B

Appendix B LAYOUT FILE SYNTAX

Proof Animation stores drawings and definitions in layout files. Every animation requires a layout file. Layout files have an open, ASCII format that is documented in this appendix. For most applications, layout files are written by Proof; however, some applications may generate their own layout files under program control. The use of an open, ASCII format makes it straightforward to develop such programs. It also makes it possible for human beings to read a layout file, although this is rarely necessary. This appendix is provided primarily for people who are interested in using programs other than Proof, such as model generators, to write layout files.

Layout command words, like all Proof commands, may be any mixture of upper and lower case letters. By convention, Proof capitalizes the first letter of command keywords when it writes layout files.

Some examples of layout file commands are given in this appendix, but the best source of examples is layout files that have been written by Proof Animation. When you are running an animation, you can click on View, View Proof File as Text, Layout File and select a layout file to be viewed. Although some examples take up more than one line due to lack of space on the page, please note that each layout command must be written on a single line in an actual layout file.

Rules for Names

Classes, Objects, Bars, Plots and Messages defined in a layout file, require specification of an alphanumeric name. Names are case-sensitive; i.e., “joe” is different from “Joe.” Each name must start with an alphabetic character (A-Z or a-z). The remaining characters in a name can be letters, digits or underscore characters.

The “name spaces” for each of the five categories mentioned above are independent. So, for example, it is possible to have an Object named STATUS and a Bar named STATUS. This is acceptable, but not recommended.

A sixth type of name is the name of a View. The rules for View names are explained in the section on the "define view" command later in this appendix.

Layout File Order Requirements

Developers of layout files have considerable freedom in choosing the order in which layout commands are written; however, a few restrictions apply. The general nature of these restrictions is that for elements defined at one point in a layout file and referenced elsewhere, the definition must precede all references. The paragraphs which follow enumerate all such restrictions on command order. Note that if Proof is used to rewrite a layout file, the original order of commands in the file will not be preserved. (Proof writes layout files in its own, canonical order.

The color command establishes a "pen color" that is used for all subsequent Lines, Arcs, Text, Messages, Fills, Bars, Plots, and Class definitions until another color command is encountered. color commands can be placed anywhere in a layout file. If layout elements of the same color are grouped together, a slight savings can be obtained by reducing the number of color commands required. If a command that requires a pen color appears in the file with no preceding color command, F2 (red, by default) is assumed.

The arc, bar, fill, line, message, text, define color, define font, define plot, and define view commands can be placed anywhere in layout file. Note, however, that if layout elements overlap, order is significant. For example, if a red line and a blue line overlap, the line which appears later in the layout file will be visible in the area of overlap. For a more detailed discussion of this topic, please see Appendix D, which describes Proof's rendering algorithms.

CPO commands can be placed anywhere in a layout file except that they must follow the define class sections for the Classes used in the CPO commands. For this reason, it is easiest to put all CPO commands at the bottom of the layout file (before or after any define path sections).

define class sections must be terminated with an end statement. arc, fill, line, text, and message commands that are part of a define class section must appear between the define class command and its corresponding end statement. No commands other than arc, fill, line, text, message, and color may appear in this context. Arcs and Lines may not be numbered within a Class. The pen color that is currently in effect when the define class command is encountered remains in effect until a color command, if any, is encountered.

define path sections must follow all numbered line and/or arc commands that are referenced in their respective segment commands. A define path section must be terminated with an end statement. The only commands that can be placed between a define path command and its corresponding end statement are segment commands. segment commands cannot be used in any other context

Since a non-empty Path definition refers to preceding line and arc commands, the easiest approach for writing Path definitions is to place them at the bottom of a layout file (before or after any CPO commands).

An end statement (when not paired with define class or define path) must appear once, at the end of the file. Subsequent lines, if any, are ignored. Every layout file must contain an end statement.

Optional Layout File Commands

If a layout file contains no define view "(Home)" command, Proof Animation will perform a Zoom-to-Fit after the layout file is read.

The use of define color commands is optional. If other animations have been run prior to an animation whose layout file has no define color command for a given color, the most recently encountered define color command, if any, for the color will remain in effect. If no other animations have been previously run, a default color is used. For this reason, it is advisable to always include define color commands for all colors used in a layout file.

Errors in the Structure of a Layout File

A layout file written by Proof Animation should never produce errors when it is reopened. However, layout files created or edited "by hand" or created by a program other than Proof may contain errors. If so, error messages will appear during the initial processing of the layout file. (Layout files are processed in their entirety before an animation begins.) Messages indicate the line number in the layout file where the problem occurred. Severe errors elicit the message "*filename*.lay contains an error requiring manual repair." In such cases, the error(s) must be corrected before the animation can be run.

Layout File Command Descriptions

In the sections that follow, the collection of layout file commands is described in alphabetical order.

arc

The arc command describes a circular Arc. Its syntax is as follows:

arc [*#IDnumber*] *radius x y alpha omega*

When specified, *#IDnumber* indicates that an Arc may underlie one or more segments in one or more Path definitions. The "#" is required to signal the presence of an ID number. The ID number must be a positive integer and must be unique among those Lines and Arcs in the layout file that have ID numbers.

The remaining values, all numeric, define the geometry of the Arc. The Arc's radius is *radius*; its center is (*x, y*); its starting angle is *alpha*, and its ending angle is *omega*.

If *alpha* is greater than *omega*, the Arc is drawn clockwise. If *alpha* is less than *omega*, the Arc is drawn counterclockwise. The total angle spanned by the Arc is equal to the difference between *alpha* and *omega*. If *alpha* and *omega* are equal, no Arc is drawn. If *alpha* and *omega* are separated by 360° or more, the result is a full circle.

Several examples of arc commands follow:

```
arc #3 10 30 50 200 90

arc #4 10 30 50 -160 90

arc 6.5 4.12 8.01 0 360
```

The first arc command defines an Arc that may be used as a segment for one or more Paths (because it as an ID number, 3). It is centered at (30, 50) in the layout's coordinate system. The Arc has a radius of 10, and is drawn clockwise from 200 degrees to 90 degrees. The second arc command has the same starting and ending points as the first Arc, but is drawn in the opposite direction. It is a 250-degree arc instead of a 110-degree arc. The third arc command describes a full circle with a radius of 6.5, centered at (4.12, 8.01).

bar

The bar command defines a single rectangular bar graph element. For a group of bars clustered together, as in a histogram display, each bar must have a bar command in the layout file.

The syntax of the bar command is as follows:

bar *barname* [BGcolor *colorID*] *xmin xmax ymin ymax*

vrange *vmin vmax* hrange *hmin hmax*

The *barname* can be any name that conforms to the Proof naming rules given earlier in this appendix.

The optional BGcolor (for "background color") *colorID* determines the color in which Proof will "paint" for the empty part of the bar. The default is Backdrop. (The color of the "full" part of the bar is the layout file's current pen color, i.e., that of the most recent color command)

The *xmin, xmax, ymin,* and *ymax* values define the locations of the left, right, bottom, and top edges of the Bar in the layout's coordinate space.

The *vmin* and *vmax* values define the maximal vertical extent of the Bar, and the *hmin* and *hmax* values define the maximal horizontal extent of the Bar, all in arbitrary units independent of the Bar's location. Bars defined in Draw Mode have default values of zero for *vmin* and *hmin* and 100 for *vmax* and *hmax*. Vertical and horizontal ranges must always be supplied. If a Bar named B has a vertical range of 10…30, and the trace stream command "Set B top 20" is issued, the result will be a Bar of half of B's full height. B's top will be drawn at y = (*ymin* + *ymax*) / 2.

color

The color command is used to specify the color of succeeding elements in a layout file. The syntax of the color command is

color *colorID*

where *colorID* is F1 through F32, L1 through L32, or Backdrop.

Each color command sets the "pen color," which is used as the color of every succeeding element (including elements within Class definitions but excluding layout Objects, which have their own colors) until another color command is encountered.

When Proof writes a layout file, color statements are inserted into the layout file automatically as necessary so that all layout elements will have the proper color. Proof does not sort all drawn elements into color categories prior to writing the layout file. So, it is normal for the same color command to show up several times in a layout file written by Proof.

CPO

CPO stands for "Create and Place Object." When Proof writes a layout file, it uses the CPO command to specify the locations, colors, and rotations of Layout Objects.

The effect of a CPO command in a layout file is the same as using create and place…at command pair in the animation trace stream at time zero, except that Layout Objects are visible in Run Mode before an animation begins, and they are visible in (and can be manipulated from) Draw Mode.

The syntax of the CPO command is as follows:

CPO *classname objectname x y* [color *colorID*] [rotation *angle*]

classname is the name of a previously defined Object Class.

objectname is a unique, case sensitive name that follows the naming rules given in this appendix. Unlike trace file Objects, which can be identified by name or number, layout Objects must be named.

The *x* and *y* values are the layout coordinates at which the Object's hot point will be placed.

The color of the layout Object is determined by the *colorID* or, if the *colorID* is omitted, by the Class's color scheme. (Note: this is different from Lines, Arcs, etc., which use the "pen" color.) The *colorID* can specify any of the Proof Animation colors (F1-F32, L1-L32, or Backdrop). If you specify a color, the entire layout Object will appear in that one color.

Finally, layout Objects may be positioned rotated about their hot point. If a rotation is specified, the *angle*, which must be numeric is specified in degrees. Positive *angle* values specify counterclockwise rotation, and negative *angle* clockwise. If no rotation is specified, the Layout Object will appear in an unrotated orientation, exactly as defined in its Class.

The following is a typical CPO command:

```
CPO machine weld6 45 29.3 color F3 rotate 90
```

define class

The define class command is used to define an Object Classes that supplies the geometry and properties of the Objects in the animation. Each Class definition must conclude with an end statement.

The syntax for defining an Object Class is :

```
define class classname [ directional ] [ clearance fore aft ]
          [ speed s ] [ RGP -offset ]
          …
          geometry (line, arc, text, fill, message commands)
          …
end
```

classname is a case-sensitive name that meets the naming rules described in this appendix. Unlike Objects which are dynamically created from within a trace stream, Layout Objects must be identified by names, not numbers.

The directional keyword specifies that Objects based on this Class will start out as directional Objects. (Directional Objects are able to "point" in the direction they are moving, and can assume the orientation of the destination Object when place…in is used.) If directional is omitted, Objects based on this Class will start out non-directional.

The clearance specifications determine how far apart, in linear units of the layout coordinate system, Objects based on this Class should come to rest on an accumulating Path. The *fore* value specifies how much space to leave in front of the Object's hot point, and the *aft* value specifies how much space to leave behind the hot point. If clearance is omitted, the fore and aft clearances of Objects based on this Class will initially be zero. Zero clearance implies that Objects will come to rest on top of one another.

The speed *s* determines the speed an Object based on this Class will use to travel on a Path. If no speed is given, the Object will travel at the Path speed (unless that is overridden by an individual Object speed).

The RGP *offset* specifies the second point of attachment when directional Objects are traveling on Paths. The offset is always negative and is measured from the hot point (0,0). If no RGP offset is specified, the Object has a single point of attachment specified by the hot point.

Here is an example of an Object Class definition, followed by a figure which illustrates the Class. This Object Class is directional, has zero clearances, and uses the Path speed.

```
define class FullAGV directional RGP -1.5

line -4.5 -2 1.5 -2

line 1.5 -2 1.5 2

line 1.5 2 -4.5 2

line -4.5 2 -4.5 -2

fill -1 -1

end
```

define color

Proof Animation supports 65 colors: L1-L32, F1-F32, and Backdrop. When Proof writes a layout file, it writes define color commands for all 65 colors. Any customizations made to color definitions via Proof's Setup menu are incorporated. If you produce your own layouts directly, you should include definitions for all colors used. (When a layout file is loaded, colors for which there is no set color command retain their current color. If a previously loaded layout has changed a color from Proof's default color scheme, you will see the altered color.)

The syntax of the define color command is as follows:

define color *ColorID redvalue greenvalue bluevalue*

ColorID must be L1-L32, F1-F32, or Backdrop. Color names and IDs are not case sensitive. *redvalue*, *greenvalue*, and *bluevalue* are the red, green, and blue numerical components of the color. These components can be any real number from 0.0 through 1.0.

The following example

```
define color F5 1.0 0.0 1.0
```

defines color F5, a color that by default has a pink appearance, to appear purple, by adding more blue to the default F5.

Although Proof imposes no restrictions on which colors can be used in which contexts, color selection is very important when objects pass over layout elements and when objects collide with one another. In such cases, conflicts are resolved by color priority. The higher a color's number, the higher its priority. Foreground colors (F1-F32) have higher priority than layout colors (L1-L32), and the Backdrop color has the lowest priority of all. For a detailed discussion of color priorities, please refer to the description of Proof's rendering algorithms in Appendix D.

define font

Each font used in a layout must be assigned a font number. Fonts used in text and message command are specified by giving their font number. Proof has two predefined fonts. Font number zero is a fixed-pitch Courier font, and font number 1 is an Arial font. Any additional fonts used must be defined in a define font command. The syntax of the define font command is as follows:

define font *fontno* *"Font Name"*

The following example defines font number 2 to be Times New Roman:

define font 2 "Times New Roman"

Note that Proof supports only TrueType™ fonts. The fonts available for use in a layout vary widely from one computer to the next. Installation of printer drivers or other software packages may install fonts on a given computer that are not widely available elsewhere. When Proof encounters a define font command for a font that is unsupported on the current computer, it renders any text from that font using the Arial font. However, Proof consistently keeps track of the font's usage, so if the same layout file is used on a different computer which *does* support the given font, the font will once again be rendered as intended. In other words, font selection information is not lost. Fonts which are defined in define font commands, but are unsupported on the current computer are prefixed with an "*" in font selection menus.

define path

The define path command is used to define the properties and components (segments) of a Path. A Path definition comprises a define path command, zero or more segment commands, and a terminating end command. The syntax of define path is as follows:

define path *pathname* [accumulating | circular] [label *x y*] [lag *delay*]

[leapok] [[speed *s*] | [time *t*]]

    [segment #*source x1 y1 x2 y2 length* [cw | ccw]]

    …

    [segment #*source x1 y1 x2 y2 length* [cw | ccw]]

end

pathname is a unique alphanumeric string that follows the naming rules defined in this appendix.

If accumulating is specified, then the Path is an accumulating Path. Objects moving along the Path will queue up visually when they stop at the end of the Path (provided the Objects have a nonzero clearance). When the first Object subsequently moves off the Path, the entire visual queue will move forward smoothly.

If circular is specified, the Path is a circular Path. Objects reaching the end of the Path immediately jump to the beginning of the Path (which may or may not be contiguous with the end) and continue moving. Objects circulate on a circular Path indefinitely until they are placed elsewhere or destroyed.

A Path cannot be both accumulating and circular.

In Path Mode, the name of every Path is displayed adjacent to the beginning of the first segment; however, the name can be dragged to a more convenient location. The optional label *x y* in the define path command specifies where the label will be located when the file is next opened. Proof always records the label location for each Path when the it writes a layout file.

If lag *delay* is specified for an accumulating Path, each Object that incurs blockage on the Path will wait *delay* animation clock units after the Object immediately in front of it resumes moving before it starts moving itself.

If leapok is specified, leapfrog and encroachment errors will be ignored for a Path when the animation runs. A leapfrog error occurs when an Object other than the first is removed from an accumulating Path. An encroachment error occurs when an Object is placed on a Path at an offset which places it too close to another Object, in violation of the fore and aft clearances specified for the two Objects. Proof Animation uses the leapok keyword to distinguish Paths for which validation of place on commands has been turned off in Path Mode.

The optional speed *s* determines the initial Path speed in linear coordinate units per animated time unit. Objects placed on the Path will move at this speed (unless they are using a Class speed or an Object speed, or the Path speed is overridden in the trace file).

If time *t* is specified, the Path speed is calculated by dividing the length of the Path (determined by the segment geometry) by the value of *t*.

If neither a time nor a speed is specified, Path speed is assumed to be zero.

The syntax of the segment command is described later in this appendix.

define plot

The define plot command is used to define the dimensions and axes of a Plot. The syntax of the define plot command, which must be contained in a single line of a layout file, is as follows:

```
define plot  plotname  left x  right x  bottom y  top y
  vrange  vmin  vmax  [ unlabeled ]
  hrange  hmin  hmax  [ unlabeled ]
  data  colorID  [ BG  colorID ]  [ vgrid  colorID ]  [ hgrid  colorID ]
  [ xticks  deltax  ]  [ yticks  deltay ]  [ labelheight  height ]
```

plotname is a unique alphanumeric string that follows the naming rules defined in this appendix.

The *left x, right x, bottom y,* and *top y* values specify the Plot's physical location in the layout.

vrange *vmin vmax* and hrange *hmin hmax* specify the vertical and horizontal ranges, respectively, of data values to be plotted.. Data values are independent from the layout's coordinate space and the physical location of the Plot within the layout. For example, you might choose to define a y-axis that goes from 0 to 1000 (*vmin* = 0, *vmax* = 1000). Vertical and horizontal axes can each be unlabeled. If unlabeled is omitted, numeric values are drawn at each tick mark along an axis.

The data *colorID* specifies the color in which the data values are plotted. BG *colorID* BG specifies the background color on which the Plot is drawn. If a data in the Plot is changed or erased, the old segment is redrawn in the color specified after the BG in the define plot statement. It is important to set this color correctly if you will be changing the data within the Plot during the course of an animation. If no BG *colorID* is specified, the default is Backdrop.

If visible vertical and/or horizontal grids are desired, the colors of the grids can be specified using vgrid *colorID* and hgrid *colorID* , respectively. The vertical grid is drawn from top to bottom at each tick mark of the x-axis. The horizontal grid is drawn from left to right at each tick mark of the y-axis. The horizontal and vertical grids may be different colors within the same Plot. If neither vgrid nor hgrid is specified, no grid is drawn.

The spacing of tick marks along the Plot's x- and y-axes can be specified as xticks *deltax* and yticks *deltay,* respectively. If these optional values are omitted, Proof will calculate tick mark spacing automatically, as a function of viewing scale.

The height of the text used to label axes can be specified as labelheight *height* . If this optional value is omitted, Proof will calculate axis label text height automatically, as a function of viewing scale.

The following is a typical define plot command:

```
define plot MyPlot 75 100 25 40 vrange 0 10 hrange 0 1000
          data  F1 BG L7 vgrid L1 hgrid L1 xticks 100 yticks 1
```

define view

The define view command is used to define two system views ("(Home)" and "(Class)") and any user-defined views used in an animation. The syntax of the define view command is as follows:

define view "*viewname*" [*option*]...

The quotation marks shown above are required. Any combination of characters may be used in the *viewname*, including spaces. If the *viewname* is other than (Class) or (Home), the view is user-defined. A layout file may contain as many of these "named view" definitions as desired.

Command options are as described in the three following sections. All options used must be separated by one or more spaces and must be on the same line as define view.

System Views

Layouts written by Proof always include at least two define view commands, one for the "(Home)" view and one for the "(Class)" view. The parentheses distinguish these two view names from user-defined view names. Although they are optional, define view (Home) and define view (Class) should be included at the beginning of any layout file that is created outside Proof.

The (Home) view stored in the layout file is the view Proof Animation uses for all modes except Class Mode immediately after reading in the layout file. The (Class) view stored in the layout file is the view used the first time Class Mode is entered.

The Physical View

The principal function of any define view command is to define the physical parameters of the view – its scale, location, and orientation. The following six options determine the physical parameters of a view:

center *x y*

height *h*

width *w*

iso

ortho

rotation *angle*

If changes to these parameters are made from within Proof Animation, two steps are required to record the changes in the layout file: first, the desired view must be updated; then the layout must be saved.

Either height or width can be used to specify a view's scale. If both are specified, the first is ignored.

The orientation of a view can be iso (isometric) or ortho (orthographic) but not both. If both are specified, the first is ignored. If neither is specified, ortho is assumed.

If no rotation is specified, zero degrees is assumed.

The Grid Size

The current sizes of the "(Home)" and "(Class)" grids are recorded whenever Proof writes a layout file. The grid size option is used to automatically record these values. A grid size can also be specified for a user-defined view.

grid size *dotspacing linespacing*

If the specified combination grid line and dot spacing cannot be displayed clearly, it will be overridden.

Settings Options

Five options are valid in the "(Home)" view only. They control overall animation settings. From within Proof, changes to these settings are saved only if the "(Home)" view is updated and the layout is saved.

aspect *orthoratio* [*isoratio*]

grid on | off

Hertz *maxframerate*

speed *speed*

tolerance *maxerror*

The aspect option specifies the ratio of vertical to horizontal pixels in the display. The default *orthoratio*, which controls the appearance of all orthographic views, is 1.0. Raising this value will "stretch" the drawing vertically, which compensates for a display that "squishes." Orthographic aspect ratios are provided for compatibility with earlier versions of Proof. We discourage the use of orthographic aspect ratios other than 1.0. The default *isoratio*, which controls the appearance of all isometric views, is 0.5. (Isometric views are tilted and rotated views, sometimes referred to as 2½D.) If a greater or lesser tilt is desired for isometric views, a non-default *isoratio* can be specified.

The grid on | off option controls whether the grid is visible when an animation runs. The default is grid off.

The Hertz option is used to specify the maximum frame rate of an animation. If a large, complex animation is known to "drop down" to a lower frame rate during execution, this option can be used to prevent frame rate changes by starting with a frame rate known to be sustainable. (Changes in frame rate are disconcerting.) With modern PC hardware, the Hertz option is rarely required.

The speed option determines the speed at which the animation will initially run, expressed as animation time units per viewing second. The default is six time units per second.

The tolerance option is used to determine whether lines & arcs intersect with other lines & arcs. Gaps of size less than the tolerance value are treated as intersections. Drawings which are drawn "by eye" may contain unintentional gaps. This is frequently true of imported CAD drawings. Such drawings are designed for visual scrutiny. When subjected to algebraic scrutiny by Proof, such drawings can cause problems, because apparent intersections may not be recognized as true intersections. In Draw Mode and Class Mode, error tolerance affects the operation of Trim (but not Fill, which depends on painted pixels). In Path Mode, it affects the algorithm that finds the route from point A to point B using connected lines & arcs. Error tolerances have no effect on the execution of an animation.

The value of *maxerror* is normally a small positive number. The value of *maxerror* must be nonnegative. Error tolerances are specified in layout coordinate units.

If no error tolerance value is included in a layout file, Proof automatically calculates a tolerance value each time the view is changed. Proof saves a tolerance value in a layout only if you explicitly override the Proof default, via Proof's Setup menu. In general, letting Proof automatically calculate tolerance values is recommended.

Multiple Window Views

The screen can be split both horizontally and vertically into rectangular windows. If a user-defined view contains multiple windows, two additional types of commands must immediately follow the define view command.

For each windowed view, there is one active window. The default active window is the first window created within a view. In this case, bounds specifications must be added to the define view command:

define view *viewname* bounds *left% right% top% bottom%* [option]...

The *left%* and *right%* values are the percentages of the entire screen width that the window includes. The *top%* and *bottom%* values are the percentages of the entire screen height that the window includes. Percentage values must be in the 0…100 range.

If the active window is other than the first window of a view, a define active window command must be used for the active window:

define active window bounds *left% right% top% bottom%* [option]...

For each window of a windowed view other than the first window and other than the active window, the following syntax must be used:

define window bounds %lwidth %hwidth %lheight %hheight [option]...

Here is an example of a view with three windows:

```
define view “splitview’ bounds 0 33 0 100
  center 14.65 29,0833 width 29.25 grid size 1 10 ortho
define active window bounds 33 100 0 75
  center 54.5625 35.8333 width 50.625 grid size 1 10 iso
define window bounds 33 100 75 100
  center 54.5625 6.75 width 50.625 grid size 1 10 ortho
```

end

An end command is required at the end of a layout file. Any lines following the end command are ignored. Note that end commands are also used to terminate Class definitions (define class … end) and Path definitions (define path … end).

fill

The fill command fills a bounded region with pixels painted in the current pen color. The syntax of the fill command is as follows:

fill *x y*

The (*x*, *y*) coordinates define a Fill’s seed point, which must be within a region bounded by lines and arcs. When a Fill is drawn, the color of the seed point is sampled. In an expanding, outward search, pixels are set to the fill color until a pixel is encountered which does not match the color initially sampled at the seed point. If the region to be filled is not completely bounded, the fill will leak beyond the intended region. Because the algorithm is pixel-dependent, a complex shape that does not leak at one scale may leak at another scale.

The seed point should be specified near the middle of the region. If the seed point is too close to a boundary and the layout is subsequently zoomed in such that the seed point lies on the boundary, the fill will not be visible. In extreme cases, the seed point can end up just outside the boundary, causing undesirable effects similar to a leaky region.

When Proof draws a layout, it performs Fills after all Lines and Arcs have been drawn but before anything else. The locations of Fill commands in the layout file do not affect this algorithm.

line

The line command describes a line connecting two points. Its syntax is:

line [*#IDnumber*] *x1 y1 x2 y2*

When specified, *#IDnumber* indicates that a line may underlie one or more segments in one or more Path definitions. The "#" is required to signal the presence of an ID number. The ID number must be a positive integer and must be unique among those Lines and Arcs in the layout file that have ID numbers.

The values of *x1*, *y1*, *x2*, and *y2* can be any real numbers. One endpoint of the line is at (*x1*, *y1*) in the layout's coordinate space, and the other endpoint is at (*x2*, *y2*).

Two examples of the line command are:

line 10.5 -30 26 -10

line #6 23.54 17 12.12 70.6

message

The message command defines a one-line dynamic textual display. The message command is very similar to the text command except that the text command is used for defining textual displays that do not change during the course of an animation. Proof Messages have names, so they can be referenced an animation's trace stream in write commands.

The syntax of the message command is as follows:

message *messagename fontnum height* [angle a] [*justification*] *x y*

[BGcolor *colorID*] [*prototypestring*]

messagename specifies the name of the Message. The name is case sensitive and must follow the naming rules given in this appendix.

The *fontnum*, *height*, angle *a*, *justification*, and *x* and *y* values and operate exactly as they do for text commands. See the description of the text command below.

The BGcolor option determines the color used to repaint the old text, if any, as a means of erasing it prior to writing updated text during the animation. The default is Backdrop. If a Message is to appear on top of a solid-colored region (other than Backdrop), that color should be specified using BGcolor *colorID*.

The *prototypestring* is optional. It begins with the first nonblank character after the *y* value or the *colorID*, and ends at the end of the line. The *prototypestring* is used to identify the Message in Draw Mode. It never appears in Run Mode. If no *prototypestring* is specified, the *messagename* is used instead.

segment

segment commands define the Segments comprising a Path. segment commands are used only between a define path command and its terminating end command. The syntax of the segment command is as follows:

segment #*IDnumber x1 y1 x2 y2 length* [CW | CCW]

The *IDnumber* identifies a numbered Line or Arc used as the basis for the Path Segment. Numbered Lines and Arcs must appear earlier in the layout file than the Path definitions that refer to them.

A Segment may encompass all or part of a Line or Arc. The (*x1*, *y1*) and (*x2, y2*) values are the beginning and ending coordinates of the part of the Line or Arc that constitutes the Segment.

length is the linear travel distance of any Object moving on that Segment. For Paths defined within Proof Animation, *length* is calculated and recorded when the layout file is saved. In layout files created outside Proof Animation, the *length* must be calculated and included. If *length* is incorrect, and the Path has a Path speed (not a time), and Objects are moving on the Path at the Path speed, then the Objects will move at an incorrect speed on (and take an incorrect amount of time to traverse) the defective Segment.

For Arcs only, a CW (clockwise) or CCW (counter-clockwise) qualifier must be supplied to tell Proof Animation which way Objects should traverse this segment of the Arc. If the wrong keyword is used, the Objects will start at (*x1*, *y1*) and go the wrong way around the Arc's circle.

The following is an example of the segment command:

segment #27 10 10 90 10 80

text

The text command defines a static text string that will be displayed unchanged throughout an animation. The syntax of the text command is as follows:

text *fontnum height* [angle *a*] [*justification*] *x y* [*textstring*]

fontnum specifies the font to be used. Please refer to the description of the define font command.

height specifies the height of uppercase letters, expressed in layout coordinate units.

If angle *a* is specified, the text will appear rotated *a* degrees about the position of the text marker. Positive angles specify counterclockwise rotations, and negative angles clockwise. The default angle is zero degrees.

justification can be either LJ, center, RJ, or middle. This keyword determines how the text is positioned relative to the text marker (which is the x, y location of the text).

Keyword	Text Position
LJ (left justified)	The bottom left corner of the first character
RJ (right justified)	The bottom right corner of the last character
center	The bottom center of the string
middle	The middle (vertical and horizontal) of the string

If no justification is specified in the layout file, the fifth form, "left middle," is assumed. However, note that for Text drawn in Draw Mode, Text is left-justified by default.

The (*x*, *y*) coordinate specifies the location of the text marker. These values must be supplied.

The *textstring* can contain spaces, and need not be surrounded by quotes. The first character of the string is the first nonblank character following the *y* value.

Here are two examples of the text command:

```
text 1 3 RJ 30 10 Conveyor Plan One
```

```
text 5 10 angle 90 40 10 NUMBER 12
```

The first example places the text string "Conveyor Plan One" at position (30,10) in the layout's coordinate space. The string is right-justified, uses font number 1 (predefined as Arial), and is three units high.

The second example places the text string “NUMBER 12” at (40,10), at a 90 degree angle. The string’s height is 10 units. This string has left middle justification. The text is drawn in font number 5, which must be defined in a preceding define font command.