The

VRR

User Manual

Table of Contents

1 Introduction

VRR (Vector-based gRaphical editoR) is an application which has been developed mainly for creating illustrations of mathematical articles. The final result will be usable in books or articles (e.g. TEX documents).

Rather than having many great faeatures for creating modern art, the editor has a simple but powerful operation set: creating, manipulating and transforming basic graphical primitives, which are points, segments, rational Bezier curves, elliptic arcs etc. The objects can be determined not only by absolute coordinates, but by geometrical dependancies on other objets (intersections etc.) as well. When an object changes, dependant objects are recalculated automatically.

VRR has a sophisticated system for working with texts. Apart from common text, allows to create TEX text defined with TEX code.

The editor is be compatible with existing drawing and typesetting programs, able to import from end export to files in common data formats (export to PDF, PS, EPS and SVG, import from IPE5 and SVG).

It is freely available and will run on Linux, using a graphical interface based on the X Window System and GTK. We suppose a user willing to spend some time by learning how to use it, which will bring him much joy later when using some powerful but non-intuitive features. Therefore, our editor should not be 'for everyone' but an advanced tool for advanced users.

The editor can be also easily extended by a plugin or by a program written in its built-in scripting language.

2 Installation Instructions

Prerequisities

For installation of VRR, your system must fulfil these prerequisities:

- GNU make
- gcc 3.0 or higher
- gawk
- perl
- GTK+ version 2.6.0 or higher
- guile-1.6
- kpathsea
- fontconfig 2.3.1 or higher
- pdfTeX
- zlib
- libxml2

In most cases you will need the development packages (libXXX-dev) to compile VRR.

Program is also able to use libpaper library, but it is not mandatory for successful compilation.

Compillation

Download the package from `http://vrr.ucw.cz/`. Unpack the archive and change directory to its root directory ('`cd`'). Then type '`make c onfig`' to configure your installation automatically or run `build/configure` to configure your installation by hand. If you want to recreate `build/configure`, type '`make build/configure`'.

Then type '`make final`' to compile the program.

You can remove all generated binaries by typing '`make clean`'.

Installation

As root type '`make install`'. Two binaries, the standard version '`vrr`' and non-GUI guile console '`guile-vrr`' version will be installed.

Local compilation

If you do not want to install the program, you can still compile and run it. Follow the instructions in chapter Compilation, but type '`make`' only to compile the program.

Running the application

Change directory to '`run`' in the source tree and type '`bin/vrr`'. The program must be run this way to make sure all icons and similar resources are found. To run the guile-vrr version (guile console only), type '`bin/guile-vrr`'.

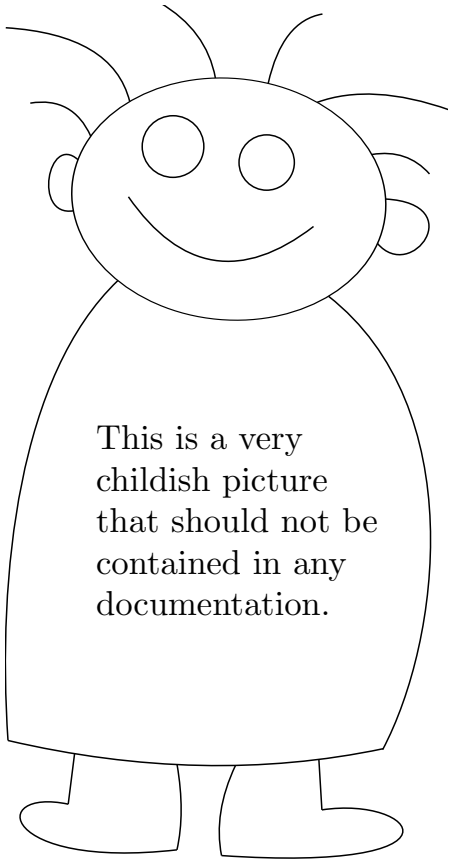
3 Tutorial

This part of documentation contains many examples of pictures created in VRR with detailed tutorial how to create them. The pictures should cover all features available in VRR.

This is a short and very descriptive paragraph that explains how to use VRR.

The following paragraphs are copied from our website to make this longer.

The buttons in the toolbar can be used to create new graphic objects, switch between factory states and set snap modes (see 3.3 Go Factory). On toolbars, there are some groups of buttons, set apart by button with horizontal separator line in it. Clicking on this separator button rolls up (down) all buttons which come under this group. In the statusbar of the view, you can see hints about what to do in the particular factory state, and the current zoom and rotation settings of the view.



There is also a pencil icon in the statusbar. It disappears and appears again from time to time and marks the views of the current context page (see Context). All context actions (like keyboard shortcuts or so) are related to the current context.

This is an example of how to include a name of function: for example, `group_get_first_go(struct go_group * group)` or something like that.

The main part of the view is a drawing area where the geometrical objects contained in the page are drawn. The area is clickable and mouse actions have the following meaning:

$$\sum_{i=0}^{n} \binom{n}{i} i$$

To include a mathematical expression in the paragraph mode, use the command `mat` like this: $1+\frac{1}{x}$ and to include a mathematical expression in the display mode, use the command `mmat` like this:

$$\int_0^\infty \frac{1}{x}\mathrm{d}x$$

This paragraph should surround the picture of ornament1. There is also a pencil icon in the statusbar. It disappears and appears again from time to time and marks the views of the current context page (see Context). All context actions (like keyboard shortcuts or so) are related to the current context. All context actions (like keyboard shortcuts or so) are related to the current context. All context actions (like keyboard shortcuts or so) are related to the current context. All context actions (like keyboard shortcuts or so) are related to the current context. All context actions (like keyboard shortcuts or so) are related to the current context.

This is a blabol.

4 The Anatomy of a Picture

This chapter explains the anatomy of pictures: what are hangers, anchors, graphical objects, what are geometric depencies, what is the difference between anchor rehang and transformation, how propagation of changes works, ...

The original idea was to start with the pictures introduced in the previous chapter and explain everything on them, show some dependency diagrams etc.

5 The Anatomy of the Graphical User Interface

This chapter describes all main parts of the GUI: the GO factory, the command structure, the visualisation, ... It describes the function of all important widgets in all windows.

5.1 Windows

5.1.1 The Main Window

After you run the application, the main window opens. It allows you to do some basic actions not connected to any particular document (create new documents, load documents from files, open some windows as Universe Browser or Undo History, set global VRR settings or open help files). These windows will be described in the following chapters.

The main window can be hidden and opened again without quitting the whole program (provided that you still have some view windows opened).

5.1.2 The View

The purpose of the view window is to display the contents of a document's page. Each view displays one page, while a page can be displayed in several independent views. All changes performed to the page are displayed in all views at once. All views displaying the same page can be used interchangebly. By closing the view, you do not delete the page nor the document containing the page, you only close the view. After creating a new document or opening the existing one (with at least one page), a view for the first page is opened.

The buttons in the toolbar (on the left side of the view window) can be used to create new graphic objects, set snap modes etc. (see Section 5.2 [The Graphic Objects Factory], page 8). On toolbars, there are some groups of buttons, set apart by button rolls up/down all buttons coming under this group. In the statusbar of the view, you can see hints about what to do in the particular factory state, and the current zoom and rotation settings of the view.

There is also a pencil icon in the statusbar. It marks the views of the current context page (see Section 7.1 [What is context?], page 16). All context actions (like keyboard shortcuts or so) are related to the current context.

The main part of the view is a drawing area where the geometrical objects contained in the page are drawn. The area is clickable and mouse actions have the following meaning:

- The **left mouse button** does various things according to the state of the GO factory. For example, when creating a new graphic object, it positions the next point of the graphic object. Almost all actions of the GO factory are done by clicking the left mouse button.
- The **middle mouse button** while holding the Ctrl key, makes the position where you clicked the center of the view. Otherwise it moves the current visible region of the page.
- The **mouse wheel** changes the zoom of the view.
- The **right mouse button** opens the context menu.

Use scrollbars to move in the drawing area. By clicking on the scrollbar arrows, you can move even farther that the current editable area borders and thus have the area enlarged.

In the corners between scrollbars and rulers, there are four buttons. The Upper buttons rotate the whole view according to the arrow's direction. The lower right button shows the View Navigator window containing the preview of whole page. The green rectangle in the preview represents the currently visible region of the page in the view. You can move it and change the visible region of the view accordingly. Enter new values of zoom and rotation to adjust preview. Button Reset sets zoom to 100.0% and rotation to 0 degrees.

5.1.3 Universe Browser

The universe browser shows you the tree structure of the whole universe. Here the left mouse button performs selection in the style similar as in Gimp: `Shift + click` adds to selection, `Ctrl`

`+ Click` removes from selection, while a single click clears the current selection and selects the clicked objects.

The selected objects are marked with a color. As the program has two different selections - selection of "namespace objects" (documents and pages) and selection of graphical objects (each of them is showed using a different color). The selections are orthogonal, hence changed independently.

The **right mouse button** opens a context menu. Context is the collection of the most significant states of editor (the current selection, the current page, the last clicked object and the current window). See also Section 7.1 [What is context?], page 16. The menu items are supported to be executed on the current selection and may be disabled accordingly. If there is no selection some other objects are considered current.

5.1.4 The Undo History Window

The undo history window can be open for a page or for the whole universe. It shows the list of all actions that have been done or undone on the particular page or on the universe. All action has a label, describing the operation type. The undone actions are striked trought.

Buttons in the toolbar have the following meaning:

- first button: undo to the previous action in the list
- second button: jump to the selected undo item (select the position by clicking on the chosen item)
- third button: redo the next action in the list

Number of items in undo history is limited by memory used by undo items (see Section 5.1.7 [VRR Settings], page 8). Therefore leading items of the list can be removed occassionally.

5.1.5 The Property Window

The property window allows changing object's properties. Each object, according to its type (segment, circle etc.), contains some specific properties, which can be - in general - changed.

Properties with dimensions are associated with a dimension unit. By setting the unit, you modify the display value of the property, not the property itself (internal value stays the same). You can also create and use your own units (click on "New ..." in unit combo box). When creating a new unit, you enter the multiplier against the base unit.

For some properties, there is a more specialized editor, like the color editor, text editor (see Section 5.1.6 [The Text Editor], page 7), filename editor etc. In that case, there is a button in the property window that opens the editor.

You can also add your own properties (the "Add" button) or delete properties you've created (the "Delete" button).

5.1.6 The Text Editor

When creating a new text (or TeX text), you first specify a hanger – location of the text reference point. Then the Text editor opens. The Text and TeX text editors differ slightly, but basically they are the same.

In case of the ordinary text, there are additional widgets for choosing the font and font size. The font list contains 10 last used fonts (if no fonts were used so far, it's 10 first fonts found by fontconfig). If you want to load all possible fonts, click on the button Load all fonts and the list in combo box will be filled with all installed fonts found (this may take a while). The font size is in points.

You can specify the position of the chosen reference point relatively to the text bounding box. For example, if you set the refpoint values to 0.5 and 0.5, the text will be positioned in such a way that center point of its bounding box equals the reference point.

Write the text to be displayed into the text area or load it from a file in the UTF-8 character encoding. Empty text string is invalid value for text object.

Any changes you have made take effect after pressing the "Refresh" button or by pressing the `Ctrl + Return` keyboard shortcut.

5.1.7 VRR Settings

The Properties of universum window (opened from main window by clicking on Settings and Set VRR Options) allows to change global editor settings.

You can set there maximum volume of undo history, terminal (VRR will use it), WWW browser (will be open when viewing the help files), external text editor used while creating texts (see Section 5.1.6 [The Text Editor], page 7), snap tolerance or universum name. All changes will take effect immidiately after these changes are made.

5.2 The Graphic Objects Factory

The GO factory is a mechanism for creating GOs. It works similarly to a regular automaton: You set the starting state (e.g. `"Create a segment"`) and then choose the desired arguments of the operation (in this case, two hangers), step by step. By pressing the BackSpace key, you return one step back, by pressing Esc, you cancel the creating process and delete the GO that is being created.

The GO factory is global for the whole VRR. It can operate on at most one page at the same time. Setting some other page as the context page resets the process. Any action incompatible with the GO factory to a start state during creating a GO, the process is cancelled.

The interface for working with the GO factory is contained in the view. Toolbar buttons switch between factory states, by clicking the drawing area you position the points as arguments of the current operation, select objects, move the transformation gadgets and thus transform them, etc. The argument required in the current state is described in the status bar.

Once you choose the desired argument (usually by clicking an object with the left mouse button). In case of success the factory moves to another state and allows you to choose another argument. In case of a failure (e.g. when creating a circumscribed circle of three points and having chosen the first two points as equal) the factory reports an error and asks you to choose the last argument again. The error can occur for various reasons: when trying to craete a circle circumscribed to three colinear points, or move a fully dependent object, for example.

When the process is finished, the fatory returns to the starting state again so that you can create another GO of the same type.

The GO factory has the following kinds of states/modes: select `+` transform mode, anchor rehanging mode and go creating modes. These modes will be described in more details in the following chapters.

5.2.1 Select and Transform Mode

When in the select and transform mode, the selection works similarly as in the Universe Browser (see Section 5.1.3 [Universe Browser], page 6). By clicking a GO, you select it, by `Shift +` clicking, you add it to the current selection and by `Ctrl +` clicking, you remove it from the selection.

The bounding box of the selected objects is framed with a red line containing the transformation gadgets. By dragging there, you transform the selected objects with affine transformations. Their functions are:

- the red center point and the bounding box lines **move**
- the edge centers **scale**
- the corners **scale by keeping the ratio**

For scale and rotate transformations, the red centerpoint is the fixpoint. The gray centerpoint shows the current center of the bounding box (which may differ from the red one).

When holding the `Shift` key, different gadgets appear. There is a blue segment running out of the centerpoint - the axis. For skew operations, it represents the fixed points. The functions are:

- **rotate** by moving a corner
- **move the centerpoint** by clicking and dragging it
- **move the axis point** by clicking and dragging it

- **skew** by dragging a center of the edge

In case that the bounding box of selected object is (almost) non two-dimensional, only some gadgets are shown and only some transformations are applicable. There are **move**, **scale** and **rotate** for horizontal and vertical lines and move for points.

5.2.2 Anchor Rehang Mode

This mode allows you to rehang anchors, which you might imagine as slots for something specified during the GO creation. For example, if you created a segment, you hung two anchors on the selected points (hangers), the startpoin anchor and the endpoint anchor (see kernel documentation for more information). Now you might want to change the anchors position - rehang one to a different hanger.

First, choose a GO by selecting it. Then choose on of its anchors by clicking it. Then move the mouse cursor, you can see the anchor moving and causing the anchor's GO to be recomputed. Then click a hanger, the anchor gets hung on it.

Anchors are displayed as green triangles, whereas hangers are blue triangles.

5.2.3 GO Creating Modes

Almost all factory icons represent states for creating new GOs. To create a GO, press such an icon. Each GO is determined by some things: points (hangers) and numbers, maybe more. You will be asked to specify all information needed - follow the instructions in the statusbar of the view (see Section 5.1.2 [The View], page 6). Positional information is specified by clicking the drawing area, numeric data can be edited in the property window.

The following chapters describe creating a GO step by step:

The following example tries to clarify this.

5.2.3.1 Decoration Point

Decoration point represents the point with defined shape. Click on the view to set its position. The point with last used style will be created. You can change it's properties (select object, right click and choose Edit/GO properties) as vertices, rotation, radius, position and name. Vertices mean the number of vertices (0 will create the circle, positive number N will create a regular n-angle).

<screenshot>

5.2.3.2 Intersection Point

Intersection point needs two graphical objects and creates in their intersection. You can't change it's position (it is dependent on both graphical objects).

<screenshot>

5.2.3.3 Segment

To create segment, you must select two end points. Then you can change it's properties.

<screenshot>

5.2.3.4 Quadratic Bezier

Quadratic bezier curveis defined by three control points. Weights for these points are automatically set to 1. In property editor, you can change all object properties.

<screenshot>

5.2.3.5 Cubic Bezier

Cubic bezier curve is being created similarly to quadratic bezier (see Section 5.2.3.4 [Quadratic Bezier], page 9). You must click 4 control points, cubic bezier curve with weights set to 1 will be created. You can edit curve's properties in the property editor (see Section 5.1.5 [The Property Window], page 7).

<screenshot>

5.2.3.6 Arc

There are two ways how to create an arc:

- **by three points** - The first point represents the arc beginning, the second one can be any point in the circumference (not necesarilly between the arc beginning and arc end) and the last one is the arc's end point
- **by center and radius** - Click the center position and the template of the arc will be created. Change it's properties in the property editor (see Section 5.1.5 [The Property Window], page 7)

<screenshot>

5.2.3.7 Circle

There are several ways of creating the circle:

- **by three points** - Click three non-colinear points on the circumference
- **center and point on the circumference** - Choose the center position and then click one point on the circumference
- **center and radius** - Click the center and the circle template with predefined radius will be created. Edit properties in the property editor (see Section 5.1.5 [The Property Window], page 7) to satisfy your needs

<screenshot>

5.2.3.8 Ellipse

5.2.3.9 Elliptic Arc

5.2.3.10 TeXText

5.2.3.11 Text

5.2.3.12 An example of creating a new GO

5.2.3.13 Snap

When choosing the arguments, you usually choose points (more precisely, hangers). If no snap mode is set, the chosen point always becomes exactly the position where you clicked. If you turn some snap modes on, the chosen point may be aligned to something near the position you clicked. The possible snap modes are:

- **snap to hangers**: if there is an existing hanger near the position where you clicked, it is chosen rather than the position
- **snap to grid**: dtto, with the grid points
- **snap to lines**: the point is put on a near line of another object, if there is such
- **snap to intersections**: dtto, with an intersection of two curves near the position where you clicked

The snap modes are independent and can be switched on and off and combined arbitrarily. If several modes are on, the nearest matching point is chosen (but not farther than a certain tolerance in pixels).

6 Scheme

Guile-vrr and scheme console are used for program controlling by scheme commands. Except for GUI and viewing images, you can access all the program functionality as image creating, loading, editing, saving, exporting or importing.

To learn about the Scheme programming language, see this Scheme Tutorial - http://cs.www.edu/~cs_dept/KU/PR/Scheme.html. To learn even more, see the Structure and Interpretation of Computer Programs book (http://mitpress.mit.edu/sicp/full-text/book/book.html) or the Revised Report on the Algorithmic Language Scheme (http://www.schemers.org/Documents/Standards/R5RS/HTML/).

6.1 VRR scheme data types

VRR proxy is a scheme object used for representing VRR entities. Proxies are compared with command `eq?` (if p1 and p2 are proxies of the same entity, then (eq? p1 p2) returns true). There are four kinds of proxies (obj, go, anchor, hanger) and they are associated with VRR entities of the same name.

Obj

Object is classified by their type and can be document or tlo. Document contains TLOs (TLO means top level object).

GO

Graphical object (GO) represents geometrical object. GOs can be classified by their type and subtype. Different types of GO represent different geometrical objects; different subtypes of one GO type mean different ways of GO representation (e.g. circle can be defined by the center and radius, center and one point on the circumference, by three non-colinear points...). Subtypes of GO are unique over all GO types and every GO has its own one, so if you want to dispatch GOs by subtypes, you don't have to dispatch them by types first. Short description of GO types and their subtypes follows.

In that description, there is an associated list of hangers after the name of the type and symbol '>' (each GO of that type has that hangers). After the subtype name and symbol '<', there is a list of specified defining parameters (hangers, numbers etc.). Each GO of that subtype needs this parameters as arguments for function which creates GO of that subtype. Each parameter of GO which is not enclosed in '*' also describes the anchor of that name. Enclosed parameters not mentioned in specific description are numbers.

GO type: **point** > center

GO subtype: **point** < center

GO type: **segment** > endpoint-1 endpoint-2

GO subtype: **segment** < endpoint-1 endpoint-2

GO type: **arc** > center

GO subtype: **circle-by-center-radius** < center *radius*

GO subtype: **circle-by-center-point** < center controlpoint

GO subtype: **circle-by-3-points** < endpoint-1 controlpoint endpoint-2

GO subtype: **arc-by-center-radius** < center *radius* *angle-start* *angle-end*

GO subtype: **arc-by-center-point** < center controlpoint *angle-length*

GO subtype: **arc-by-3-points** < endpoint-1 controlpoint endpoint-2

GO type: **elliptic arc** > center endpoint-1 endpoint-2

GO subtype: **elarc-by-center-2-radii-rotation-2-angles** < center *a-radius* *b-radius* *rotation* *angle-start* *angle-length*

GO subtype: **ellipse-by-2-foci-point** < focus-1 focus-2 controlpoint

GO subtype: **ellipse-by-3-points-smallest** controlpoint-1 controlpoint-2 controlpoint-3

GO subtype: **ellipse-by-3-points-rotation-eccentricity** < controlpoint-1 controlpoint-2 controlpoint-3 *rotation* *eccentricity*

GO subtype: **ellipse-by-center-point-rotation-eccentricity <** center controlpoint *rotation* *eccentricity*

GO type: **bezier >** endpoint-1 endpoint-2

GO subtype: **quadratic-bezier <** endpoint-1 controlpoint endpoint-2

GO subtype: **cubic-bezier <** endpoint-1 controlpoint-1 controlpoint-2 endpoint-2

GO type: **text >**

GO subtype: **text <** controlpoint *text* *fontname* *fontsize* (*text* is string, *fontname* is string - font file name, *fontsize* is number in mm)

GO type: **tex_text >**

GO subtype: **tex_text <** controlpoint *text* (*text* is string)

6.2 VRR scheme functions

Follows the list of functions description. On the first line of each description, , there is a 'function prototype' - function name and formal arguments enclosed in parenthesis. On the second line, there is a 'type signature'. For each argument, there is a permitted type (name of type or type signature enclosed in parenthesis in case of functional argument). Character '+' is used to union two type sets. Valid values are scheme type names, vrr type names (proxy, o, obj, go, anchor and hanger), 'any' (means anything), 'false' (#f) and 'specific' (for specific permitted values, which depend on the function). String 'unspecified' is valid only as a return value (in case that return value has no meaning). Characters '[' and ']' are used to specify optional arguments (and their types), '...' can be used to specify that structure of arguments is described in function description. On the following lines, there is a description of functions with links on formal arguments enclosed in '*'.

6.2.1 Functions for typing system

`(proxy? obj [kind [type [st]]])`
any [symbol [symbol [symbol]]] -> boolean
If no additional arguments are given then returns whether *obj* is Vrr proxy. Otherwise also checks given type information - whether obj kind is *kind*, obj type is *type* and obj subtype is *st*.

`(proxy-get-kind proxy)`
proxy -> symbol
Returns the kind of *proxy*.

`(proxy-get-type proxy)`
proxy -> symbol+false
Returns the type of *proxy*.

`(proxy-get-subtype proxy)`
proxy -> symbol+false
Returns the subtype of *proxy*.

6.2.2 Functions for creating and manipulating with GOs

In this section, there are several meta-definition of functions. Sometimes some parts of function names are capital. In that case you can substitute any representative for that category to get valid function. For example make-SUBTYPE means that VRR implements functions like make-circle-by-center-radius or make-quadratic-bezier.

`(make-SUBTYPE ...)`
... -> go
Creates a new GO with subtype SUBTYPE and returns it. Arguments are exactly GO parameters of that subtype (see Section 6.1 [VRR scheme data types], page 12)

`(ANCHOR go)`
go -> anchor
Returns *go*'s anchor named ANCHOR.

`(set-ANCHOR go hanger)`
go hanger -> unspecified
Hangs *go*'s anchor named ANCHOR on *hanger*.

`(aget-ANCHOR go)`
go -> hanger
Returns hanger on which hangs *go*'s anchor named ANCHOR.

`(hget-HANGER go)`
go -> hanger
Returns *go*'s hanger named HANGER.

`(get-named-hanger go name)`
go symbol -> hanger+false
Returns *go*'s hanger named *name* (or false if no such hanger exists).

`(hanger-get-type hanger)`
hanger -> symbol
Returns *hanger*'s name.

`(coords x y)`
real real -> hanger
Creates a free hanger with coordinates (*x*, *y*).

`(coords-c complex)`
complex -> hanger
Create a free hanger with coordinates (real(*complex*), imag(*complex*)).

For some functions, there are aliases (shortened versions):

Function alias	**Function full name**
make-circle/cr	make-circle-by-center-radius
make-circle/cp	make-circle-by-center-point
make-circle/3p	make-circle-by-3-points
make-circle	make-circle-by-center-radius
make-arc/cr2a	make-arc-by-center-radius-2-angles
make-arc/cpa	make-arc-by-center-point-angle
make-arc/3p	make-arc-by-center-radius-2-angles
make-ellipse/c2rr	make-ellipse-by-center-2-radii-rotation
make-ellipse/2fp	make-ellipse-by-2-foci-point
make-ellipse/3ps	make-ellipse-by-3-points-smallest
make-ellipse/3pre	make-ellipse-by-3-points-rotation-eccentricity
make-ellipse/cpres	make-ellipse-by-center-point-rotation-eccentricity
make-elarc/c2rr2a	make-elarc-by-center-2-radii-rotation-2-angles
make-bezier/2	make-quadratiz-bezier
make-bezier/3	make-cubic-bezier

6.2.3 Functions for manipulating with properties

`(lookup name)`
string -> obj+false
Returns obj which has property 'name' set to *name* (or false if no such hanger exists).

`(look tlo name)`
tlo string -> go+false
Returns GO (in tlo *tlo*) which property 'name' is set to *name* (or false if no such hanger exists).

`(property-get o name [missing])`
o symbol [any] -> specific
Returns value of property named *name* in *o* or *missing* if no such property exists or false if *missing* is not set.

`(property-set! o name value)`
o symbol specific -> unspecified
If property *name* exists in *o* then sets its value to *value*, else creates the property with name *name* and value *value*.

`(property-for-each o cb)`
o (symbol specific boolean -> unspecified) -> unspecified
Calls (*cb* name value virtual?) for each property of *o*. Callbacks are called sequentially.

6.2.4 Save and load

`(save-doc doc filename)`
obj string -> unspecified
Saves the document represented by obj *doc* into the file named *filename*

`(load-doc filename)`
string -> obj
Loads the file named *filename* and returns associated obj of subtype document.

7 FAQ

7.1 What is context?

During your work in the program, some objects or windows become implicitly significant from time to time: the object you clicked last, the toplevel window or the current selection. So, when performing an action, you do not have to specify all the subjects explicitly. The collection of these significant things is called The Context. It consists of:

- the current selection
- the current page
- the last clicked object
- the current window, if there is such (usually a view, which determines its page and the page's father document)

The selection has always the highest priority. If the selection is empty, the other objects are used. Almost all actions like keyboard shortcut commands, menu commands etc. operate with the current context. For example, if you press `Ctrl+Z`, which means Undo, the last action is undone in the context page.

Index

Function Index