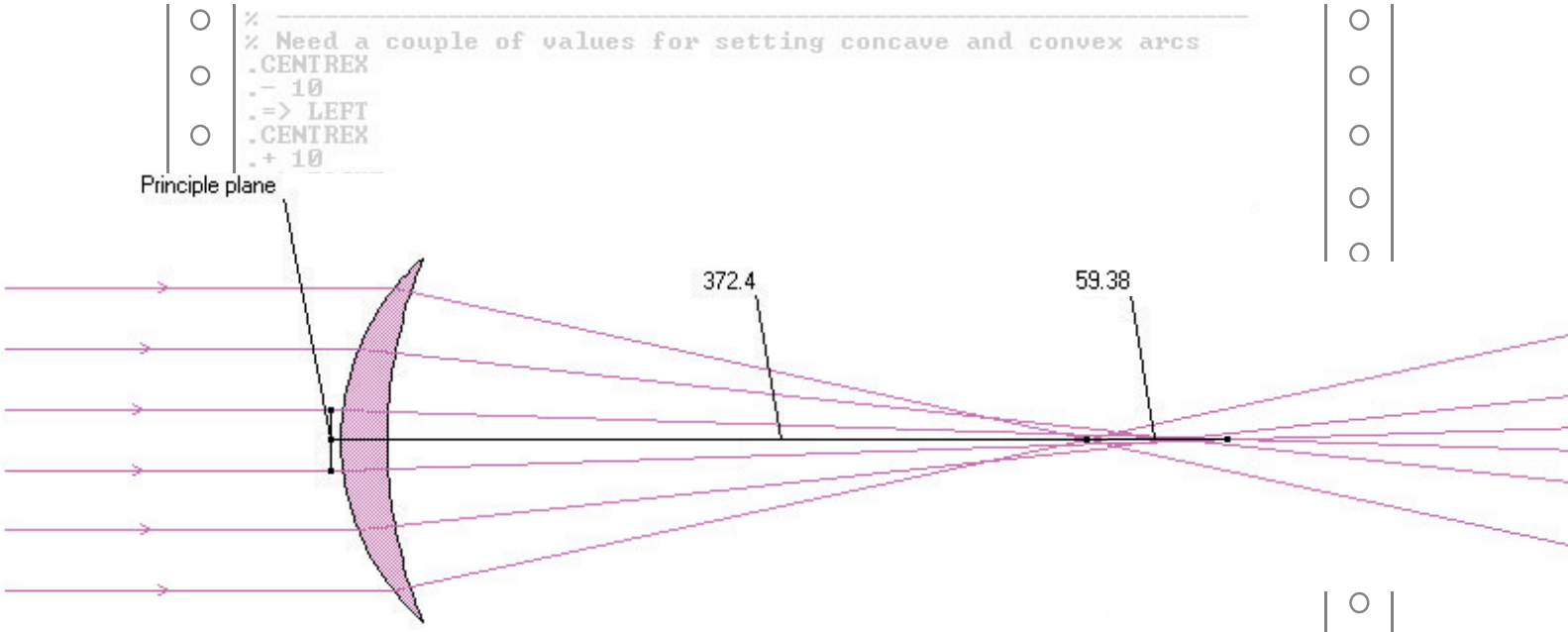
RAYScript

Reference Manual



Ian Moore

IME Software, P.O. Box 1153 Toowong, QLD 4066, Australia

ISBN 0 646 27183 0

Preface

Rayscript refers to the script facility provided as part of the Raytrace package. This facility allows you to program sequences of operations in Raytrace. Using Rayscript you can

- Set up free-running or interactive demonstrations
- Pre-program lecture demonstrations
- Create lessons or tasks for student use
- Program complex sequences of Raytrace operations into "macros"

This manual documents the script language. You should take the time to become reasonably familiar with the normal use of Raytrace before embarking upon writing your own scripts. See the Raytrace Tutorial and Reference manuals for information on using Raytrace.

Raytrace comes with an ever increasing number of example scripts - choosing the menu item **Help -- Helper scripts** will introduce you to most of these.

CONTENTS

INTRODUCTION

Raytrace offers you enormous power and flexibility in designing and working with ray diagrams. The cost of this flexibility is the time required to learn to use Raytrace. Students can ill afford to spend time learning the intricacies of a program that they may only use a few times. Similarly, if you only use Raytrace once or twice a year for lecture demonstrations then you will probably need to refresh your memory each time you use it.

Raytrace's script facility allows you to pre-define demonstrations, tasks or lessons using all of Raytrace's interactive features. Once you have programmed a demonstration as a script it becomes a useful resource that can be used by you and others without recourse to special skills. Another advantage of performing demonstrations with scripts (apart from avoiding embarrassment if you forget how to do something) is that sequences of operations that might distract from the demonstration's main purpose can be done "all at once" within the script.

This manual is divided into three broad sections. The first of these, The script language, presents the basic syntax and operation of the script language and is required reading if you intend to write your own scripts or modify those supplied. After reading this section, try looking at and running some of the supplied script files.

You can run a script either explicitly by selecting the menu item **File -- Run script** and specifying the name of the script file or implicitly by using either of the menu items **Help -- Quick Tour** or **Help -- Helper scripts**. If you have turned on the button bar facility and are using the standard button bar shipped with Raytrace then you can simply click on the RunScript button.

If you have not already done so then try using **Help -- Helper scripts** now to get some idea of what can be done with scripts. Be aware that there are some script files supplied which are not "stand alone" they are meant to be run as subroutines by other scripts - these are clearly identified by comments at the start and some will report errors if you attempt to run them explicitly using **File -- Run script**.

THE SCRIPT LANGUAGE

Syntax

Raytrace scripts are a sequence of commands to Raytrace interspersed with text that is presented to a user in the Script dialog box. Generally, commands mimic a single specific operation within Raytrace. For example, one of the commands that you will use most frequently is "**.click**". This command moves the cursor to specific Raytrace coordinates and performs the same function as if the user had clicked the real mouse button at that position.

Script files are simply ASCII text files. You can use any text editor to create a script file, for example the Windows Notepad editor or the MS-DOS Editor. You can use a word-processor provided you can save your file as plain ASCII text without formatting information.

The syntax of the Raytrace script language is very simple. The first character on a line determines how that line is interpreted.

If the first character is:

- a period, '.', character, then the line contains a command. All commands effectively begin with a period and are shown that way in this manual. There can be only one command per line, e.g

  ```
  .new_text
  ```

- a percent, '%', character, then the line is a comment and is ignored by the script interpreter. Comments serve to provide more information to you or someone else trying to work out how the script works, e.g.

  ```
  % This is a comment
  ```

  Use comments liberally.

- a colon, ':', character, then the line contains a label. The label is the word immediately after the colon. The label is used as a target for transfer of control within the script and performs no direct function, e.g.

  ```
  :LABEL
  ```

- any other character (including the "white space" characters space and tab), then the line contains text which will be displayed in the script dialog box as directions or information for the user, e.g.

  ```
  Click on continue.
  ```

  A blank line in the script is used to force a new line in the text displayed in the script dialog. Hence N blank lines in sequence produces N-1 blank lines in the script dialog.

The script interpreter is totally case insensitive. This applies to commands, variable names and labels.

Strategies for writing scripts

There are two basic strategies you can take in designing a script. The first is to create a ray diagram manually, save it and then have the script open the diagram and perform some operations on it. This approach is often the quickest and simplest and much of the Quick Tour provided with Raytrace uses this method. Once you have created the ray diagram you can use the menu item **Info -- Identify Point** to find the coordinates of points needed by the script.

This method suffers from the disadvantage that if the ray diagram is modified even slightly, then the script may become useless. A second approach is to have the script create the ray diagram from scratch and then operate on it. This requires more effort when initially writing the script but will probably pay off in the long run. Script files also tend to be much smaller than any ray diagram they create so you save on disk space; not that this is so much of an issue these days.

Before writing your first few scripts using this strategy, create the basic ray diagram manually to get an idea of what coordinates you will use and the sequence of operations involved; note these down and you are well on the way to developing your script.

Develop your scripts a small section at a time. Scripts that accomplish a complex task such as a lesson or in depth demonstration may require several hundred lines. While developing such a script it is a good idea to break it into shorter segments which are linked in sequence using **.script** commands as illustrated in the three example scripts shown below.

In a file called part1.rsc:

```
% This is the first in a series of scripts.
% In this part the ray diagram is cleared and a prism created
.file_new
.clear_var
.create_region
.fc -80 0
.fc 80 0
.fc 0 100
.end_element
.=> PRISM
% Save the diagram before jumping to the next part. This
% allows part 2 to run independently if necessary by
% opening this file. Use .save/d so that all handles etc
% are guaranteed to be valid on reloading.
.save/d temp1.ray
% Jump to part 2
.script part2.rsc
```

In a file called part2.rsc:

```
% Part 2 in this series creates a white ray striking the prism
% Get the state saved at the end of part 1
.new_text
Now running part 2
.pause
.open temp1.ray
% Allow stepping back (provided part1 can re-initialise itself)
.run_from part1.rsc
.source_ray_count 3
.point_source
.fc -160 -47
.mid_snap
.object_coordinates PRISM 3 middle
.fc X Y
.mid_snap
.fc X Y
% Make the source a white ray
.all_rays
.white_ray
.unselect_all
% Save state for part 3
.save/d temp2.ray
```

```
.script part3.rsc Jump to the next part
```

In a file called part3.rsc:

```
% Part 3 in this series of scripts adds a normal
% Get state saved at end of part 2
.new_text
Now running part 3
.pause
.open temp2.ray
.run_from part2.rsc
% Put a normal on the first incident ray
.all_rays
.selected_ray
.=> RAY
.unselect_all
.set_length 50
.select_object RAY
.normal 1
% Save etc. if you wish to add more parts
```

And you can continue with many parts to make up a script as long as you want. This sequence allows you to step forward and backward through the series of scripts or to start midway through after an error. When you have finished developing and debugging you can, if you wish, combine all the parts into one big script file and delete all the temporary ray files.

It is generally a good idea to use **.save/d** instead of **.save** in this situation to ensure that the re-opened file reproduces an identical internal structure for the ray diagram database.

Prototypes

Base a script which creates its own ray diagram on the following prototype:

```
% A comment describing what the script does
.new_text
This script does .... describe it again. This will appear
to the user.

Do you wish to proceed?
% Give user a chance to stop before launching a long demo/task
.yes_no SKIP
.file_new
.clear_var
.enable_undo 0
.quiet_snap 1
.grid_snap 0
%
% The body of the script which does what you intend
%
:SKIP
.enable_undo 1
.quiet_snap 0
.return
```

The first part of the script tells the user what is about to happen and gives them a chance to get out before becoming involved in something they may have started by accident.

Setting the state of **.enable_undo** and **.quiet_snap** at the start is a good habit since you cannot predict their state when the script is run. Setting their state before exiting is a protocol that might help other scripts which have neglected to set the states survive. Do the same with any other options that you may decide to change - for example **.drag_handles**.

You can end a script with **.close_script** but using **.return** is functionally the same in that it will close the script dialog when the user has run the script explicitly. Using **.return** is more flexible since it allows another script to run this script - like the Helper Scripts.

Base a script which is intended only for use as a sub-routine on this prototype:

```
% Describe the sub routine's function (maybe several lines)
% Expects:
%    List here any variables that the routine expects as "arguments"
%    and how they are used
% Creates:
%    List here any variables the routine creates as "return values"
%    and how they should be interpreted
% Clears:
%    List here any variables that are cleared by the routine.
% Describe any other important information for a caller.
%
% The body of the script routine
%
.return
```

The "Expects:, Creates:, Clears:" pre-amble provides essential information needed by a caller. A caller should avoid using any variable names which appear in the "Clears" section so as not to lose information.

If you want to give users an easy path into a series of scripts then create a script along the following lines.

```
% This script gives users easy access to other scripts
:REPEAT
% Give the user some options
.new_text
```

```
Click on...

A - to run "good1"; a script on...

B - to run "good2"; a script on...

C - to see more options

D - to exit this script.
.choose 4 A B C END
:A
.call good1.rsc
.goto REPEAT
:B
.call good2.rsc
.goto REPEAT
:C
% Another set of options
.new_text
Click on...

A - to run "good3" a script on...

B - to go back to the previous options

C - to exit this script.
.choose 3 A2 REPEAT END
:A2
.call good3.rsc
.goto REPEAT
:END
.return
```

This allows the user to progress through a series of choices to locate the script they want without having to search through a list of filenames. The script file helper.rsc supplied with Raytrace is written along these lines - it is always run in response to the user selecting the menu item **Help -- Helper scripts**. If you like, you can replace the suppled helper.rsc file with your own version with the same name to customise access to your own scripts.

The script calculator

The script facility provides the basic functionality of a scientific calculator. At this stage the script interpreter cannot parse complicated arithmetic expressions; these must be evaluated by a series of commands not unlike the sequence of keystrokes you would use when using a simple calculator. The script calculator is for the occasional necessary calculation not for solving complex arithmetic problems and the interpreter was designed to be as simple as possible.

Central to using the script calculator is an understanding of the "accumulator". You can think of the accumulator as a variable which holds the number that would be displayed on a simple calculator. Binary operators like + and * use the accumulator as one of their arguments. Results of calculations are placed in the accumulator. You can enter numbers into the accumulator as if they were commands, e.g.

```
.1.23
```

places 1.23 in the accumulator. Note the initial period is not part of the number, a digit before a decimal point is not necessary but is recommended for clarity.

You can used named variables. These are defined simply by the action of storing a number in them. For example the sequence

```
.0.25
.=> QUARTER
```

first enters 0.25 into the accumulator then stores the accumulator in a variable called QUARTER. You can then use the variable name QUARTER as if it was a number. For example,

```
.0.25
.=> QUARTER
.QUARTER
.+ QUARTER
.=> HALF
```

Results in a value of 0.5 in both the accumulator and another variable called HALF.

If a variable name is followed by a numerical value then the value is stored in the variable. The accumulator is unaffected by this. For example,

```
.0
.=> X
.=> Y
.=> COUNT
.COUNT 20
```

defines three variables, X, Y and COUNT. X, Y and the accumulator contain 0 and COUNT contains 20.

There is a limit of 50 named variables at any one time. You should clear unwanted variables with the **.clear_var** command to avoid running into this limit.

There is no precedence in the script calculator, each command is processed immediately it is encountered. For example:

```
.3
.+ 2
.* 5
```

results in a value of 25 in the accumulator not 13. Using a temporary variable is often one way of avoid precedence problems. For example, one way to evaluate the expression: $(3x^2 + 10x + 8)^{-1}$ would be the following sequence (assuming the variable x is defined):

```
% Eval 1/(3x^2+10x+8)     Comment is a good idea unless the expression is trivial
.^2 X                     Squares X and place result in the accumulator
.* 3                      Multiply the accumulator by 3
.=> TEMP                  Save the result in a variable called TEMP
.X                        Get contents of x into accumulator
```

```
.* 10                   Multiply accumulator by 10
.+ TEMP                 Add the previous result
.+ 8                    Add 8 to the accumulator
.1/                     Take the reciprocal of the accumulator
.clear_var TEMP         Be tidy and remove the temporary variable
```

Some thought will often simplify the resulting script. For example the same expression could be evaluated without using a temporary variable as shown below

```
% Eval 1/(3x^2+10x+8)
.X                      Get contents of x into accumulator
.* 3                    Multiply by 3
.+ 10                   Add 10
.* X                    Multiply the sum by x
.+ 8                    Add 8 to the accumulator
.1/                     Take the reciprocal of the accumulator
```

There are four keywords which you can use instead of numeric strings These are:

userinput This displays an edit box in the script dialog and expects the user of to enter the number which will be used.

Example:

```
% Get coords from user
.userinput
.=> X
.=> Y
.Y userinput
```

Would get two numbers from the user and place them in the variables X and Y.

See also the commands **.default_input** and **.input_limits**.

random This generates a random number (uniformly distributed) between 0 and 1 which will be used in its place. You might use random numbers to generate variations between ray diagrams for individual students.

Example:

```
% Vary focal length between 150 and 250
.100
.* random
.round
.+ 150
.=> FLEN
```

See also the command **.randomize**.

lastclick.x These replace themselves with the x and y coordinates (respectively) of the most **lastclick.y** recent mouse click (user and script generated clicks included).

Example:

```
.user_click
.new_text
You click at coordinates:
.printf %g lastclick.x
,
.printf %g lastclick.y

% Displays coordinates of user's click
```

Note that the initial period of the command line is not part of these keywords.

And finally...

As you become familiar with the script language you may find yourself asking the questions, "Why don't commands like, for example, **.point_source** take coordinates as arguments? Why do these coordinates have to be specified using subsequent commands like **.click**?" The answer is that this method allows control to be passed seamlessly from the script to the user and back again without any change in syntax.

SCRIPT COMMANDS BY FUNCTION TYPE

Arithmetic and calculator related commands	
.abs	absolute value
.sign	find sign -1,0,1
.arccos, .arcsin, .arctan, .arctan2	inverse trig functions
.clear_var	clear script variable(s)
.=>	assign to variable
.cos, .sin, .tan	trig functions
.deg>rad, .rad>deg	degree <> radian conversions
.exp, .log_e, .^10, .log_10	exponentials and logarithms
.max, .min	min and max functions
.randomize	re-seed random numbers
.round, .truncate	convert to integer
.sqrt, .^2	square root, square
.+, .-, ,*, ./	basic 4 functions
.1/	reciprocal
.+/-	change sign

Clear or delete commands	
.clear_all	clears entire ray diagram
.clear_rays, .clear_elements, .clear_tapemeasures, .clear_protractors, .clear_annotations, .clear_trails	clear all of one type of object
.delete	delete selected objects

Element modification commands	
.change_conic	change constraints on existing conic segment
.element_on, .element_off	turn ray interactions with selected elements on and off
.explode, .join	break apart and join together selected elements
.group, .ungroup	make/unmake element groupings
.link_segments, unlink_segments	make/unmake links between segments
.link_move, .link_stretch	options for .link_segments
.lock, .lock_x, .lock_y, .unlock	constrain how selected elements may be dragged
.make_region, .make_shape, .make_surface	change selected elements to specified type
.set_radius	when dragging arc tangent point, specify numerical radius

Tables of commands by category

Editing commands	
.cut, .copy, .paste	clipboard type functions
.delete	delete selected objects
.mirror	flip selected objects about a line
.move_selection	move selected objects
.rotate, .rotate_drag	rotate selected objects
.scale	resize selected objects
.undo	undo previous operation

Execution flow control	
.call, .return	call into "subroutine" script and return from it
.choose	give user choices to follow
.goto	unconditional execution branch
.if==, .if!=, .if<=, .if>=, .if<, .if>	conditional execution branching
.run_from, .script	set function of "Previous" button, jump to script
.yes_no	give user yes/no choice
.--if!=0	decrement variable and branch if non-zero

File commands	
.file_new	clear diagram and reset some options
.open	open existing diagram
.save, .save/d	save diagram to file

Miscellaneous commands	
.close_script	exit script dialog
.control_key, .shift_key	set state of control and shift keys
.exit_raytrace	close raytrace application
.new_text	start new text in script dialog
.pause	wait for user to click continue
.printf	convert number to text in script dialog
.button_bar	control display of button bar
.auto_drag	start an auto drag operation

Mouse related commands	
.click, .click_rel, .ck, ckr	move cursor and "click"
.fast_click, .fast_click_rel, .fc, .fcr	"click" without cursor movement
.fast_move, .fast_move_rel	reposition cursor
.mouse_limits	set limits on user mouse input
.move, .move_rel	move cursor on screen

Ray and element creation and related commands	
.anticlockwise_arc, clockwise_arc	set arc direction for radius/end, centre/end options
.arcseg, .conicseg, .lineseg	change element segment type
.centre_end, .chord_tan, .radius_end	change arc creation option
.conic_options	set default conic segment constraints
.create_ray, .create_rays	create a ray or rays
.finished_rays	end ray creation
.create_region, .create_shape, .create_surface, .create_iris, .create_grating	create element of specified type
.end_element	complete creation of an element

Option control	
.drag_handles	control display of drag handles while script executing
.drag_single	set option when dragging a source
.enable_undo	control saving of undo information
.grid, .grid_size	control display and size of grid
.max_path_depth	set maximum number of child rays
.painting	control refresh of raytrace display
.recalc	control recalculation of ray paths
.quiet_snap	control use of sound when snapping
.update_on_element_drag, .update_on_ray_drag	control recalculation of rays during dragging

Protractor/Tapemeasure/Annotation and Trail commands	
.annotation	create an annotation
.protractor	create a protractor
.protractor_options	set options for selected protractors
.tapemeasure	create a tapemeasure
.pause_trails	suspend updating of trail objects
.reset_trails	wipe all trail info, start again
.trail	create a trail
.tapemeasure_options	set options for selected tapemeasures

Query diagram	
.child_by	find handle of a child of a ray
.object_coordinates	find coordinates related to an object
.object_type	find object type from handle
.ray_of_source	find handle of particular ray from source
.segment_count	count segments in an element
.selected_annotation, .selected_element, .selected_protractor, .selected_ray, .selected_tapemeasure, .selected_trail	find handle of a selected object

Ray property commands	
.arrow_end, .arrow_middle, .arrow_none	set arrow position of selected rays or default position
.back_project, .forward_project, .normal, .parent_refract, .parent_reflect, .refract, .reflect, .reflect_if_no_refract	set fertility of selected rays or default fertility
.blue, .green, .red	set colour of selected rays or default colour
.childray_arrow, .childray_fertility	copy default properties for user created rays into defaults for child rays
.cycle_colour	change colours of selected rays to next in sequence R>G>B>R
.set_length	set default length for selected rays of default for created rays
.ray_diffraction	set the diffraction orders at grating interactions
.white_ray	set colours of rays in sequences of 3 of RGB

Refractive index and wavelength commands	
.material	set refractive index properties by material
.refractive_index	set refractive index properties numerically
.blue_index, .green_index, .red_index	get current refractive index values or values for selected element
.additive, .subtractive	sets selected regions to add or subtract to refractive index sum for overlapping regions
.background, .background_material	set the background refractive index values
.thin_lenses_with_background	controls if thin lenses are modified when background refractive index is changed
.wavelengths	set wavelengths of RGB

Selection commands	
.all_rays, .all_elements, .all_tapemeasures, .all_protractors .all_annotations, .all trails	select all objects of specified type
.select_child_rays	select all child rays of selected rays
.select_extended	begin extended selection operation
.select_group	select all elements in same group as selected elements
.select_object, unselect_object	select/unselect an object by handle
.unselect_all	unselect any selected objects
.unselect_annotations, .unselect_elements, .unselect_protractors, .unselect_rays, .unselect_tapemeasures, .unselect_trails	unselect any selected objects of specified type

Show/Hide types of object	
.blue_rays, .green_rays, .red_rays	control display of rays by colour
.show_annotations, .show_protractors, .show_tapemeasures, .show_trails	control display of specified objects

Snap commands	
.centre_snap, .end_snap, .focus_snap, .grid_snap, .intersect_snap, .mid_snap, .perp_snap, .tangent_snap, .vertex_snap	set a snap type for next mouse click
.no_snap	remove any requested snap type

Source commands	
.auto_trace, .step, .more_steps?, .steps_per	begin/control tracing of a source around an element
.link_by_child_rays	link a point source by its child rays
.plane_source, .point_source	create a plane/point source
.source_ray_count	set number of rays in selected or new sources
.flip_source	reverse aperture points for point source

Thin lens and paraxial mirror related commands	
.converging_lens, .diverging_lens	create a thin lens
.make_converging, .make_diverging	change property of a thin lens
.paraxial_mirror	create a paraxial spherical mirror
.paraxials_on_centre	set option for style of creation

User input and control commands	
.allow_drag	let user continue but not complete a drag operation
.choose	let user choose from a set of options
.default_input	set default value of next numeric input by user
.input_limits	set limits on next numeric value input by user
.mouse_limits	limit region in which user can move cursor
.user_click	let user click the mouse button
.user_control	give user complete control
.yes_no	give user a yes/no option

Window and display control	
.pan_left, .pan_right, .scroll_up, .scroll_down	move window around over ray diagram
.zoom_extents, .zoom_in, .zoom_out, .zoom_previous, .zoom_reset	change scale and position of window over ray diagram

SCRIPT COMMANDS IN ALPHABETIC ORDER

Notation used in command descriptions

Commands descriptions are presented in the following format:

---

[1].name

[2]**.name** [3]argument

[4]A short description of the operation performed.

Example:

```
[5]% Example script command sequence
[6]>     .name
```

[7]**See also**: .other_commands

Notes:

[1] The name of the command in bold. Sometimes groups of commands are described together; in this case the commands are separated by commas.

[2] If the command takes arguments then the syntax is given next. If it takes no arguments then this line is omitted. If more than one command is being described in the same section then a syntax line like this is used to delimit information on each command regardless of whether arguments are taken or not.

[3] Arguments appear in italics. Arguments which may be omitted are enclosed in square brackets. Sometimes numeric arguments are shown enclosed in square brackets when the command obviously must operate upon a number, see for example the description of **.abs**. Arguments indicated in this way will be replaced by the contents of the accumulator.

[4] A short description of the function performed.

[5] Generally a short example is given. Almost all examples given can be executed directly without any further additions - there are a couple of exceptions. Blank lines in examples are there intentionally as they perform the function of forcing text displayed in the script dialog to start a new line.

[6] Lines in the example which contain the command being described are indicated by a '>' character at the left. The '>' character is not part of the script.

[7] Other commands which are related or may perform similar functions are shown last.

.abs, .sign

.abs [*real*]

Places the absolute value of the *real* in the accumulator.

.sign [*real*]

If *real* is negative then places -1 in the accumulator.
If *real* is positive then places 1 in the accumulator.
If *real* is zero then places 0 in the accumulator.

.additive, .subtractive

Change how all selected region elements contribute to the summation of refractive index values when regions overlap. Same as the menu items **Modify -- Element > Additive refractive index** and **Modify -- Element > Subtractive refractive index** respectively. See the Raytrace reference manual for more information.

.allow_drag

Allows the user to control the current drag operation. The dragging must be initiated and terminated by the script. User control of dragging terminates when the primary mouse button is clicked.

It is recommended that some direction should be given to the user about returning control to the script and that the **.mouse_limits** command be used to set the region in which the user can drag the point about.

Example:

```
      % Create a ray for the user to drag about
      .file_new
      .create_ray
      .fast_click 200 200
      .fast_click 250 250
      .=> RAY
      .finished_rays
      % Give some guidance to the user
      .new_text
      You can now drag the end of the ray about. Click on the
      primary mouse button to return to the script.
      % Set limits on user dragging
      .mouse_limits 225 150 275 250
      % Select the ray and initiate dragging
      .unselect_all
      .select_object RAY
      .fast_click 250 250
>     .allow_drag
      % Return the ray to the original position and terminate drag
      .fast_click 200 200
```

See also: .mouse_limits, .user_click and .user_control

.all_rays, .all_elements, .all_tapemeasures, .all_protractors, .all_annotations, .all_trails

Each of these commands selects all objects of the specified type within the ray diagram.

See also: .unselect_rays

.annotation

.annotation *text*

Creates an annotation with the given text. Must be followed by appropriate click type commands to specify the base and leader points.

Example:

```
      % Create an arc reflector for this example
      .file_new
      .create_surface
      .click 0 -50
      .arcseg
      .chord_tan
      .click 0 50
      .click 25 50
      .end_element
      % Create an annotation on the centre of the arc
>     .annotation Centre
      .centre_snap
      .click 0 -50
      .click_rel -20 30
```

.anticlockwise_arc, .clockwise_arc

When creating an element and using arc segments with either the centre/end radius/end options then these commands allow you to set the direction in which the arc is drawn. Under manual operation this option is set using the secondary mouse button and choosing the arc segment sub menu.

Example:

```
      % Create a meniscus shaped lens with arc centres specified
      .file_new
      .create_region
      .click 0 -50
      .arcseg
      .centre_end
>     .anticlockwise_arc
      % Centre point at (-200,0) chord is from (0,-50) to (0,50)
      .click -200 0
      .click 0 50
>     .clockwise_arc
      % Next arc centre at (-400,0) same chord as first arc
      .click -400 0
      .click 0 -50
      .end_element
```

See also: .arcseg and .centre_end

.arccos, .arcsin, .arctan, .arctan2

.arccos [*real*]
.arcsin [*real*]
.arctan [*real*]

Place the inverse trig function the *real* in the accumulator.

.arctan2 [*real_x*] [*real_*y]

Find the arc tangent of real_y/real_x - in four quadrants.

Example:

```
      % Find the arcsin of a constant value
>     .arcsin 0.43
      .printf %g

      % Find the arctan of a value in a variable
      .0.43
      .=> X
      .printf %g X
      ,
>     .arctan X
      .printf %g

      % Find an arctan in four quadrants
>     .arctan2 -1 1
      .rad>deg
      .printf %g
```

```
% End of example script
```

.arcseg, .conicseg, .lineseg

These commands are used to switch between the different segment types when creating an element. Under manual control this is done using the secondary mouse button and choosing the segment type from the popup menu.

Example:

```
      % Start creating a region
      .file_new
      .create_region
      % First segment is a line from (100,100) to (200,100)
      .click 100 100
      .click 200 100
      % Now switch to arc segments using the chord/tangent method
>     .arcseg
      .chord_tan
      % Arc end point at (200,200)
      .click 200 200
      % Arc tangent point at (250,200)
      .click 250 200
      % Return to line segments
>     .lineseg
      .click 100 200
      % And finish the element
      .end_element
```

See also: .create_region, .centre_end

.arrow_end, .arrow_middle, .arrow_none

Set the arrow position of any selected rays. If no rays are selected then the default arrow position for new rays is set.

See also: childray_arrow

.auto_drag

.auto_drag *element delay*

*element*either a valid element handle or -1, see below

delay 0 Moves the point without delay
1 Pauses at each step

Start an auto drag function. The same as clicking on the secondary mouse button and choosing Auto drag when dragging some point of the ray diagram.

If the first arguement is a valid element handle then the autodrag function will start immediately using that element as the path. If however the first arguement is not a valid element handle (preferably use -1) then the drag path element must be specified by a mouse click operation which should follow immediately.

If *delay* is zero then the dragging proceeds without delay. If it is non-zero then the stepping must be controlled by the script using the **.step** and **.more_steps?** commands.

Example:

```
      .file_new
      .create_shape
      .fc 0 0
      .fc 100 100
      .end_element
      .=> SHAPE
      .create_ray
      .fc 10 0
      .fc 30 0
      .=> RAY
      .finished_rays
      .select_object RAY
      .object_coordinates RAY middle
      .fc X Y
```

```
      .move 0 0
>     .auto_drag SHAPE 1
      :LOOP
      .step
      .pause 300
      .more_steps? LOOP
      .click 20 0
```

See also: .auto_trace, .step, .more_steps ans .steps_per

.auto_trace, .step, .more_steps?, .steps_per

.auto_trace *option [delay]*

option	1	Move the source centre (for a point source) or base point (for a plane source)
	2	Move the first aperture point
	3	Move the second aperture point
delay	0	Moves point without delay
	other	Pauses at each step

The **.auto_trace** command performs the same function as the menu item **Modify -- Auto trace**. A source and an element (preferably a shape element) must be selected; one of the source points (usually the centre or base point *option* = 1) will be stepped around the selected element. The *option* argument specifies which point of the source is moved during the trace. If *delay* is zero then the tracing proceeds as rapidly as possible. If delay is non-zero then the **.step** and **.more_steps?** commands must be used; see below.

.step

If the *delay* argument of the **.auto_trace** command is non-zero then the tracing will proceed one step at a time and the **.step** command must be used to perform each step. If the *delay* argument is omitted then the accumulator contents are used.

.more_steps? *label*

If the *delay* argument of the **.auto_trace** command is non-zero then **.more_steps?** can be used to control a loop containing a **.step**. If there are more steps to be performed in the tracing operation the **.more_steps?** will branch to the *label*. If the tracing operation is complete then execution continues on the next line.

.steps_per *line arc conic*

Sets the number of steps that will be taken along each type of segment during an **.auto_trace** function. The same as using the **Defaults -- Auto drag settings** menu item to set these values.

Example:

```
      % Assumes that an element and source have been selected
      % Tracing proceeds without delay
>     .steps_per 5 10 10
>     .auto_trace 1 0
      % Tracing proceeds in steps with a 500 ms delay between steps
>     .auto_trace 1 1
      :LOOP
      .pause 500
>     .step
>     .more_steps? LOOP
```

See also: .drag_handles

.background, .background_material

.background *R G B*

Set the background refractive index values to R, G abd B for the red, green and blue wavelengths.

.background_material *Name*

Select refractive indices of a material called *Name* from the raytrace.mat file. There must be a single space between the end of **.background_material** and the start of *Name*. *Name* may contain spaces and is taken as everything up to the end of the line. See the reference manual for information on background refractive indices.

.back_project, .forward_project, .normal, .parent_refract, .parent_reflect, .refract, .reflect, .reflect_if_no_refract

.back_project [*flag*]
.forward_project [*flag*]
.normal [*flag*]
.parent_refract [*flag*]
.parent_reflect [*flag*]
.refract [*flag*]
.reflect [*flag*]
.reflect_if_no_refract [*flag*]

Set or toggle the fertility property of any selected rays. If no rays are selected then operates on the default fertility for new rays.

If *flag* is zero then the specified fertility is cleared.
If *flag* is non-zero then the specified fertility is set.
If *flag* is omitted then the current fertility state is toggled.

Example:

```
      .create_ray
      .fast_click 0 0
      .fast_click 100 100
      .=> RAY
      .finished_rays
      .unselect_all
      % Select ray, make it always reflect but never refract
      .select_object RAY
>     .reflect 1
>     .refract 0
      .unselect_all
      % Turn on normals for all rays created in future
>     .normal 1
```

See also: .childray_fertility

.blue, .green, .red

Sets the colour of any selected rays. If no rays are selected then sets the default colour for newly created rays.

See **.create_ray** for an example.

.blue_index, .green_index, .red_index

If no elements are selected then places the value of the default refractive index for the specified colour into the accumulator. If an element is selected the places the refractive index for that element and colour in the accumulator.

.blue_rays, .green_rays, .red_rays

.blue_rays [*flag*]
.green_rays [*flag*]
.red_rays [*flag*]

Control whether rays of specified colour a visible or not.

If *flag* is zero then rays of specified colour are hidden.
If *flag* is non-zero then rays of specified colour are shown.
If *flag* is omitted then visibility is toggled.

.button_bar

.button_bar *state* [*filename*]

state	0	Turns the button bar display off
	1	Turns on a horizontal button bar
	2	Turns on a vertical button bar
filename		If given Raytrace searches the work directory followed by the system file directory for *filename* and attempts to load it as a button bar.

.call, .return

.call *filename*

Transfers execution to the start of the script file *filename.* Execution returns to the current script when a **.return** statement is encountered.

If the maximum nesting depth for script routines is exceeded then the error message "Script stack overflow" will be reported.

There is no facility to explicitly pass argument to called scripts. However values may be "passed" in the global variables. This requires some discipline in using variable names and some comments at the start of the script indicating the variables that are expected, any that are created and any that are cleared is usually a good idea.

.return

If the script has been called from another script then execution returns to the line following the .call statement. If the script was run by the user then it closes the script dialog box.

Example:

Calling script

```
      % Script calls a sub function script to create a polygon
      .file_new
      .5
      .=> POLYN
      .50
      .=> POLYRAD
      .0
      .=> POLYCENX
      .=> POLYCENY
>     .call polygon.rsc
      .select_object POLY
      .make_region
```

Called script in file "polygon.rsc"

```
      % Script draws a closed polygon shape element
      % Expects:
      %   POLYN = number of sides
      %   POLYCENX = centre x coord
      %   POLYCENY = centre y coord
      %   POLYRAD = polygon inscribed in circle of radius
      % Creates:
      %   POLY = handle of polygon element
      % Clears:
      %   POLYCENX, POLYCENY, POLYANG, POLYINC
      % On return POLYN will contain 0.
      .POLYN
      .if< 3 NOGO
      .360
      ./ POLYN
      .DEG>RAD
      .=> POLYINC
      .POLYN
      .+ 1
      .=> POLYN
      .0
      .=> POLYANG
      .create_shape
      :MORESIDES
      .cos POLYANG
      .* POLYRAD
      .+ POLYCENX
```

```
        .=> POLYX
        .sin POLYANG
        .* POLYRAD
        .+ POLYCENY
        .=> POLYY
        .fc POLYX POLYY
        .POLYANG
        .+ POLYINC
        .=> POLYANG
        .--if!=0 POLYN MORESIDES
        .end_element
        .=> POLY
        .clear_var POLYCENX POLYCENY POLYANG POLYINC
        :NOGO
>       .return
```

See also: .close_script, .script

.centre_end, .chord_tan, .radius_end

.centre_end Set the method of specifying an arc segment to Centre/End
.chord_tan Set the method of specifying an arc segment to End/Tangent
.radius_end Set the method of specifying an arc segment to Radius/End

Note that this setting is "sticky" it persists from one element creation to the next so don't assume it is any particular state.

Example:

```
        % Create surface of three arcs spec'd in different ways
        .create_surface
        .click 0 0
        % First arc has chord from (0,0) to (0,100)
        .arcseg
>       .chord_tan
        .click 0 100
        % Tangent such as to make a semi-circle
        .click 50 0
        % Second arc has centre at (-50,0) and ends at (-100,100)
>       .centre_end
        .anticlockwise_arc
        .click -50 0
        .click -100 100
        % Third arc has radius specified as 300 and ends at (-100,0)
>       .radius_end
        .clockwise_arc
        .click_rel 300 0
        .click -100 0
        .end_element
```

See also: .arcseg

.centre_snap, .end_snap, .focus_snap, .grid_snap, .intersect_snap, .mid_snap, .no_snap, .perp_snap, .tangent_snap, .terminal_snap, .vertex_snap

Specify that the next point will be a snap. Behaves in the same manner as under manual control - that it the cursor must be placed over the object that is to be snapped to. You can use any of **.click**, **.click_rel**, **.fast_click** or **.fast_click_rel** commands to position the cursor for the snap.

.grid_snap *flag*

If *flag* is zero then turns of the grid snap
If *flag* is non-zero then turns on the grid snap
If *flag* is omitted then toggles the state of the grid snap

.no_snap may be used to ensure that the user has not left a snap in force when the script starts running or after using a **.user_control** command.

Examples:

You can often find appropriate coordinates at which to place the cursor using the **.object_coordinates** command .

```
      % Create an arc for this example
      .create_surface
      .fast_click 0 0
      .arcseg
      .chord_tan
      .fast_click 0 100
      .fast_click 50 75
      .end_surface
      .=> ARC
      % Find mid point of arc - coords put in X,Y
      .object_coordinates ARC 1 middle
      % Snap a tapemeasure between the centre and mid points
      .tapemeasure
>     .centre_snap
      .click X Y
>     .mid_snap
      .click X Y
```

In the case of an intersect snap you may want to select the two rays by means other that "clicking" you can do this using a sequence along the following lines:

```
      % Select the two rays by some means like this
      .select_object RAY1 RAY2
      % Specify the snap
>     .intersect_snap
      % "Press" the control key
      .control_key 1
      % Click somewhere out in no-mans land where there are no rays
      .fast_click -100000 -100000
      % "Release" the control key
      .control_key 0
```

See also: .object_coordinates

.change_conic

.change_conic *element segment*

*element*object handle of existing element

segment segment number of the segment to be modified

Changes the conic settings for the specified segment of the specified element to those currently set as the default values (see .conic_options). The segment number is zero offset; that is the first segment in the element is numbered 0.

Three self explanatory errors may be reported:
Specified object is not an element
Specified segment number does not exist
Specified segment is not a conic

Example:

```
      % Create surface with a conic, pause then change conic setting
      .conic_options focus_end_axis
      .create_surface
      .fast_click 100 100
      .fast_click 0 100
      .conicseg
      .fast_click -20 50
      .fast_click -50 50
      .fast_click 0 0
      .lineseg
      .fast_click 100 0
      .end_element
      .=> SURFACE
```

```
      % Wait until Continue is pressed so that initial shape is
      visible
      .pause
      % Make the conic a parabola
      .conic_options vertex_end_eccent 1
>     .change_conic SURFACE 1
```

See also: .conic_options

.childray_arrow

Copies the current default arrow position for user created rays to the default for child rays. Use in a similar fashion to **.childray_fertility**.

Use in the same manner as childray_fertility.

See also: .childray_fertility and .arrow_end

.childray_fertility

Copies the current default ray fertility for user created rays to the default for child rays. Hint: Set the child ray fertility first so that you don't have to reset the default fertility.

Example:

```
      % To set all child rays to reflect always
      % Set the default fertility for created rays
      .reflect 1
      % Then copy it to the childray fertility
>     .childray_fertility
      % Then set the real desired default fertility for created rays
      .reflect_if_no_refract 1
```

See also: .childray_arrow and .back_project

.child_by

.child_by *ray option*1 [option2] ... [optionN]

ray is the object handle of a ray

options are one of:
reflection
refraction
normal
parent
forward_proj
back_proj
parent_refraction
parent_reflection
diffraction N

Finds the child of the specified ray that is descended by the specified set of fertility options. Places the object handle of this child ray in the accumulator. If the specified descendent ray does not exist then places -1 in the accumulator. If the diffraction option is used then the order of the diffracted ray must be specified too.

Example:

```
      % Create a slab prism
      .file_new
      .create_region
      .click -100 -50
      .click 100 -50
      .click 100 50
      .click -100 50
      .end_element
      .set_length 50
      % Hit it with a ray
      .create_rays
      .click -25 100
      .click 0 50
```

```
        .=> INCIDENT
        .finished_rays
        % Find the handle of the ray within the prism
>       .child_by INCIDENT refraction
        .=> REFRACTED
        % And reflect it
        .select_object REFRACTED
        .reflect
        % Find the ray coming back out the top of the prism
>       .child_by INCIDENT refraction reflection refraction
        .=> FINAL
        % And set its arrow position to the end
        .unselect_all
        .select_object  FINAL
        .arrow_end
        .unselect_all
```

.choose

.choose *N label1 ... labelN*

N An integer between 1 and 10 inclusive

labeln Script label

Presents the user with *N* buttons labelled A,B,C etc. When the user clicks on one of these buttons execution is transferred to the corresponding label within the script.

Example:

```
      % Give the user some prompting information
      .new_text
      Choose on of the possible options:

      A -- Possibility 1

      B -- Possibility 2

      C -- Possibility 3

      D -- Skip all
      % Display the choice buttons
>     .choose 4 A B C D
      :A
      .new_text
      You selected possibility 1.
      .goto D
      :B
      .new_text
      You selected possibility 2.
      .goto D
      :C
      .new_text
      You selected possibility 3.
      :D
      % Rest of script follows
```

See also: .if==, .goto and .yes_no

.ck, .ckr

See **.click**

.clear_all

Same function as choosing the menu item Clear -- All. It clears everything from the ray diagram without changing zoom or other settings.

See also: .clear_rays, .delete and .file_new

.clear_rays, .clear_elements, .clear_tapemeasures, .clear_protractors, .clear_annotations, .clear_trails

Same functions as available in the Clear menu. Clears all objects of specified type from the ray diagram.

See also: .clear_all and .delete

.clear_var

.clear_var [*name*] ... [*nameN*]

name is the name of a script variable

Clears a variable or variables so that they no longer exist. If no argument is given then all variables are cleared. It is a good idea when starting a new script (that is not going to be called

by other scripts) with a **.clear_var** command so that your variable space is not cluttered up. There is a limit of 50 variables so clearing unused ones make sense.

Example:

```
      % Clear all variables
      .clear_var
      .2
      .=> X
      .3
      .=> Y
      .X
      .* Y
      .=> Z
      % Clears variables X and Y
>     .clear_var X Y
```

See also: .=>

.click, .click_rel, .ck, .ckr

.click [*realx*] [*realy*]
.ck [*realx*] [*realy*]

Perform the same function is as if the cursor was moved to the coordinates (*realx*,*realy*) and the primary mouse button was clicked. This is the way coordinates are specified to the various commands such as **.create_element** etc.

.ck is a synonym for **.click** - its just shorter to type.

The cursor is actually moved on the screen and a "click" cursor is displayed briefly to indicate the action taking place. The cursor movement is timed to take about one second to reach the final position. Use this command when you want the user to see the clicking action explicitly; for example in a script illustrating the actual functioning of Raytrace. Use **.fast_click** if you just want to perform a mouse click without the user noticing.

It does not matter if the coordinates lie within the Raytrace window or not.

.click_rel [*realdx*] [*realdy*]
.ckr [*realdx*] [*realdy*]

Like click but (*realdx*,*realdy*) specify a displacement relative to the last mouse click through which the cursor is moved.

.ckr is a synonym for **.click_rel**

Example:

```
      % Example of creating a ray from (10,30) to (110, 230).
      .create_ray
>     .click 0 0
>     .click_rel 100 200
      .finished_rays
```

See also: .fast_click and .move

.clockwise_arc See **.anticlockwise_arc**

.close_script

Terminates the current script and closes the script dialog box.

See also: .return

.conicseg See **.arcseg**

.conic_options

.conic_options *option* [*ecc*]

option is one of focus_end_axis
focus_vertex_endangle
vertex_axis_eccent
vertex_end_eccent
focus_end_eccent

ecc is the eccentricity. Required only for the last three options

Sets the conic options (see the Raytrace reference manual for a full description) for newly created conics. The five key words correspond to the five options presented in the dialog box started by the menu item **Defaults -- Conic Settings**.

If an invalid keyword is used then the error "Invalid conic option" is reported.

See **.change_conic** for an example.

See also: .change_conic

.control_key, .shift_key

.control_key [*flag*]
.shift_key [*flag*]

Set the apparent state of the control or shift keys - this is independent of the actual state of the keyboard. If *flag* is omitted then the current state is toggled. These commands should always be used in pairs within a script so as not to change the state on the user.

Example:

```
      % User will be given drag control with fine drag option set
>     .shift_key 1
      .allow_drag
>     .shift_key 0
```

.converging_lens, diverging_lens

Create a thin lens element with the specified convergence. Must be followed by appropriate click type commands to specify the position, height and focal length of the lens. Usage is subject to the setting of the menu item **Create -- Par-axials on centre** which may be set using the command **.paraxials_on_centre**.

When the lens is completed its object handle is placed in the accumulator for later reference.

Example:

```
      % Use the centre/height specification for the lens
      .paraxials_on_centre 1
>     .converging_lens
      % Centre point
      .click 0 0
      % Height and orientation
      .click 0 100
      % Focal length
      .click 200 0
      % Store handle for later use
      .=> LENS
      .pause
      .unselect_all
      .select_object LENS
      .make_diverging
```

See also: .paraxials_on_centre

.cos, .sin, .tan

.cos [*real*]
.sin [*real*]
.tan [*real*]

Place the trig function of *real* in the accumulator.

Example:

```
      % Find the sine of a constant value (in radians)
>     .sin 27.5
      .printf %g

      % Find the tangent of a value in a variable
      .45.67
      .=> THETA
>     .tan THETA
      .printf %g
```

See also: .arccos

.create_grating

.create_grating [*type spacing orders*]

type is one of transmission or reflection
spacing is the line spacing of the grating in metres
orders is a list of the diffraction orders to be generated from this grating i.e. numbers between -3 and +3 (Note that the *orders* setting can be overridden by the ray fertility setting

Create a diffraction grating element. Must be followed by appropriate click type commands. If no parameters are given on the command line then the user is prompted for the grating options.

Example:

```
>     .create_grating transmission 1.8e-6 -2 -1 0 1 2
```

See also: .converging_lens and .paraxial_mirror

.create_iris

Create an iris element. Must be followed by appropriate click type commands.

See also: .converging_lens and .paraxial_mirror

.create_ray, create_rays, .finished_rays

Begin creation of rays. Must be followed by appropriate click type commands to specify the coordinates of the ray(s). As each ray is completed its object handle is placed in the accumulator. When all desired rays are created use the command **.finished_rays**.

.create_rays is a synonym for **.create_ray** - it save remembering whether it should be plural or not.

Example:

```
      % Create three rays for fun
>     .create_ray
      .red
      .click 0 0
      .click 100 0
      .=> RAY1
      .green
      .click 0 0
      .click 100 -100
      .=> RAY2
      .blue
      .click 0 0
      .click 100 100
      .=> RAY3
      .finished_rays
      .select_object RAY1
      .arrow_end
      .unselect_all
      .pause
      % And now delete them one at a time
      .select_object RAY1
```

```
.delete
.pause
.select_object RAY2
.delete
.pause
.select_object RAY3
.delete
```

create_region, .create_shape, .create_surface, .create_screen, .end_element

Begin creation of an element of specified type. Must be followed by appropriate click type commands to specify the coordinates of the vertices. Complete the element using the **.end_element** command.

When the element is completed the object handle of the element is placed in the accumulator. It may then be stored in a variable to allow later reference to the element.

Example:

```
      % Create a simple square prism then delete it
      .new_text
      Click where you want the lower left corner of the square
      .user_click
>     .create_region
      .fast_click_rel 0 0
      .fast_click_rel 100 0
      .fast_click_rel 0 100
      .fast_click_rel -100 0
      .end_element
      .=> SQUARE
      .pause
      .new_text
      .unselect_all
      .select_object SQUARE
      .delete
```

See also: .arcseg, .converging_lens and .paraxial_mirror

.cut, .copy, .paste

These three commands are the same as using the menu items **Edit -- Cut**, **Edit -- Copy** and **Edit -- Paste**. They must be followed by an appropriate click type command to specify the base point coordinates except in the case of **.copy** when no objects are selected since this does not require a base point.

Example:

```
      % Select all elements and rays
      .all_elements
      .all_rays
      % Make a copy with origin as base point
>     .copy
      .fast_click 0 0
      % Paste another copy in, moved through displacement (100,100)
>     .paste
      .fast_click 100 100
```

.cycle_colour

Changes the colours of all selected rays one step in the sequence red->green->blue->red starting at their current colour. Same as pressing Shift+F2 in manual control.

See also: .blue and .white_ray

.default_input

.default_input [*real*]

Sets the default value which will appear in the edit control when a **userinput** keyword is used as an argument to a command.

Example:

```
      % Ask the user for a numerical value
      .0
      .=> RADIUS
      .new_text
      Please enter a value for the radius
      % Set default input value to 150
```

```
>     .default_input 150
      .RADIUS userinput
```

See also: .input_limits

.deg>rad, .rad>deg

.deg>rad [*real*] Converts the real argument from degrees to radians
.rad>deg [*real*] Converts the real argument from radians to degrees

Note that all trig functions use arguments in radians and inverse trig functions return values in radians.

Example:

```
      % Convert a value in the accumulator to radians and print it
      .123.4
>     .deg>rad
      .=> VALUE
      .printf %g VALUE

      % Convert value to degrees and display
>     .rad>deg VALUE
      .printf %g
```

.delete

Deletes all selected objects from the ray diagram

See also: .clear_all, .clear_annotations

.diverging_lens See **.converging_lens**

.drag_handles

.drag_handles [*flag*]

If *flag* is 0 then drag handles are not displayed in selected objects while the script is running.
If *flag* is non-zero then drag handles are displayed on selected objects.
If *flag* is omitted then the state is toggled.

Controls whether drag handles are displayed during the script. This is useful if you want to perform an **.auto_trace** without the drag handles being displayed.

See also: .auto_trace

.drag_single

If a drag operation is performed on a source then using this command before the drag is initiated sets the option of dragging a single ray. If this is not used then the operation defaults to dragging all the rays from the source. Applies only to the next source drag operation.

Example:

```
      % Create a point source for example
      .source_ray_count 10
      .point_source
      .fast_click 0 0
      .fast_click 100 0
      .fast_click 0 100
      % Save handle to source for later use
      .=> SRC
      % Select the source rays with next click
      .fast_click 0 0
      % Drag initiated by the next click will operate on all the rays
```

```
        .click 0 0
        % Move centre of source
        .click -50 50
        .pause 1000
        .unselect_all
        % Find the fourth ray of the source (0 based count)
        .ray_of_source SRC 3
        .=> RAY3
        .select_object RAY3
        % Find the end of the ray
        .object_coordinates RAY3 end
        % The next drag will only affect the one ray
>       .drag_single
        % Click on end of 4th ray to drag it
        .click X Y
        .click 200 -50
```

.element_off, .element_on

Same functions as the menu items **Modify -- Element > Turn Off** and **Modify -- Element > Turn On**. These commands operate on all selected elements. If elements are turned off then they still appear in the ray diagram but they do not interact with rays.

.enable_undo

.enable_undo *flag*

If *flag* is 0 then undo information is not saved during the script.
If *flag* is non-zero then undo information is saved.
If *flag* is omitted then the state is toggled.

Normally Raytrace saves undo information in a file before every operation. This is relatively quick and you are unlikely to notice the short delay involved when using Raytrace manually. However when many operations are performed in sequence in a script there is rarely any need to save undo information and the delays accumulate and become noticeable. This command allows you to stop the saving of undo information so that scripts execute quicker. This has no effect on the undo function outside the script facility but does affect the use of undo when control is passed to the user with **.user_control**.

See the section on a typical script prototype for more information.

Example:

```
>       .enable_undo 0
        % Perform functions within script where undo is not needed
        ...
        % Enable undo before giving the user control
>       .enable_undo 1
        .user_control
```

See also: .undo

.end_element

Terminates any of **.create_region**, **.create_surface** or **.create_shape** commands.

See **.create_region** for more information and for an example.

.end_snap See **.centre_snap**

.exit_raytrace

Same as choosing **File -- Quit**; closes the Raytrace application. Don't confuse with **.close_script**.

.exp, .log_e., .10^, log_10

.exp [*real*] Place the inverse natural logarithm of *real* in the accumulator.
.log_e [*real*] Place the natural logarithm of *real* in the accumulator.
.10^ [*real*] Place the inverse logarithm to base 10 of *real* in the accumulator.
.log_10 [*real*] Place the logarithm to base 10 of *real* in the accumulator.

Example:

```
      .123.4
>     .log_e
      .=> LOGVAL
      .printf %g

>     .exp LOGVAL
      .printf %g
```

.explode, .join

.explode

Same function as the menu items **Modify -- Element > Explode**. Breaks an element up into individual segments.

.join

Same function as and **Modify -- Element > Join**. Joins elements together into a single element.

See the Raytrace reference manual for more information on these functions.

.fast_click, .fast_click_rel, .fc, .fcr

.fast_click [*realx*] [*realy*]
.fc [*realx*] [*realy*]
.fast_click_rel [*realdx*] [*realdy*]
.fcr [*realdx*] [*realdy*]

Basically the same commands as **.click** and **.click_rel** except the cursor is not moved on the screen and hence these commands are much faster. Use these commands when you just want to specify points without the user seeing the cursor movements.

.fc is a synonym for **.fast_click** its just faster to type.
.fcr is a synonym for **.fast_click_rel**

See **.fast_move** for an example.

See also: .click

.fast_move, .fast_move_rel

.fast_move [*realx*] [*realy*]
.fast_move_rel [*realdx*] [*realdy*]

Basically the same commands as **.move** and **.move_rel** except the cursor is moved immediately to the new position. Use this function to ensure the cursor is in the desired position when you hand over control to the user with a function like **.allow_drag**.

Example:

```
      .create_ray
      .fc 0 0
      .fc 100 100
      .finished_rays
      .select_object
      .fc 100 100
      % Without the next line the cursor could be anywhere.
>     .fast_move 100 100
      .allow_drag
      .ck 100 100
```

See also: .click, .fast_click and .move

.fc See **.fast_click**

.file_new

Same function as the menu item **File -- New**. See the Raytrace reference manual for a full description. Basically it clears the ray diagram and resets some default parameters. It's good idea to use this at the start of scripts which create their own ray diagrams.

See also: .clear_all

.finished_rays

See **.create_ray**

.focus_snap

See **.centre_snap**

.forward_project

See **.back_project**

.goto

.goto *LABEL*

Transfers execution of the script to the line following :*LABEL where LABEL* is a label name of your choice. If there is no occurrence of :*LABEL* in the script file then the error message "Missing label" is reported.

See **.choose** for an example

See also: .choose, .if== and .yes_no

.green

See **.blue**

.green_index

See **.blue_index**

.green_rays

See **.blue_rays**

.grid, .grid_size

.grid [*flag*]

If *flag* is zero, turns the display of the grid off.
If *flag* is non-zero then turns the display of the grid on.
If *flag* is omitted, toggles the display of the grid.

The current state of the grid display is put in the accumulator.

.grid_size *realx realy*

Sets the grid spacing to *realx* in the x direction and *realy* in the y direction

See also: .grid_snap

.grid_snap

See **.centre_snap**

.group, .ungroup

.group
.ungroup

Same functions as the menu items **Modify -- Element > Group** and **Modify -- Element > Ungoup**. See the Raytrace reference manual for more information.

.if==, .if!=, .if<=, .if>=, .if<, .if>

.if== *real LABEL* Transfer if accumulator equal to *real*.
.if!= *real LABEL* Transfer if accumulator not equal to *real*.
.if<= *real LABEL* Transfer if accumulator less than or equal to *real*.
.if>= *real LABEL* Transfer if accumulator greater than or equal to *real*.
.if< *real LABEL* Transfer if accumulator less than *real*.
.if> *real LABEL* Transfer if accumulator greater than *real*.

These commands provide conditional transfer of execution. The contents of the accumulator are compared to *real*. If the condition is met then execution is transferred to the line following :*LABEL* in the script file. If :*LABEL* does not exist then the error message "Missing label" is reported.

Example:

```
      % Turn off the arrow on a user selected ray
      .unselect_all
      :RETRY
      .new_text
      Select a ray by clicking on it somewhere.
      .user_click
      .selected_ray
>     .if!= -1 OK
      .new_text
      You have not selected a ray. Click on Continue to try again.
      .pause
      .goto RETRY
      :OK
      .arrow_none
      .close_script
```

See also: .choose, .goto, .yes_no and .--if!=0

.input_limits

Sets the limits of values which will be accepted from the user in response to the use of the **userinput** keyword. If the user enters a value outside the set range then the appropriate limit is used. In the example shown below, if the user enters a value less than 50 then 50 will be used as the radius, if a value greater than 1000 is entered then 1000 will be used.

Example:

```
      % Ask the user for a numerical value
      .0
      .=> RADIUS
      .new_text
      Please enter a value for the radius
      % Set default input value to 150
      .default_input 150
      % Set acceptable limits to between 50 and 1000 inlc.
>     .input_limits 50 1000
      .RADIUS userinput
```

See also: .default_input

.intersect_snap See **.centre_snap**

.join See **.explode**

.lastclick.x, .lastclick.y

See the section "**The script calculator**" for info on this keyword.

.lineseg See **.arcseg**

.link_by_child_rays

The same function as **Modify -- Source > Link point source via child rays**. A point source must be selected (by having one of its rays selected) and two child rays of its extreme ray must be selected before using this command. Must also be followed by appropriate click type commands to set the two aperture points through which the child rays will be forced to pass.

Example:

```
      % What does the eye see through a simple magnifier
      .file_new
      .enable_undo 0
      .create_region
      .fc 0 60
      .arcseg
      .chord_tan
      .fc 0 -60
      .fc -30 -60
      .fc 0 60
      .fc 30 60
      .end_element
      .=> LENS
      % Create an "object" shape to the left of the lens
      .create_shape
      .fc -80 20
      .fc -80 0
      .fc -100 0
      .fc -100 20
      .fc -80 20
      .end_element
      .=> OBJECT
      % Create a point source with two rays coming from the "object"
      .set_length 200
      .source_ray_count 2
      .point_source
      .fc -80 20
      .fc 0 -30
      .fc 0 30
      .=> SRC
      % Create an eye by calling supplied script
      .180
      .=> EYEX
      .30
      .=> EYEY
      .180
      .=> ANGLE
      .call eye.rsc
      % Now finally get to link the source via its child rays.
      % First select the source
      .select_object SRC
      % Select the transmitted child rays of the extreme source rays
      .ray_of_source SRC 0
      .=> RAY
      .child_by RAY refraction refraction
      .=> RAY1
      .select_object
      .ray_of_source SRC 1
      .=> RAY
      .child_by RAY refraction refraction
      .=> RAY2
      .select_object
>     .link_by_child_rays
      .click 180 23
      .click 180 33
      % Backproject rays entering eye and create a trail to
      % show what the eye sees
      .unselect_all
      .select_object RAY1 RAY2
      .back_project 1
      .trail
      .intersect_snap
      .control_key 1
      .fc 10000 10000
      .control_key 0
      % Trace around the object
      .drag_handles 0
```

```
.unselect_all
.select_object SRC OBJECT
.auto_trace 1 0
.drag_handles 1
.enable_undo 1
```

.link_segments, .link_move, .link_stretch, .unlink_segments

.link_segments

The same function as the menu item **Modify -- Link segments**. Must be followed by appropriate click type commands to select the segments which will be linked. See the Raytrace reference manual for more information on linking segments of elements.

.unlink_segments

The same function as **Modify -- Unlink segments**. Unlinks any segment links on all selected elements.

.link_move
.link_stretch

When using **Modify -- Link segments** in manual control you are presented with a dialog box asking you to select from two options; move or stretch the linked element. These commands can be used prior to **.link_segments** in a script to select these options.

Example:

```
% Create a plano-convex cemented doublet similar to the
% one in the Tutorial manual tutorial.
% Create left part of the doublet
.create_region
.fc -50 60
.arcseg
.chord_tan
.fc -50 -60
.fc 10 -60
% Second arc
.fc -50 60
.fc -100 60
.end_element
.=> LEFTPART
% Create the right part of the doublet
.create_region
.fc 50 60
.fc 30 60
.arcseg
.fc 30 -60
.fc 90 -60
.lineseg
.fc 50 -60
.end_element
.=> RIGHTPART
% Set the linking option
>    .link_move
% And start the linking process
>    .link_segments
% Where to click on the left lens (middle of first segment)
.object_coordinates LEFTPART 1 middle
.click X Y
% Where to click on the right lens (middle of second segment)
.object_coordinates RIGHTPART 2 middle
.click X Y
% It is done, you can now set the refractive indices.
.close_script
```

.lock, .lock_x, .lock_y, .unlock

.lock	Lock all selected elements so they cannot be dragged.
.lock_x	Lock all selected elements so they can only be dragged in the y direction.
.lock_y	Lock all selected elements so they can only be dragged in the x direction.

.unlock Remove any locks placed on all selected elements.

Same functions as the menu items in the menu **Modify -- Element > Lock >**.

Example:

```
      % Lock a user selected element
      .unselect_all
      :RETRY
      .new_text
      Select a element by clicking on it somewhere.
      .user_click
      .selected_element
      .if!= -1 OK
      .new_text
      You have not selected an element.

      Click on Continue to try again.
      .pause
      .goto RETRY
      :OK
>     .lock
      .close_script
```

.log_e, .log_10

See **.exp**

.make_converging, .make_diverging

Same functions as the menu items **Modify -- Element > Make Converging** and **Modify -- Element > Make Diverging**. Operates on all selected thin lens type elements to make them into converging/diverging lenses respectively.

.make_region, .make_screen, .make_shape, .make_surface

Same functions as the menu items in **Modify -- Element > Change to >**. Operates on all selected elements (other than par-axial type elements) to turn them into the specified type of element.

See also: .create_region

.material

.material *Name*

Select refractive indices of a material called *Name* from the raytrace.mat file. There must be a single space between the end of **.material** and the start of *Name*. *Name* may contain spaces and is taken as everything up to the end of the line. If any elements are selected then their refractive indices are changed to the values of the material. If no elements are selected then the default values for new elements are set.

Example:

```
      .create_region
      .fc 0 0
      .fc 100 0
      .fc 50 100
      .end_element
      .select_object
      % Set refractive index of element just created and selected
>     .material heavy flint
      .unselect_all
      % Set default refractive indices
>     .material quartz
```

See also: .refractive_index

.max, .min

.max *real*
.min *real*

Places the maximum/minimum of the contents of the accumulator and *real* into the accumulator

Example:

```
.123.4
.max 456
.printf %g

% End of example
```

.max_path_depth

.max_path_depth [*integer*]

Sets the maximum depth to which child rays will be generated from any user created ray to *integer*. Same function as the menu item **Options -- Maximum path depth...**.

.mid_snap See **.centre_snap**

.mirror

Same function as the menu item **Modify -- Mirror**. Must be followed by two click type commands to specify the end points of the mirror line. Mirrors all selected objects about the specified line.

Example:

```
      % Create a simple element then mirror it
      .create_surface
      .fc 0 0
      .fc 100 0
      .fc 100 100
      .end_element
      .=> CORNER
      .select_object CORNER
>     .mirror
      .click 0 0
      .click 0 100
      .unselect_all
```

See also: .move_selection, .rotate and .scale

.more_steps? See **.auto_trace**

.mouse_limits

.mouse_limits *xmin ymin xmax ymax*

Limits the region in which the user can drag the mouse when allowed by an use of **.allow_drag** or **.user_click**. To allow dragging without limits, set *xmin*, *xmax* etc to very big values like 1E5.

See **.allow_drag** for an example

See also: .allow_drag and .user_click

.move, .move_rel

.move [*realx*] [*realy*]

Moves the cursor on the screen to the specified coordinates (*realx*,*realy*). The movement is timed to take about one second.

.move_rel [*realdx*] [*realdy*]

Same as move but the coordinates are specified relative to the most recent mouse click.

Use these commands to move the cursor around so that the user can follow what is happening in demo type scripts.

See also: .click and .fast_move

.move_selection

Same function as the menu item **Modify -- Move**. Must be followed by two click type commands to specify the "from" and "to" points. Moves the selected objects so that the "from" point is mapped onto the "to" point.

Example:

```
      % Create a simple element then move it
      .create_surface
      .fc 0 0
      .fc 100 0
      .fc 100 100
      .end_element
      .=> CORNER
      .select_object CORNER
>     .move_selection
      .click 0 0
      .click 100 100
      .unselect_all
```

See also: .mirror, .rotate and .scale

.new_text

Clears any existing text from the script dialog box display. New text is added to the display by including it within the script. Once the available space in the script dialog fills with text further text will be lost from view. You need to keep comments or directions short and direct and use **.new_text** frequently.

Example:

```
>     .new_text
      Click on continue to see more text appear.
      .pause

      Now click on continue to clear the text and replace it with
      something new.
      .pause
>     .new_text
      Something new.
```

.normal

See **.back_project**

.no_snap

See **.centre_snap**

.object_coordinates

In general, finds the coordinates of some point on an object and places them in the script variables X and Y. These variables are created if necessary. The syntax depends upon the object type as described below. The first argument is always an object handle which is placed in the accumulator when an objected is created.

.object_coordinates *element segment type*

element	object handle of an element
segment	segment number - first segment is numbered 1
type	one of end middle centre tangent focus vertex

To obtain the starting point of an element use a segment number of 0 and *type* = end, i.e. **.object_coordinates** *element* 0 end

When using with a thin lens element which has two foci use:
.object_coordinates *element* 0 focus to obtain the first focus point
and **.object_coordinates** *element* 1 focus to obtain the second focus point.

When using with an iris element you can find any of the four end points using
.object_coordinates *element* n end where n = 0,1,2 or 3
the only other point on an iris that you can find is the middle of the aperture using

.object_coordinates *element* 0 middle

.object_coordinates *ray type*
ray object handle of a ray
type one of start middle end

.object_coordinates *tapemeasure type*
tapemeasure object handle of a tapemeasure
type one of end_one end_two middle cartesian polar lead_to

type = cartesian places the (x,y) displacement of the ruler in the variables X and Y
type = polar places the length of the tapemeasure in X and the angle in degrees Y

.object_coordinates *protractor type*
protractor object handle of a protractor
type one of centre end_one end_two angle lead_to

type = angle places the angle in radians in X and in degrees in Y

.object_coordinates *annotation type*
annotation object handle of an annotation
type one of base lead_to

.object_coordinates *trail*
trail object handle of a trail

Always places the coordinates of the current position of the trail in X,Y. There are no "type" options.

.object_coordinates *point_source type*
point_source object handle of a point source
type one of centre aperture_one aperture_two

.object_coordinates *plane_source type*
plane_source object handle of a plane source
type one of base aperture_one aperture_two

There are two errors which may be reported, these are self explanatory:
"Invalid object handle" and "Invalid key word for .object_coordinates"

Example:

```
      % Demonstrate first on a region element
      .create_region
      .fc 0 0
      .fc 100 0
      .arcseg
      .chord_tan
      .fc 50 100
      .fc 150 100
      .conicseg
      .fc 0 120
      .fc 0 150
      .fc -50 100
      .end_element
      .=> REGION
      % Find the starting point
>     .object_coordinates REGION 0 end
      .printf %g X
      ,
      .printf %g Y

      % Find the end of the first side
>     .object_coordinates REGION 1 end
      .printf %g X
      ,
      .printf %g Y

      % Find the centre of the arc segment
>     .object_coordinates REGION 2 centre
      .printf %g X
      ,
      .printf %g Y
```

```
      % Find the vertex of the conic segment
>     .object_coordinates REGION 3 vertex
      .printf %g X
      ,
      .printf %g Y

      % Find the middle of the closing segment
>     .object_coordinates REGION 4 middle
      .printf %g X
      ,
      .printf %g Y

      .pause
      .select_object REGION
      .delete
      % Now demonstrate on a ray
      .new_text
      .create_ray
      .fc 10 10
      .fc 120 130
      .=> RAY
      .finished_rays
>     .object_coordinates RAY middle
      .printf %g X
      ,
      .printf %g Y

      .pause
      .select_object RAY
      .delete
```

.object_type

.object_type *handle*

handle is an object handle

Determines the type of object belonging to the *handle*. Places the type in the accumulator where type is one of:

-1	invalid handle
1	region element
2	surface element
3	shape element
4	converging thin lens element
5	diverging thin lens element
6	paraxial mirror element
7	annotation
8	protractor
9	tapemeasure
10	trail
11	point source
12	plane source
13	screen element
14	iris element
15	grating

Example:

```
      .create_shape
      .fc 0 0
      .fc 100 0
      .fc 100 100
      .end_element
      .=> OBJECT
>     .object_type OBJECT
      .printf %g
      .pause
      .select_object OBJECT
      .make_region
>     .object_type OBJECT
      .printf %g
```

```
      .pause
      .delete
>     .object_type OBJECT
      .printf %g
      % End of example script
```

.open, .save, .save/d

.open *filename*

Same as using **File -- Open**. Opens an existing ray file and displays it.

.save *filename*
.save/d *filename*

Same as using **File -- Save As**. Saves current ray diagram under in a file called *filename*. This command does not change the current filename as displayed in the raytrace window title bar.

.save/d saves the file but without compacting the internal database; this allows the ray diagram to be reload in exactly the same state as prior to the save. The saved file is generally much larger when using **.save/d** so only use it when necessary - see the section on strategies for writing scripts.

filename may contain a path and should include an extension.

All script variables are saved and loaded with these commands. A corollary is that all script variables existing at the time of an "**.open**" are wiped and replaced by whatever variables were in existence at the time file was saved.

Example:

```
>     .open disperse.ray
>     .save newdisp.ray
```

.optical_length, .physical_length

Calculate and sum either the optical length or the physical length of all selected rays and place the result in the accumulator. The optical length of a ray is the physical length multiplied by the refractive index of the medium in which it is contained.

.painting

.painting [*flag*]

Normally Raytrace updates its display after every function. When using Raytrace manually the delay caused by repainting the window is negligible however, when many functions are performed in sequence in a script the delays accumulate and can be annoying. This command allows you to stop the repainting of the window and thus speed up script execution; only painting when the desired display is ready.

If *flag* is 0 then turns painting off.
If *flag* is non-zero then turns painting on.
If *flag* is omitted then toggles the painting state.

The updated painting state is placed in the accumulator.

When the painting state is re-enabled the Raytrace window is automatically repainted.

Example:

```
      % Turn painting off to speed script up
>     .painting 0
      .create_region
      .fc 0 0
      .fc 100 0
      .fc 50 100
      .end_element
      .source_ray_count 10
```

```
      .point_source
      .fc -100 30
      .end_snap
      .fc 0 0
      .end_snap
      .fc 50 100
      % Turn painting on and reveal result
>     .painting 1
```

See also: .recalc

.pan_left, .pan_right, .scroll_up, .scroll_down

Move the window around over the ray diagram as if the scroll buttons on the Raytrace window were used.

Example:

```
      .open disperse.ray
      .pause
>     .pan_left
      .pause
>     .pan_right
      .pause
>     .scroll_up
      .pause
>     .scroll_down
      .close_script
```

.paraxial_mirror

Create a par-axial mirror element. Must be followed by appropriate click type commands to specify the position, height and focal length of the mirror. Usage is subject to the setting of the **Create -- Par-axials on centre** menu item which may be set using the command **.paraxials_on_centre**.

When the mirror s completed its object handle is placed in the accumulator for later reference.

Example:

```
      % Use the aperture ends specification for the mirror
      .paraxials_on_centre 0
>     .paraxial_mirror
      % Aperture end point one
      .click 0 0
      % Other aperture end
      .click 0 100
      % Focal length
      .click 200 0
      % Store handle for later use
      .=> MIRROR
```

See also: .converging_lens, .create_region and .paraxials_on_centre

.paraxials_on_centre

.paraxials_on_centre [*flag*]

Same function as available with the menu item **Create -- Par-axials on centre**.

If *flag* is 0 then turns the option off.
If *flag* is non-zero then turns the option on.
If *flag* is omitted then toggles the setting of the option.

The updated state of the option is placed in the accumulator.

See also: .converging_lens and paraxial_mirror

.parent_reflect, .parent_refract

See **.back_project**

.paste

See **.cut**

.pause

.pause [*delay*]

Pauses execution of the script for *delay* milliseconds.

If *delay* is omitted then stops execution of the script and enables the Continue button in the script dialog box. Execution continues when the user clicks on the continue button.

.pause_trails, .reset_trails

.pause_trails [*flag*]

Same function as the menu item **Modify -- Pause Trail update**.

If *flag* is 0 then suspends updating of all trails.
If *flag* is non-zero then resumes updating of all trails.
If *flag* is omitted then toggles the Modify -- Pause Trail update state.

The updated state of the option is placed in the accumulator.

.reset_trails

Same function as the menu item **Modify -- Reset Trails**.

See also: .trail

.perp_snap

See **.centre_snap**

.physical_length

See **.optical_length**

.plane_source, .point_source, .source_ray_count

.plane_source
.point_source

Create a source. Must be followed by appropriate click type commands to define the three points needed.

When the source is completed its object handle is placed in the accumulator.

.source_ray_count [*integer*]

Sets the number of rays in either selected sources or as the default for sources to be created.

Example:

```
>     .source_ray_count 10
>     .plane_source
      .click 0 100
      .click 100 100
      .click 100 0
>     .source_ray_count 30
>     .point_source
      .click -50 -50
      .click 0 -100
      .click 0 -20
      .=> PNT
```

.printf

.printf *format* [*real*]

Converts *real* to a string and appends it to the text appearing in the script dialog box.

format is a format specifier string which controls how the number is output. It has the same conventions as used with the printf function in C programming as summarised below.

format = % [flags] [width] [.prec] [F|N|h|l|L] type_char

format specifiers begin with the percent character (%). After the % come the following, in this order:

Component	Opt./Req.	What It Controls or Specifies
[flags]	(Optional)	Flag character(s) Output justification, numeric signs, decimal points, trailing zeros
[width]	(Optional)	Width specifier Minimum number of characters to print, padding with blanks or zeros
[.prec]	(Optional)	Precision specifier Maximum number of characters to print; type_char
	(Required)	Conversion type character

type_char must be one of the following for correct functioning.

f	Floating point	Signed value of the form [-]dddd.dddd.
e	Floating point	Signed value of the form [-]d.dddd or e[+/-]ddd
g	Floating point	Signed value in either e or f form, based on given value and precision. Trailing zeros and the decimal point are printed if necessary.
E	Floating point	Same as e; with E for exponent.
G	Floating point	Same as g; with E for exponent if e format used

Example:

```
      % Print a number in different formats
>     .printf %6.2f 123.456

>     .printf %g 123.456

>     .printf %1.4e 123.456

      % Need the blank line above to update output after .printf
```

See also: .new_text

.protractor

.protractor

Creates a protractor object. Must be followed by appropriate click type commands to define the centre and end points of the protractor.

Example:

```
% Create a protractor that measures the angle between two rays
% User selects the two rays
:START
.unselect_all
.new_text
Select the first ray by clicking on it.
:GET1
.user_click
.selected_ray
.=> RAY1
.if!= -1 OK1
.new_text
You have not selected a ray, try again
.goto GET1
:OK1
```

```
        .unselect_all
        .new_text
        Select the second ray by clicking on it.
        :GET2
        .user_click
        .selected_ray
        .=> RAY2
        .if!= -1 OK2
        .new_text
        You have not selected a ray, try again
        .goto GET2
        :OK2
        .RAY1
        .if!= RAY2 OK3
        .new_text
        You have selected the same ray twice.

        Click on continue to try again.
        .pause
        .goto START
        :OK3
        % Now create the protractor
>       .protractor
        % Centre on the intersection of the rays
        .intersect_snap
        .select_object RAY1 RAY2
        .control_key 1
        .fc 10000 10000
        .control_key 0
        % First end point on one ray
        .object_coordinates RAY1 end
        .end_snap
        .fc X Y
        % Second end on other ray
        .object_coordinates RAY2 end
        .end_snap
        .fc X Y
        .close_script
```

.protractor_options

.protractor_options *decimals lead display*

decimals	the number of decimal places to be displayed		
lead	one of:	leader	
		no_leader	
display	one of:	degrees	for angle in degrees
		radians	for angle in radians
		none	for no angle display

Example:

```
        .protractor
        .fc 0 0
        .fc 100 0
        .fc 100 100
        .=> PROT
        .select_object PROT
>       .protractor_options 3 no_leader radians
        .unselect_all
```

.quiet_snap

.quiet_snap [*flag*]

Raytrace normally responds to the result of a snap operation with a sound. This can be inappropriate in a script so this command allows you to control the
use of sound in snapping. Same as the menu item **Options -- Quiet Snap**.

If *flag* is 0 then turns the sound on.
If *flag* is non-zero then turns the sound off.

If *flag* is omitted then toggles the quiet snap state.

The updated state is placed in the accumulator.

.radius_end See **.centre_end**

.rad>deg See **.deg>rad**

.random

See the section "**The script calculator**" for info on this keyword.

.randomize

Randomizes the sequence of number generated when the keyword **random** is used in place of an argument. Without this, the same sequence of random numbers would be generated each time Raytrace was started.

.ray_diffraction

.ray_diffraction *orders*

orders is a list of the diffraction orders i.e. integer numbers between -3 and +3 inclusive

Sets the diffracted rays that will be generated if this ray strikes a diffraction grating element. This setting overrides the setting of the diffraction grating. If rays are selected then this command affects the settings for those rays otherwise it sets the default values for new rays.

See also: .create_grating

.ray_of_source

.ray_of_source *source N*

source is an object handle of a source

Finds the *N*th ray of a source and places its object handle in the accumulator. Rays are counted from 0.

Example:

```
      .source_ray_count 5
      .plane_source
      .fc 0 100
      .fc 100 100
      .fc 100 0
      .=> SRC
      % Select the first and last rays of the source
>     .ray_of_source SRC 0
      .select_object
>     .ray_of_source SRC 4
      .select_object
```

.recalc

.recalc [*flag*]

Normally Raytrace recalculates the ray paths after any function which could lead to a change in ray paths. When many functions are performed in sequence in a script it is often not necessary to recalculate the ray paths until a number of changes have been completed. Delays caused by recalculating ray paths and can be annoying. This command allows you to stop the recalculation and thus speed up script execution; only recalculating when the desired configuration is ready.

If *flag* is 0 then turns ray calculations off.
If *flag* is non-zero then turns ray calculations on.
If *flag* is omitted then toggles the recalc state.

The updated recalc state is placed in the accumulator.

When the recalc state is set to on the ray paths are automatically recalculated.

Example:

```
      % Turn ray calculations off while a lens is constructed
>     .recalc 0
      .call mklens.rsc
      % Turn the ray calculations back on and update
      .recalc 1
```

See also: .painting

.red

See **.blue**

.red_index

See **.blue_index**

.red_rays

See **.blue_rays**

.refract, .reflect, .reflect_if_no_refract

See **.back_project**

.refractive_index

.refractive_index *R G B*

R,*G* and *B* are the refractive indices for red, green and blue respectively. If any elements are selected then their refractive indices are changed to the specified values. If no elements are selected then the default refractive indices are changed.

Example:

```
.create_region
.fc 0 0
.fc 100 0
.fc 50 100
.end_element
.select_object
% Set refractive index of element just created and selected
.refractive_index 1.2 1.3 1.4
.unselect_all
% Set default refractive indices
.refractive_index 1.5 1.6 1.7
```

See also: .material

.reset_trails

See **.pause_trails**

.return

See **.call**

.rotate, .rotate_drag

.rotate [*angle*]

angle is the rotation angle in degrees (anticlockwise = positive). If this argument is omitted then the contents of the accumulator are used - do not confuse with **.rotate_drag**.

Same function as available with the menu item **Modify -- Rotate** if the rotate by angle option is selected. Must be followed by a click type command to specify the centre of the rotation.

Example:

```
      % Create a simple element then rotate it
      .create_surface
      .fc 0 0
      .fc 100 0
      .fc 100 100
      .end_element
      .=> CORNER
      .select_object CORNER
>     .rotate 30
      .click 0 0
      .unselect_all
```

.rotate_drag

Same function as available with the menu item **Modify -- Rotate** if the rotate by dragging option is selected. Must be followed by a click type command to specify the centre of the rotation and a click to specify the reference direction and then dragging starts.

Example:

```
      % Create a simple element then rotate it by dragging
      .create_surface
      .fc 0 0
      .fc 100 0
      .fc 100 100
      .end_element
      .=> CORNER
      .select_object CORNER
>     .rotate_drag
      .click 0 0
      .click 100 100
      .new_text
      Move the mouse to rotate the object. Click when finished.
      .user_click
      .unselect_all
```

See also: .mirror, .move_selection and .scale

.round, .truncate

.round [*real*] Rounds *real* to the nearest integer.

.truncate [*real*] Truncates *real* to the integer closer to zero.

Example:

```
      .-1.64
      .=> X
>     .round X
      .printf %g

>     .truncate X
      .printf %g

      % End of example script
```

.run_from, .script

.run_from *filename*

Sets the name of the script which will be run if the user clicks on the "Previous" button in the script dialog box to *filename*. Enables the "Previous" button.

.script *filename*

Executes a script file called *filename*. Do not confuse with **.call**, you cannot "return" to the original script.

Example:

In a script file called fred.rsc

```
      % Jump to the next script in sequence
>     .script bill.rsc
```

In a script file called bill.rsc

```
      % Can step back to previous script after next line executes
      % Works regardless of whether fred.rsc was run or not
>     .run_from fred.rsc
```

See also: .call

.save, .save/d

See **.open**

.scale

.scale [*real*]

Same function as the menu item **Modify -- Scale**. Must be followed by a click type command to specify the centre of the scale operation. Scales all selected objects about the specified point by the factor *real*.

Example:

```
      % Create a simple element then scale it
      .create_surface
      .fc 0 0
      .fc 100 0
      .fc 100 100
      .end_element
      .=> CORNER
      .select_object CORNER
>     .scale 1.5
      .click 0 0
      .unselect_all
```

See also: .mirror, .move_selection and .rotate

.script

See **.run_from**

.scroll_up, .scroll_down

See **.pan_left**

.segment_count

.segment_count *element*

element is an object handle of an element.

Counts the number of segments making up the element and places the result in the accumulator. Note that the closing segment of region elements is not counted.

Example:

```
      .create_region
      .fc 0 0
      .fc 100 0
      .fc 50 100
      .end_element
      .=> TRI
>     .segment_count TRI
      .printf %g

      % End of example script
```

.selected_annotation, .selected_element, .selected_protractor, .selected_ray, .selected_tapemeasure, .selected_trail

If an object of the specified type is selected then places the object handle of that object in the accumulator. The results are not always predictable if more than one object of the same type is selected.

See **.protractor** for an example using **.selected_ray**; others are used in a similar manner.

.select_child_rays

Same function as the menu item **Edit -- Select Child Rays**. See the Raytrace reference manual for more information.

.select_extended

Same function as the menu item **Edit -- Select extended** or pressing the short cut key Shift+F3. Must be followed by appropriate click type commands to specify the corner of the selection rectangle. See the Raytrace reference manual for more information.

.select_group

Same function as the menu item **Edit -- Select Group**. See the Raytrace reference manual for more information.

.select_object

.select_object *handle*

Selects the object whose object handle is given.

Example:

```
      .create_ray
      .fc 0 0
      .fc 100 100
      .=> RAY
      .finished_rays
>     .select_object RAY
```

.set_length

.set_length [*real*]

Sets the default length of any selected rays. If no rays are selected then sets the default length for rays created in the future. Only has an apparent effect on rays that do not interact with an element.

Example:

```
      .file_new
      .source_ray_count 5
>     .set_length 40
      .point_source
      .fc -50 0
      .fc 0 0
      .fc 0 50
>     .set_length 80
      .point_source
      .fc 0 0
      .fc 100 0
      .fc 100 100
```

.set_radius

.set_radius [*real*]

When an arc tangent control point is being dragged, use this command to set the arc's radius to the value *real*. Same as the function which is accessed using the secondary mouse button while dragging an arc tangent control point. To set the arc to an infinite radius set *real* = 0. Completes the dragging operation just as it does in manual control.

Example:

```
      .create_surface
      .fc 0 0
      .arcseg
      .chord_tan
      .fc 0 100
      .fc 50 100
      .end_element
      .=> ARC
      .select_object ARC
      .object_coordinates ARC 1 tangent
      .fc X Y
>     .set_radius 60
```

.shift_key

See **.control_key**

.show_annotations, .show_protractors, .show_tapemeasures, .show_trails

.show_annotations [*flag*]

.show_protractors [*flag*]
.show_tapemeasures [*flag*]

Controls the setting of the menu items **Options -- Show > Annotations** etc.

If *flag* is 0 then the objects will be hidden.
If *flag* is non-zero then the objects will be shown.
If *flag* is omitted the the state is toggled.

The updated state is placed in the accumulator.

.sign

See **.abs**

.sin

See **.cos**

.source_ray_count

See **.plane_source**

.sqrt, .^2

.sqrt [*real*]

Places the square root of *real* in the accumulator. If *real* is negative then reports the error "Invalid argument".

.^2 [*real*]

Squares *real* and places the result in the accumulator.

.step

See **.auto_trace**

.tan

See **.cos**

.tangent_snap

See **.centre_snap**

.tapemeasure

Creates a tapemeasure object. Must be followed by appropriate click type commands to specify the end points of the tape measure.

Example:

```
      .file_new
      % Create an arc to do measuring on
      .create_surface
      .fc 0 0
      .arcseg
      .chord_tan
      .fc -10 100
      .fc 150 100
      .end_element
      % Snap a tapemeasure between centre and mid point
>     .tapemeasure
      .centre_snap
      .fc 0 0
      .mid_snap
      .fc 0 0
      .=> TAPE
      .select_object TAPE
      % Modify tapemeasure options
      .tapemeasure_options 3 1 leader degrees
      % Move leader
      .object_coordinates TAPE lead_to
      .fc X Y
      .fast_click_rel -30 20
      .unselect_all
```

.tapemeasure_options

.tapemeasure_options *dist_decimals angle_decimals lead display*

dist_decimals	number of decimal places for distance display		
angle_decimals	number of decimal places for polar angle display		
lead	one of	leader	
		no_leader	
display	one of	distance	display only length
		cartesian	display x,y displacement
		degrees	polar form, angle in degrees
		radians	polar form, angle in radians
		none	no readout displayed

See **.tapemeasure** for an example.

.terminal_snap

See **.centre_snap**

.thin_lenses_with_background

.thin_lenses_with_background *flag*

If *flag* is zero then unchecks the menu item **Modify -- Thin lenses with background**
If *flag* is non-zero then checks **Modify -- Thin lenses with background**
If *flag* is omitted then toggles the check state of **Modify -- Thin lenses with background**

.trail

Create a trail object. Must be followed by a click type command with an object snap.

Example:

```
      .file_new
      % Create 2 rays, will put trail on intersection
      .create_ray
      .fc 0 0
      .fc 20 20
      .fc 0 100
      .fc 20 80
      .finished_rays
      % Select the 2 rays
      .all_rays
>     .trail
      % Snap trail on intersection
      .intersect_snap
      .control_key 1
      .fc 10000 10000
      .control_key 0
      % Re-select rays for dragging
      .all_rays
      %Drag end of first ray
      .click 20 20
      .click 20 30
      % Drag end of second ray
      .click 20 80
      .click 20 70
      .unselect_all
```

.truncate

See **.round**

.undo

Same as selecting **Edit -- Undo**. Will not be fully functional if **.enable_undo** has been used to suspend saving of the undo information.

See also: .enable_undo

.ungroup

See **.group**

.unlink_segments

See **.link_segments**

.unlink_sources

Same function as the menu item **Modify -- Source > Unlink snaps**. Breaks any links between all selected sources and any other objects in the ray diagram.

.unlock

See **.lock**

.unselect_all

Unselects all selected objects in the ray diagram. A good idea to use this before establishing a new selection set just to be sure you don't have something selected you did not know about.

See also: .unselect_annotations

.unselect_annotations, .unselect_elements, .unselect_protractors, .unselect_rays, .unselect_tapemeasures, .unselect_trails

Unselect all objects of the specified type.

Example:

```
      % Leave only elements selected
>     .unselect_annotations
>     .unselect_protractors
>     .unselect_rays
>     .unselect_tapemeasures
>     .unselect_trails
```

.unselect_object

.unselect_object *handle*

If the object whose object handle is given is selected then this will unselect it. Use in same manner as **.select_object**.

.update_on_element_drag, .update_on_ray_drag

.update_on_element_drag [*flag*]
.update_on_ray_drag [*flag*]

Same functions as available with the menu items **Options -- Update on Element Drag** and **Options -- Update on Ray Drag**.

If *flag* is 0 then unchecks the option.
If *flag* is non-zero then checks the option.
if *flag* is omitted then toggles the state of the option.

The updated state of the option is placed in the accumulator.

.userinput

See the section "**The script calculator**" for info on this keyword.

.user_click

.user_click [*LABEL*]

Gives control to the user for one click. When the user clicks on the primary mouse button control is returned to the script. If the user clicks outside the limits set by **.mouse_limits** and the *LABEL* argument is given then the click has no effect and execution is transferred to the line following :*LABEL*. If the *LABEL* argument is not given then the click is accepted regardless of whether it lies within the **.mouse_limits** region or not. Unlike **.allow_drag**, the user's click is fully functional and can select objects or complete a drag operation.

Example:

```
        % Create a surface to work on
        .file_new
        .create_surface
        .fc 0 0
        .arcseg
        .chord_tan
        .fc 0 100
        .fc 50 100
        .end_element
        .=> ARC
        % Create a rectangle to show mouse limits in this example
        .create_shape
        .fc 10 50
        .fc 100 50
        .fc 100 100
        .fc 10 100
        .fc 10 50
        .end_element
        .select_object ARC
        .mouse_limits 10 50 100 100
        .fc 50 100
        % Let the user drag and decide where to stop
        :DOCLICK
        .new_text
        Drag the tangent control point now to set the shape of the arc.


        Click somewhere in the rectangular box when you have finished.
        .fast_move 30 70
>       .user_click NOGOOD
        .new_text
        You clicked at coordinates:
        .printf %g lastclick.x
        ,
        .printf %g lastclick.y

        .pause
        .close_script
        :NOGOOD
        .new_text
        You did not click within the rectangular box.


        Do you really want to click outside the box?
        .yes_no DOCLICK
        .new_text
        Go ahead then.
        .mouse_limits -10000 -10000 10000 10000
>       .user_click
        .close_script
```

See also: .allow_drag, .mouse_limits and .user_control

.user_control

Gives the user full control of Raytrace. To return to the script the user must click on the Continue button in the script dialog box. Be aware that the user can do anything here and may not follow any direction exactly. If you want the ray diagram to be in a particular state when the user is finished then you should **.save** the ray diagram before releasing control and **.open** it when the user has finished; you can then modify the ray diagram as expected. Of course if the user modifies and explicitly saves the ray diagram with the same name you choose to use then they have defeated you!

Example:

```
        % Create a simple diagram for this example
        .file_new
        .painting 0
        .quiet_snap 1
        .enable_undo 0
        % First create a simple lens
        .create_region
```

```
        .fc 0 -50
        .arcseg
        .chord_tan
        .fc 0 50
        .fc 50 50
        .fc 0 -50
        .fc -50 -50
        .end_element
        .=> LENS
        % And a source linked to the lens
        .source_ray_count 6
        .point_source
        .fc -100 0
        .end_snap
        .fc 0 -50
        .end_snap
        .fc 0 50
        .=> SRC
        % Show the result
        .painting 1
        .new_text
        In this example a source has been linked to the lens using
        snaps. When the lens is moved the source is always incident
        upon it.
        .pause
        .unselect_all
        % Move the lens about to demonstrate the linking
        .select_object LENS
        .object_coordinates LENS 1 middle
        .click X Y
        .pause 500
        .move_rel 100 50
        .pause 500
        .move_rel 0 -100
        .move X Y
        .pause 300
        .click X Y
        .unselect_all
        % Delete the source and make some annotations to guide user
        .clear_rays
        .annotation Centre here
        .fc -100 0
        .fast_click_rel -20 20
        .annotation Ap'ture 1 - end snap here
        .end_snap
        .fc 0 -50
        .fast_click_rel 20 -20
        .annotation Ap'ture 2 - end snap here
        .end_snap
        .fc 0 50
        .fast_click_rel 20 20
        .quiet_snap 0
        .enable_undo 1
        % Save the diagram then let the user have a go.
        .save temp123.ray
        :USERTRY
        % Give some explicit directions
        .new_text
        You now have control of Raytrace.


        Use the menu item Create -- Point source and try linking the
        source to the lens using end snaps. If you succeed then when
        you drag the lens about the source should always be incident
        upon it.


        Click on Continue when you have finished.
        % Let the user try
>       .user_control
        .new_text
        Did you succeed? Click on No to try again.
        .yes_no PROCEED RETRY
        :RETRY
```

```
.open temp123.ray
.goto USERTRY
:PROCEED
.open temp123.ray
% You could continue on with something else now.
.close_script
```

See also: .allow_drag and .user_click

.vertex_snap

See **.centre_snap**

.wavelengths

.wavelengths *R G B option*

Sets the wavelengths of the three colours red, green and blue in that order. Values are specified in nanometres. If *option* is is zero then leaves refractive indices of existing elements alone. If *option* is non-zero then changes refractive indices to the correct values for the new wavelengths.

No checking is done on sensible values for the wavelengths!

Example:

```
% Set specific wavelengths and modify all elements
.wavelengths 450 560 700 1
```

.white_ray

Sets the colour of the selected rays in the sequence red, green, blue. Can be used to create a "white ray".

Example:

```
% Create a prism
.file_new
.material light flint
.create_region
.fc -80 0
.fc 80 0
.fc 0 100
.end_element
.=> PRISM
% Create a source hitting the prism
.source_ray_count 3
.point_source
.fc -160 -47
.mid_snap
.object_coordinates PRISM 3 middle
.fc X Y
.mid_snap
.fc X Y
% Make the source a white ray
.all_rays
>     .white_ray
.unselect_all
```

.yes_no

.yes_no *YES_LABEL NO_LABEL*
.yes_no *NO_LABEL*

Presents the user with two buttons labelled Yes and No. Transfers execution depending upon which button is clicked. If two arguments are given the execution is transferred to the line following the appropriate label. If only one argument is given and the Yes button is clicked then execution passes to the next line.

Example:

```
:LOOP
% This section will be repeated until user likes the result
```

```
      ...
      :TEST
      .new_text
      Are you happy with the result?
>     .yes_no LOOP
      % Continue here if Yes is clicked
```

See also: .choose, .goto, .if==

.zoom_extents, .zoom_in, .zoom_out, .zoom_previous, .zoom_reset

Same functions as available with the Zoom menu. **.zoom_in** and **.zoom_out** must be followed by appropriate click commands to specify the zoom range.

See also: .pan_left

.10^

See **.exp**

.=>

Store the contents of the accumulator in a script variable. Creates the variable if necessary.

If the available variable space is full then the error message "Too many script variables defined" is reported. Make sure you clear any unwanted variables.

Example:

```
      .123.456
      .=> FRED
```

See also: .clear_var

.+, .-, .*, ./, .1/, .+/-, .^2

.+ [*real*] add *real* to the accumulator
.- [*real*] subtract *real* from the accumulator
.* [*real*] multiply accumulator by *real*
./ [*real*] divide accumulator by *real*
.1/ [*real*] put the reciprocal of *real* in the accumulator
.+/- [*real*] change the sign of *real* and put result in accumulator
.^2 [*real*] square *real* and place result in accumulator

The result of these operations always ends up in the accumulator.

Both ./ and .1/ may report "Divide by zero" if *real* is zero.

Example:

```
      % Approximate specific function (y = x²/40 + 2x + 7)
      % in the range -50<=X<=50 by line segments.
      % Step X +5 each time.
      .file_new
      .-50
      .=> X
      .create_surface
      :LOOP
      .X
>     ./ 40
>     .+ 2
>     .* X
>     .+ 7
      .=> Y
      .fc X Y
      .X
>     .+ 5
      .=> X
      .if<= 50 LOOP
      .end_element
```

.--if!=0

.--if!=0 *variable LABEL*

Decrements the script variable called *variable* and if the result is not equal to zero branches execution to the line following :*LABEL*.

Example:

```
      % Approximate specific function (y = x^2/40 + 2x + 7)
      % starting at X = -50 stepping X +5 for 20 steps
      .file_new
      .-50
      .=> X
      .20
      .=> COUNT
      .create_surface
      :LOOP
      .X
      ./ 40
      .+ 2
      .* X
      .+ 7
      .=> Y
      .fc X Y
      .X
      .+ 5
      .=> X
>     .--if!=0 COUNT LOOP
      .end_element
```

See also: .if==