action, the prompt, and the mouse mode.

ARGUMENTS

None.

RETURN VALUE

None.

3.281 selection-init-keywords

NAME

selection-init-keywords

SYNOPSIS

```
(selection-init-keywords value keywords)
```

DESCRIPTION

The function works like the AUTOCAD initget function. The function sets the control "bits" that modify the selection and it also establishes the user selectable keywords.

ARGUMENTS

value – control bits, the possible values are described for the *SELECT_INIT* variable

keywords – a string, containing keywords separated by spaces. The format of the keyword is "keyword,shortcut" or "KEYword" where "KEY" is the shortcut.

RETURN VALUE

None.

3.282 selection-keyword-internal

NAME

selection-keyword-internal

SYNOPSIS

```
(selection-keyword-internal selection action)
```

DESCRIPTION

This function carries out the keyword selection processing. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no keyword is selected or "keyword" when a keyword was entered. The identified keyword is stored in the *SELECT_ANSWER* variable.

ARGUMENTS

selection – a string

action – a function to call when valid selection is made

RETURN VALUE

None.

3.283 selection-integer-internal

NAME

selection-integer-internal

SYNOPSIS

```
(selection-integer-internal selection action)
```

DESCRIPTION

This function carries out an integer number selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no integer is entered, "integer" when an integer was entered or "keyword" when a keyword was entered. The integer number or the keyword is stored in the *SELECT_ANSWER* variable.

ARGUMENTS

selection – an integer number or a string for keywords
action – a function to call when valid selection is made

RETURN VALUE

None.

3.284 selection-real-internal

NAME

selection-real-internal

SYNOPSIS

```
(selection-real-internal selection action)
```

DESCRIPTION

This function carries out a real number selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no number is entered, "real" when a number was entered or "keyword" when a keyword was entered. The number or the keyword is stored in the *SELECT_ANSWER* variable.

ARGUMENTS

selection – a number or a string for keywords
action – a function to call when valid selection is made

RETURN VALUE

None.

3.285 selection-string-internal

NAME

selection-string-internal

SYNOPSIS

```
(selection-string-internal selection action)
```

DESCRIPTION

This function carries out a string selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no string is entered or "string" when a string was entered. The string is stored in the *SELECT_ANSWER* variable. In this case it is not possible specify a keyword.

ARGUMENTS

selection – a string
action – a function to call when valid selection is made

RETURN VALUE

None.

3.286 selection-object-by-mouse

NAME

selection-object-by-mouse

SYNOPSIS

```
(selection-object-by-mouse objects)
```

DESCRIPTION

The function stores the selected objects in the *SELECT_MOUSE_OBJ* variable.

ARGUMENTS

objects – a list of strings that can be passed to the entry widget

RETURN VALUE

None.

3.287 selection-object-add-internal

NAME

selection-object-add-internal

SYNOPSIS

```
(selection-object-add-internal sellist)
```

DESCRIPTION

The function adds/removes the newly selected objects to/from the global selection set, *SELECT_LIST*. Whether the objects are added or removed are controlled by the *SELECT_OBJ_MODE* variable.

ARGUMENTS

sellist – a list of object descriptors

RETURN VALUE

The function returns the number of duplicates.

3.288 selection-object-prompt-update

NAME

selection-object-prompt-update

SYNOPSIS

```
(selection-object-prompt-update)
```

DESCRIPTION

The function updates the prompt, according to the last selected objects and according to the restrictions applied by the user. For example, the prompt is updated by the function to notify the user, that only points can be selected or that the last selected object was a quad.

ARGUMENTS

None.

RETURN VALUE

None.

3.289 selection-object-process

NAME

selection-object-process

SYNOPSIS

```
(selection-object-process selection action)
```

DESCRIPTION

The function processes the selection list, handles keywords and objects. The function assembles a list of valid object selection.

ARGUMENTS

selection – a list of strings and integers
action – a function to call when valid selection is made

RETURN VALUE

It returns the list of selected objects

3.290 selection-object-final

NAME

selection-object-final

SYNOPSIS

```
(selection-object-final objlist)
```

DESCRIPTION

The function carries out some final steps in the object selection. For example it updates the prompt, prints some messages to the user and updates the graphical screen.

ARGUMENTS

objlist – a list of object descriptors, the selected objects

RETURN VALUE

None.

3.291 selection-object-internal

NAME

selection-object-internal

SYNOPSIS

```
(selection-object-internal selection action)
```

DESCRIPTION

This function carries out the object selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no object is selected, "objects" when objects were selected or "keyword" when a keyword was entered. The keyword is stored in the *SELECT_ANSWER* variable while the selected objects are stored in the *SELECT_LIST* variable.

ARGUMENTS

selection – a list of a strings and integers
action – a function to call when valid selection is made

RETURN VALUE

None.

3.292 selection-object-single-update

NAME

selection-object-single-update

SYNOPSIS

```
(selection-object-single-update)
```

DESCRIPTION

The function is used during single object selection. When the user selects several objects, but only a single object can be selected then the program iterates through all selected objects. This function displays the current object during iteration.

ARGUMENTS

None.

RETURN VALUE

None.

3.293 selection-object-single

NAME

selection-object-single

SYNOPSIS

```
(selection-object-single)
```

DESCRIPTION

This function is used during single object selection. When the user selects several objects, but only a single object can be selected then the program iterates through all selected objects. This function is the iteration function.

ARGUMENTS

None.

RETURN VALUE

None.

3.294 selection-coord-by-mouse

NAME

selection-coord-by-mouse

SYNOPSIS

```
(selection-coord-by-mouse points)
```

DESCRIPTION

The function is used in the coordinate selection by a mouse click. In this case it is possible, that the mouse selects several points under each other. Since for a point coordinate only one point is required, this function checks whether a single point has been selected and warns the user if several points are selected.

ARGUMENTS

points – list of point object descriptors

RETURN VALUE

None.

3.295 selection-coord-internal

NAME

selection-coord-internal

SYNOPSIS

```
(selection-coord-internal selection action)
```

DESCRIPTION

This function carries out a coordinate selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no point is entered or "point" when a point was entered. The point coordinates are stored in the *SELECT_X*, *SELECT_Y* and *SELECT_Z* variables. When a keyword was entered the *SELECT_STATE* variable stores "keyword" and the keyword string is stored in the *SELECT_ANSWER* variable.

ARGUMENTS

selection – a list of strings
action – a function to call when valid selection is made

RETURN VALUE

None.

3.296 selection-store-base

NAME

selection-store-base

SYNOPSIS

```
(selection-store-base)
```

DESCRIPTION

The function is used for distance selection, at the first stage. When a point is selected as a base point this function stores this base point in the *SELECT_BASE* variable.

ARGUMENTS

None.

RETURN VALUE

None.

3.297 selection-calc-distance

NAME

selection-calc-distance

SYNOPSIS

```
(selection-calc-distance)
```

DESCRIPTION

The function calculates the components of the distance between the base point and the currently selected point. The calculated distance components are stored in the *SELECT_X*, *SELECT_Y* and *SELECT_Z* variables.

ARGUMENTS

None.

RETURN VALUE

None.

3.298 selection-dist-point-internal

NAME

selection-dist-point-internal

SYNOPSIS

```
(selection-dist-point-internal selection action)
```

DESCRIPTION

This function carries out a distance selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no distance is selected or "dist" when a distance was selected. The components of the distance are stored in the *SELECT_X*, *SELECT_Y* and *SELECT_Z* variables. When a keyword was entered the *SELECT_STATE* variable stores "keyword" and the keyword string is stored in the *SELECT_ANSWER* variable. The selection uses the same internal keywords as for the point selection.

ARGUMENTS

selection – a list of strings and numbers
action – a function to call when valid selection is made

RETURN VALUE

None.

3.299 selection-file-internal

NAME

selection-file-internal

SYNOPSIS

```
(selection-file-internal selection action)
```

DESCRIPTION

This function carries out a file selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no file is selected or "string" when a file was selected. The selected file name is stored in the *SELECT_ANSWER* variable. No keyword can be selected with this selection mechanism.

This selection is different from other selection mechanism, since in this case if the ENTER is pressed when the entry line is empty, then it opens a file selection dialog.

ARGUMENTS

selection – a list of strings and numbers
action – a function to call when valid selection is made

RETURN VALUE

None.

3.300 selection-color-internal

NAME

selection-color-internal

SYNOPSIS

```
(selection-color-internal selection action)
```

DESCRIPTION

This function carries out a color selection. When the "action" function is called the value of the *SELECT_STATE* variable can be "cancel" when no color is selected or "color" when a color was selected. The selected color is stored in the *SELECT_ANSWER* variable. When a keyword is entered the *SELECT_STATE* variable is "keyword" and the *SELECT_ANSWER* variable stores the identified keyword string.

When a color is selected the *SELECT_ANSWER* variable can be a 4 element list or a one element list. When the list has four elements, then they denote the red, green, blue and alpha components of the color. When the list hs one element then it denotes the index of the color from the *GL_COLOR_LIST* list defined in the render module.

ARGUMENTS

selection – a list of strings and numbers
action – a function to call when valid selection is made

RETURN VALUE

None.

3.301 selection-setup

NAME

selection-setup

SYNOPSIS

```
(selection-setup func)
```

DESCRIPTION

The function should be called during the initialisation of the SX program. It sets the *SELECT_VIEW_SETUP* variable to the specified function and it also resets all selection variables.

ARGUMENTS

func – the function that is used to setup the viewing mode during selection

RETURN VALUE

None.

3.302 selection-closest-set!

NAME

selection-closest-set!

SYNOPSIS

```
(selection-closest-set! flag)
```

DESCRIPTION

The function sets the *SELECT_CLOSEST* variable to #t or #f.

ARGUMENTS

flag – it can be #t or #f

RETURN VALUE

None.

3.303 selection-closest-get

NAME

selection-closest-get

SYNOPSIS

```
(selection-closest-get)
```

DESCRIPTION

The function returns the value of the *SELECT_CLOSEST* variable.

ARGUMENTS

none.

RETURN VALUE

It returns the value of the *SELECT_CLOSEST* variable.

3.304 selection-select-list-set!

NAME

selection-select-list-set!

SYNOPSIS

```
(selection-select-list-set! objects)
```

DESCRIPTION

The function stores a new object selection set in the *SELECT_LIST* variable.

ARGUMENTS

objects – an object selection set

RETURN VALUE

None.

3.305 selection-select-x-set!

NAME

selection-select-x-set!

SYNOPSIS

```
(selection-select-x-set! x)
```

DESCRIPTION

The function stores a new value in the *SELECT_X* variable.

ARGUMENTS

x – a new x component

RETURN VALUE

None.

3.306 selection-select-y-set!

NAME

selection-select-y-set!

SYNOPSIS

```
(selection-select-y-set! y)
```

DESCRIPTION

The function stores a new value in the *SELECT_Y* variable.

ARGUMENTS

y – a new y component

RETURN VALUE

None.

3.307 selection-select-z-set!

NAME

selection-select-z-set!

SYNOPSIS

```
(selection-select-z-set! z)
```

DESCRIPTION

The function stores a new value in the *SELECT_Z* variable.

ARGUMENTS

z – a new z component

RETURN VALUE

None.

3.308 selection-select-answer-set!

NAME

selection-select-answer-set!

SYNOPSIS

```
(selection-select-answer-set! answer)
```

DESCRIPTION

The function stores a new value in the *SELECT_ANSWER* variable.

ARGUMENTS

answer – a new selection result

RETURN VALUE

None.

3.309 selection-select-state-set!

NAME

selection-select-state-set!

SYNOPSIS

```
(selection-select-state-set! state)
```

DESCRIPTION

The function stores a new value in the *SELECT_STATE* variable.

ARGUMENTS

state – a new selection state

RETURN VALUE

None.

3.310 selection-push

NAME

selection-push

SYNOPSIS

```
(selection-push)
```

DESCRIPTION

The function collects all selection related variables and stores their current value in the *SELECT_STACK* variable. The selection related variables includee the current option menu and the text of the prompt. In this way the state of selection is stored and can be restored at a later stage.

ARGUMENTS

None.

RETURN VALUE

None.

3.311 selection-pop

NAME

selection-pop

SYNOPSIS

```
(selection-pop)
```

DESCRIPTION

The function restores a previously stored selection state from the *SELECT_STACK* variable.

ARGUMENTS

None.

RETURN VALUE

None.

3.312 selection-crossing-enter

NAME

selection-crossing-enter

SYNOPSIS

```
(selection-crossing-enter text pt)
```

DESCRIPTION

This function can be used to setup the crossing selection. The function only modifies the text in front of the command line, changes the picking mode and sets the corners of the selection window

ARGUMENTS

text – string which appears in front of the command line
pt – the first two dimensional point of the selection window as a list. It can be a #f value as well.

RETURN VALUE

None.

3.313 selection-crossing-apply

NAME

selection-crossing-apply

SYNOPSIS

```
(selection-crossing-apply pt)
```

DESCRIPTION

The function sets the second point of the selection window or area. It stores this second point in the *SELECT_WIN_CORNER_2* variable. The function also updates the graphical screen.

ARGUMENTS

pt – the second two dimensional point of the selection window as a list.

RETURN VALUE

None.

3.314 selection-crossing-quit

NAME

selection-crossing-quit

SYNOPSIS

```
(selection-crossing-quit)
```

DESCRIPTION

This function leaves the crossing object selection mechanism and returns to normal object selection.

ARGUMENTS

None.

RETURN VALUE

None.

3.315 selection-init

NAME

selection-init

SYNOPSIS

```
(selection-init callback action text opt_menu gl_mode control keywords)
```

DESCRIPTION

This function sets up the selection mechanism for a command. The fcuntion resets all state variables and set them to new values. However the function does not affect the *SELECT_STACK* variable, therefore it does not affect the recursive selection calling.

ARGUMENTS

callback – this function will be called when the user pressed an ENTER or SPACE character or clicked the right mouse button. This function also validates the entered data.
action – once a valid data has been entered this function is called to process the data
text – this string will be printed in front of the command line as a prompt
opt_menu – optional menu for the right hand side of the window
gl_mode – selection mode, see the *PICK_MODE* variable
control – list of control "bits", symbols, controlling the selection process, see the *SELECT_INIT* variable
keywords – a string containing the keywords, separated by spaces

RETURN VALUE

None.

3.316 selection-clear

NAME

selection-clear

SYNOPSIS

```
(selection-clear)
```

DESCRIPTION

This function clears all selection variables. It also ensures that there is no selection mechanism on the stack. The stack is also cleared. To return to another, stored selection mechanism use the "selection-return" function. The function also ensures that if there was a successfull object selection then the selected objects are stored and they can be recalled later as previously selected objects, see the *SELECT_PREVIOUS* variable.

ARGUMENTS

None.

RETURN VALUE

None.

3.317 selection-return

NAME

selection-return

SYNOPSIS

```
(selection-return)
```

DESCRIPTION

If there is no selection mechanism stored on the selection stack the function is identical to "selection-clear". However when there is a stored selection mechanism then this function restores that selection and the restored selection can continue.

ARGUMENTS

None.

RETURN VALUE

None.

3.318 selection-keyword

NAME

selection-keyword

SYNOPSIS

```
(selection-keyword)
```

DESCRIPTION

This is the callback function for keyword selection. The function only allows keyword selection! It communicates with other functions through global variables, see the "selection-keyword-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.319 selection-integer

NAME

selection-integer

SYNOPSIS

```
(selection-integer)
```

DESCRIPTION

This is the callback function for integer selection. The function also allows keyword selection! It communicates with other functions through global variables, see the "selection-integer-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.320 selection-real

NAME

selection-real

SYNOPSIS

```
(selection-real)
```

DESCRIPTION

This is the callback function for real number selection. The function also allows keyword selection! It communicates with other functions through global variables, see the "selection-real-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.321 selection-string

NAME

selection-string

SYNOPSIS

```
(selection-string)
```

DESCRIPTION

This is the callback function for string selection. The function only allows string selection! It communicates with other functions through global variables, see the "selection-string-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.322 selection-object-filter

NAME

selection-object-filter

SYNOPSIS

```
(selection-object-filter object)
```

DESCRIPTION

This function determines whether the selected object is allowed by the the "selection-init" function and by the user. First the restrictions of the "selection-init" function are applied than the user exclusions. For example the "selection-init" function may allow the selection of all objects but the user restricts the selection only to points.

ARGUMENTS

object – the object descriptor to filter or check

RETURN VALUE

It returns #t if the object can be selected otherwise #f is returned.

3.323 selection-add-ref-nodes

NAME

selection-add-ref-nodes

SYNOPSIS

```
(selection-add-ref-nodes expand)
```

DESCRIPTION

The function adds all those points to the selection set which are referenced by the already selected objects, for example: links, triangles, etc.

ARGUMENTS

expand – not required, FIXME

RETURN VALUE

None.

3.324 selection-object

NAME

selection-object

SYNOPSIS

```
(selection-object)
```

DESCRIPTION

This is the callback function for object selection. The function also allows keyword selection! It communicates with other functions through global variables, see the "selection-object-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.325 selection-coord

NAME

selection-coord

SYNOPSIS

```
(selection-coord)
```

DESCRIPTION

This is the callback function for coordinate selection. The function also allows keyword selection! It communicates with other functions through global variables, see the "selection-coord-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.326 selection-dist-point

NAME

selection-dist-point

SYNOPSIS

```
(selection-dist-point)
```

DESCRIPTION

This is the callback function for distance selection. The function also allows keyword selection! It communicates with other functions through global variables, see the "selection-coord-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.327 selection-file

NAME

selection-file

SYNOPSIS

```
(selection-file)
```

DESCRIPTION

This is the callback function for file name selection. The function only allows file name selection! It communicates with other functions through global variables, see the "selection-file-internal" function.

ARGUMENTS

None.

RETURN VALUE

None.

3.328 selection-color

NAME

selection-color

SYNOPSIS

```
(selection-color)
```

DESCRIPTION

This is the callback function for color selection. The function allows the selection of an indexed color (integer) or an RGBA color, defined by four numbers in the range of zero and one. The function also allows keyword selection! It communicates with other functions through global variables.

ARGUMENTS

None.

RETURN VALUE

None.

3.329 selection-cancel

NAME

selection-cancel

SYNOPSIS

```
(selection-cancel)
```

DESCRIPTION

The function is generally used when the ESC key has been pressed. In this case all selection mehcanism should be reset and the state of the program should return to its default state. However it is only possible if all dialog windows are closed.

ARGUMENTS

None.

RETURN VALUE

It returns #t if the program returned to its default state otherwise #f.

3.330 trackball-calc-quaternion

NAME

trackball-calc-quaternion

SYNOPSIS

```
(trackball-calc-quaternion p1x p1y p2x p2y)
```

DESCRIPTION

The function simulates a trackball for modell rotation. It projects the positions of the mouse onto a virtual trackball, then calculates how much rotation that means in the model.

ARGUMENTS

p1x – the x component of the current mouse position
p1y – the y component of the current mouse position
p2x – the x component of the previous mouse position
p2y – the y component of the previous mouse position

RETURN VALUE

It returns a quaternion as a four element list.

3.331 trackball-rotmatrix

NAME

trackball-rotmatrix

SYNOPSIS

```
(trackball-rotmatrix q)
```

DESCRIPTION

The function builds an OpenGL rotation matrix from a quaternion.

ARGUMENTS

q – a quaternion, as a four element list

RETURN VALUE

It returns an OpenGL rotation matrix, a 4 by 4 matrix, as a 16 element list.

3.332 trackball-add-quaternions

NAME

trackball-add-quaternions

SYNOPSIS

```
(trackball-add-quaternions q1 q2)
```

DESCRIPTION

Provided two rotations expressed as quaternions, the function calculates the equivalent single rotation and returns it.

ARGUMENTS

q1 – a quaternion, as a four element list
q2 – a quaternion, as a four element list

RETURN VALUE

It returns a quaternion, as a four element list.

3.333 undo-changed-set

NAME

undo-changed-set

SYNOPSIS

```
(undo-changed-set flag)
```

DESCRIPTION

This function modifies the *CHANGED* variable.

ARGUMENTS

flag – the new value of the *CHANGED* variable, #t or #f

RETURN VALUE

None.

3.334 undo-changed?

NAME

undo-changed?

SYNOPSIS

```
(undo-changed?)
```

DESCRIPTION

This function returns the value of the *CHANGED* variable, #t or #f.

ARGUMENTS

None

RETURN VALUE

It returns the current value of *CHANGED*.

3.335 redo-get

NAME

redo-get

SYNOPSIS

```
(redo-get)
```

DESCRIPTION

This function returns the list of commands that can be redone.

ARGUMENTS

None

RETURN VALUE

It returns the value of the *REDO* variable.

3.336 undo-get-list

NAME

undo-get-list

SYNOPSIS

```
(undo-get-list)
```

DESCRIPTION

This function returns the list of commands that can be undone. The list also contains all the necessary arguments for the commands.

ARGUMENTS

None

RETURN VALUE

It returns the value of the *UNDO* variable.

3.337 undo-get-size

NAME

undo-get-size

SYNOPSIS

```
(undo-get-size)
```

DESCRIPTION

This function returns the number of undoable commands.

ARGUMENTS

None

RETURN VALUE

It returns the length of the *UNDO* list.

3.338 undo-redo-reset

NAME

undo-redo-reset

SYNOPSIS

```
(undo-redo-reset)
```

DESCRIPTION

This function resets the *REDO* and the *UNDO* variables to empty lists, which also means that the recording of user operations start after this function call.

ARGUMENTS

None.

RETURN VALUE

None.

3.339 redo-add

NAME

redo-add

SYNOPSIS

```
(redo-add com)
```

DESCRIPTION

This function adds a new command to the redoable command list, *REDO*.

ARGUMENTS

com – the command, which can be redone. It is a list where the first element is an "apply"-able function and the rest of the list contains the requires arguments for the function.

RETURN VALUE

None.

3.340 undo-add

NAME

undo-add

SYNOPSIS

```
(undo-add com)
```

DESCRIPTION

This function adds a new command to the undoable command list, *UNDO*.

ARGUMENTS

com – the command, that when is executed will cancel the effect of the current operation. As a data structure it is a list where the first element is an "apply"-able function and the rest of the list contains the requires arguments for the function.

RETURN VALUE

None.

3.341 undo-add-op-mark

NAME

undo-add-op-mark

SYNOPSIS

```
(undo-add-op-mark)
```

DESCRIPTION

This function adds an operational mark to the undo list. Operational marks are used internally to undo a group of operations carried out by a command.

ARGUMENTS

None.

RETURN VALUE

None.

3.342 undo-add-user-mark

NAME

undo-add-user-mark

SYNOPSIS

```
(undo-add-user-mark)
```

DESCRIPTION

This function adds a user mark to the undo list. User marks are placed by user and generally separate different stages to which the user can come back at a later time (of course only by undoing the previous stages).

ARGUMENTS

None.

RETURN VALUE

None.

3.343 undo-one-operation

NAME

undo-one-operation

SYNOPSIS

```
(undo-one-operation)
```

DESCRIPTION

This function undoes one command. This means that it will execute the functions in the undo list as long as it either encounters an operation mark or a user mark or the undo list is empty.

ARGUMENTS

None.

RETURN VALUE

None.

3.344 undo-all-operations

NAME

undo-all-operations

SYNOPSIS

```
(undo-all-operations)
```

DESCRIPTION

This function undoes everything, it executes every function in the undo list without stopping at operation or user marks.

ARGUMENTS

None.

RETURN VALUE

None.

3.345 undo-until-user-mark

NAME

undo-until-user-mark

SYNOPSIS

```
(undo-until-user-mark)
```

DESCRIPTION

This function undoes everything until it encounters a user placed mark. Once the user placed mark is encountered it stops and returns.

ARGUMENTS

None.

RETURN VALUE

None.

3.346 redo-operation

NAME

redo-operation

SYNOPSIS

```
(redo-operation)
```

DESCRIPTION

This function executes all functions in the redo list. The process does not stop at user or operational marks. In fact no mark should be in the redo list.

ARGUMENTS

None.

RETURN VALUE

None.