A

Appendix A
TRACE STREAM COMMAND SYNTAX

Trace Stream Command Set

A complete list of the commands that can be used in an animation trace stream is shown below. When Proof is run as a post-processor, trace stream commands are stored in a trace (.ATF or .PTF) file. When the library version of Proof is run concurrently with another application, trace stream commands are passed as character strings to library functions. The command set for both modes of operation is identical.

attach
create
destroy
detach
dt
end
move
place
play
plot
rotate
set bar
set class
set hertz
set [object]
set path
set rotation_resolution
set view
set viewing_speed
sound
syscall
time
write

General Rules for Trace stream Commands

Although some syntax examples in this reference appear on more than one line due to limited width on the page, note that each trace stream command in Proof must fit on one line in the trace stream.

Command names and command keywords shown in boldface are case-insensitive. For

example, “place 27 on MyPath”, “Place 27 on MyPath”, and “place 27 on MyPath” are equivalent In contrast to command names and keywords, user-defined names of Objects, Classes, Bar Graphs, Plots, and Messages referenced within commands are case-sensitive. For example, if MyPath is a Path name, “Place 27 on mypath” is invalid.

Names of Views may contain embedded blanks; however, names with embedded blanks must be enclosed within double quotes, e.g., “Loading Dock”.

Many animation trace stream commands include the optional keyword [object]. Object-related commands are so common that the optional keyword [object] is typically omitted in actual use. For example, “set 27 speed 5” is equivalent to “set object 27 speed 5”.

attach

The attach command is used to attach an Object to another Object in follow-the-leader fashion. The syntax of the attach command is as follows:

attach *objectID1* to *objectID2*

ObjectID1 identifies an Object that is to be attached to the Object identified by *objectID2*. The separation between *objectID1* and *objectID2* is the sum of *objectID1*’s fore clearance plus *O2*’s aft clearance, and is measured from hot point to hot point. Proof maintains a history of Path placements (if any) for the leading Object in a string of attached Objects. Each attached object follows the leader’s movement along each path traversed by the leader and duplicates the leader’s transfers from path to path (if any), at appropriate time lags.

Examples of attach are:

attach Boxcar to Engine
attach 4 to 6

create

The create command creates an individual Object from an Object Class. The syntax of the create command is as follows:

create *classname objectID*

Examples of create commands are shown below:

```
create AGV 306
create Digit1 101
create TYPE3SUB7 422
create car Herbie
```

The Object Class named *classname* must exist in the layout file.

The *objectID* can be either a name that meets the rules for Object names given above, or a positive integer. Objects whose *objectID* is specified as a name are assigned unique negative ID numbers inside Proof. This prohibits any Object from having both a name (which is converted into a negative number) and a number (which must be positive).

destroy

The destroy command erases an Object from the screen and releases the memory the Object consumes. The syntax of the destroy command is as follows:

```
destroy objectID
```

Examples of the destroy command are shown below:

```
destroy 422
destroy Herbie
```

Once an Object has been destroyed, its number or name may be reused for another Object.

detach

The detach command detaches two Objects that are currently attached to one another. The syntax of the detach command is as follows:

```
detach objectID1 from objectID2
```

ObjectID1 becomes a new "leader" for the string of Objects, if any, attached to it. Please refer to the description of the attach command. Examples of detach are shown below:

```
detach Boxcar from Engine
detach 4 from 6
```

dt

The dt command ("Delta Time") increments the animation clock by a specified amount. It is similar to the time command, except that the time command specifies the actual new value of the clock, rather than an increment to be added to the clock's current value. The syntax of the dt command is as follows:

```
dt increment
```

Most animations use time, not dt. One occasional use of dt is in hand-editing small trace streams for demonstration purposes.

Suppose the current animation clock is at 0.0, and the following commands are executed:

```
dt  5
dt 10
create WIDGET 1
```

When the "create WIDGET 1" command is executed, the animation clock will be at time 15 (0 + 5 + 10). This is equivalent to:

```
time  5
time  15
create WIDGET 1
```

dt and time are the only animation trace stream commands that can cause viewing time to elapse. All other commands are processed instantaneously until the next dt or time command is encountered

end

The end command terminates all processing of the animation trace stream. Proof Animation remains active but will not read any further in the trace stream. The syntax of the end command is as follows:

```
end
```

move

The move command causes an Object to move in a straight line from its current location to a specified destination.

The syntax of the move command is as follows:

move *objectID duration xdest ydest* [relative]
move *objectID* speed *s xdest ydest* [relative]

When the first form is used, *duration* implies how fast the Object will move (speed = move distance / duration). move uses the Object's exact current location - even if the Object is already moving or traveling on a Path - as the starting point for the move. For example,

move 1 30 20 45.5

will move Object number 1 from its current location to the location (20, 45.5) in the Proof Animation coordinate space, over 30 units of animation time.

Although this form of move requires a duration, the move command does not cause the clock to advance or any time to elapse. Only time (or dt) can cause time to elapse. Actual movement begins with the execution of the first time (or dt) statement following the move command.

If the second form is used, the duration of the move is implied (duration = move distance / speed).

move 3 speed 10 20 16.3

will move Object number 3 from its current location to (20,16.3) with a speed of 10 Proof Animation coordinate units per animation time units.

If the relative option is used in a move command, the *xdest* and *ydest* are interpreted as being relative to the Object's current position.

move 12 15 3 6 relative

Here, Object 12 will move 3 units in the *x* direction and 6 units in the *y* direction in 15 units of animation time. If Object 12 were at (10,10), it would end up at (13,16) after the move was completed.

Note that an Object that has been created but not placed has no current location. An Object must undergo at least one place before it can undergo a move.

place...at

The place...at command places an Object at a specified (x, y) location in the Proof Animation coordinate space. The syntax for place...at is as follows:

place *objectID* at *x y*

Here are some examples of place...at.

place 5 at 47 -10
place car at 30 20

An Object is always at rest (not moving) immediately after undergoing a place...at.

An Object will be invisible if it is placed off-screen, if it is completely BACKDROP-colored, or if it is placed at a location where its colors are all of lower priority than colors of other objects and/or layout elements. See Appendix D for a discussion of Object rendering.

place...in

The place...in command places the hot point of one Object at the same location as the hot point of another Object. The syntax for place...in is:

place objectID1 in objectID2

Here are two examples of place...in.

place 26 in WaitArea
place Carrier2 in Machine12

place...in leaves *objectID1* at rest, even if *objectID2* is moving. If *objectID2* is moving (via move, place...on, or place...on...at), the place...in command will place *objectID1* wherever *objectID2* is at the instant the place...in command is processed. Any subsequent relocation, motion, destruction, and/or rotation of *objectID2* has no effect on *objectID1*.

If *objectID1* is directional, it will assume the rotational orientation of *objectID2*. If *objectID1* is non-directional, it will appear in its normal orientation, without any rotation.

place...in is frequently used to place an Object into a layout Object (positioned at the desired point in the layout using Draw Mode).

place...on

The place...on command places an Object on a Path. If the object has been assigned a speed, and movement along the Path is possible, the Object begins moving at the given speed. If the Object has not been assigned a speed, but the Object's Class has a speed, and movement along the path is possible, the Object begins moving at the speed specified for its Class. If neither an Object speed nor a Class speed have been assigned, but the Path has been assigned a speed, and movement along the path is possible, the Object begins moving at the speed specified for the Path.

The syntax of place…on is:

place *objectID1* on *pathname* [squeeze] [at *offset*]
[at end]
[before *objectID2*]
[after *objectID2*]

Unless at is specified, the Object will be placed at the beginning of the first segment in the Path. For example,

```
place 112 on TRACK
```

will place Object number 112 on the Path named TRACK at the beginning of the first segment.

If at is included, the Object is placed *offset* units beyond the beginning of the first segment of the Path. This displacement is measured in linear units (in the Proof Animation coordinate system) beyond the beginning of the first segment of the Path. For example,

```
place 112 on TRACK at 10
```

will place Object number 112 on the Path named TRACK starting 10 linear units beyond the beginning of the first segment.

```
place 112 on TRACK squeeze at end
```

will place Object 112 with its hot point on the end of the Path. If the Path is accumulating, any Objects already at the end of the Path will be pushed back to make room for the incoming Object if squeeze is specified.

If the Path is accumulating and one or more Objects are already accumulated at the end, then "place...on...at end" without the squeeze keyword will cause an encroachment warning.

If the Path is accumulating and you do not use the squeeze keyword, you may get an Encroachment warning when using the before and after options if other Objects already reside before or after the specified Object on the Path.

An optional RGP clause can be appended to place...on commands for Objects that have a rear guide point (RGP). The following three forms of RGP clauses are available:

```
RGP on pathname2
RGP pulled
RGP follows
```

The first form specifies the Path onto which an Object's RGP is to be placed. An Object's hot point and RGP can be on two different Paths:

```
place 27 on Path1 RGP on Path2
```

The second form indicates that an Object's RGP is not placed on a Path, but rather, is pulled behind the Object's hot point. This form of motion can be used for automating wheeled vehicles, e.g. automobiles. When an automobile turns a corner, its front and rear wheels follow different routes. (The rear wheels always traverse a shorter distance than the front wheels.) For such usage, the Object used to represent an automobile would have its hot point located between its front wheels and its RGP would be located between its rear wheels. The following example illustrates the use of a pulled RGP:

```
place 99 on Roadway RGP pulled
```

The third form of RGP clause is used to revert to normal RGP behavior, in which Proof attempts to have an Object's RGP follow the same Path(s) as the Object's hot point:

```
place 99 on MyPath RGP follows
```

place…RGP

The place...RGP command is an abbreviated form of the place…on command which affects only an Object's RGP placement. Three forms of place…RGP are available:

```
place objectID RGP on pathname
place objectID RGP pulled
place objectID RGP follows
```

See the descriptions of the three forms of RGP clauses allowed in place…on commands,

described above.

play

The play command is used to play .WAV files. The play command can be used to add sound effects, music, or narration to an animation. Note that there is also a play command for use in presentation script (.PSF) files. The syntax of the play command is as follows:

play [asynchronous] *filename*

The specified *filename* must be a .WAV file. The asynchronous keyword indicates that the playing of *filename* is to be overlapped with ongoing execution of an animation. This option is almost always used in a trace stream. If the asynchronous option is not used, execution of the animation is suspended until *filename* has been played in its entirety.

The following example plays a file named tada.wav:

play asynch tada

Ongoing, asynchronous .WAV file playing can be terminated as follows:

play off

plot

The plot command is used to add data line segments to a Plot. The syntax of the plot command is as follows:

plot *plotname* [*#segmentID*] *x1 y1 x2 y2* [color *c*]

x1,y1 is the start point of the line segment and *x2,y2* is the endpoint. The coordinates are relative to each individual Plot.

If a color is specified, the data line will be drawn in the specified color instead of the default color defined when the Plot was created.

If a *segmentID* is specified, and that segment has already been plotted, the old segment is erased before the new segment is plotted. The old segment is erased by redrawing it in the Erase color defined for the Plot. The following are examples of the plot command:

plot ordernum 10 16.4 10 20.8
plot capacity #3 120 22 178.5 411 color F2

A Plot can be completely cleared using the following form of the Plot command:

plot *plotname* clear

clear erases all existing line segments by being rewriting them in the Plot's Erase color.

rotate

The rotate command rotates an Object around its hot point The rotate command takes one of two forms. The first form, which causes an Object to rotate to a specific angle, is:

rotate *objectID* [to] *angle* [time *duration*] | [speed *s*] [step *degrees*]

The *angle* is specified in degrees. If to is specified, then *angle* is the final angle. If this angle is greater than the Object's current angle, rotation is counterclockwise; otherwise, rotation is clockwise.

If to is omitted, then *angle* is a relative angle that is added to the Object's current rotational angle to calculate the final angle. If the relative angle is positive, rotation is counterclockwise.

A *duration* may be specified for a rotation. Note that this *duration* does not cause time to elapse in the animation. Only a time (or dt) command can do that.

A speed (*s*) may be specified for a rotation, in degrees per time unit. The sign of *s* is ignored unless no *angle* is specified (see below).

Given a final angle and a duration or speed, Proof Animation will rotate the Object at a constant rate (in degrees per time unit) until the Object reaches the final angle. If neither a duration nor a speed is specified, the rotation is performed instantaneously.

The second form of rotate causes an Object to rotate indefinitely without stopping. The syntax of this form is:

rotate *objectID* speed *s* [step *degrees*]

In this case the sign of *s* determines the direction of rotation. A positive *s* causes counterclockwise rotation.

With either form of rotate, smoothness of rotational motion is controlled by the step size. By default, the step size is 30 degrees. To make the rotation appear smoother, specify a smaller step size, as in the example below.

rotate robot 45 time 36 step 5

Using a small step size makes Proof work harder, because the Object needs to be rendered more frequently and additional space must be allocated in which to store the various orientations of the Object.

Note that Object rotations are affected by the Angular Resolution option of Proof's Setup menu. If Angular Resolution is set to 3 degrees, using a step size of 1 degree will not achieve the desired results.

Object rotations can be combined with motion along a Path or with linear motion resulting from a move command.

An Object's angular orientation on the screen is the sum of three components, the Object's "directional angle," the Object's "rotational" angle, and any viewing rotation applied to the entire layout in which the object appears. A directional Object's "directional angle" is determined by motion along a Path or motion due to execution of a move command. Directional Objects are always pointed in the direction in which they move. Non-directional Objects always have a directional angle of zero. An object's rotational angle is added to its directional angle. An Object's rotational angle can be set in three ways. The rotate command modifies an Object's rotational angle. A place...in command adjusts a directional Object's rotational angle to match the orientation of the Object into which the given Object is placed. A Layout Object can be defined with an initial rotation.

Suppose, for example, that a directional Object is moving along a Path Segment which is oriented at a 30-degree angle. If the Object is rotated by 90 degrees, the object will appear at a 120-degree angle, relative to the orientation in which the Object's Class was drawn.

set bar

Every Bar in a layout has a top, bottom, left and right location, expressed in the coordinate system of the layout. Bars cannot be rotated, so only four values are required. The data values a bar represents are also specified as top, bottom, left, and right values, but the two sets of four values are independent. This allows you to move and/or resize a Bar without having to change the data values a Bar represents.

The set bar command controls the current size and/or color of a Bar. Any combination of the four edges of a Bar can be modified in a single set bar command. The syntax of the set bar command is as follows:

set bar *barname* [*direction* [*level*] [rate *r*]]... [color *c*]

where *direction* is top, bottom, left, or right

barname is the name of the Bar, as defined in Draw Mode.

The specifications for a Bar's top, bottom, left, right, and color can be given in any order and combination. Top, bottom, left, and right specifications can include a level, a rate, or both. If a rate is specified, it is interpreted as data units per unit of animation time (*not* viewing time). For example, if a Bar's height ranges from zero to 100, and the the animation time unit is minutes, a "top" rate of 1.0 implies a 1% increase in height per animated minute.

The Bar's color (*c*) can be any of Proof's color names: backdrop, L1 through L32, or F1 through F32. When the very first set bar command is executed for a given Bar, any edges not specified in the command are set to the data values specified when the Bar was defined. Each set bar command after the first for a given Bar affects only the specified edge(s). The other edge(s) are left unchanged.

For example, if MyBar is defined as representing data values of bottom=0, top=100, left=0, right=200, and the first set bar command for MyBar is

```
set MyBar top 50
```

MyBar will be drawn as top=50, bottom=0, left=0, right=200. The result will be a half-height, full-width Bar.

Here are some additional examples of set bar:

```
set bar Hist1 top 3
set bar Volume top 70.7 right 70.7
set bar CPUutil right .953 color F2
set bar TankContents top 20 rate 3.7
set bar squeeze left rate 5 right rate -5
```

Several special cases of edge values are recognized. If a top value is less than or equal to the Bar's current bottom value, or a left value is less than or equal to the Bar's current right value, the bar will be invisible. If a top value is less than the minimum bottom data value defined for the Bar; if a bottom value is greater than the maximum top value defined for the Bar; if a left value is greater than the maximum right value defined for the Bar; or if a right value is less than the minimum left value defined for the Bar, the Bar will be invisible.

If you wish to set only the top value of a Bar (a common special case), you may omit the TOP keyword:

set bar Hist1 3

set class...clearance

The set class...clearance command changes the default clearances of a specified Class. Existing Objects of the Class are not affected. New Objects created after the set class...clearance command has been executed will inherit the new clearances. The syntax of the set class…clearance command is as follows:

set class *classname* clearance *fore aft*

For example:

set class MEDCART clearance 3 9

set class...directional
set class...nondirectional

The directional/non-directional property of a class can be changed using the following syntax:

set class *classname* directional

or

set class *classname* nondirectional

Already existing Objects of the specified Class are not affected. New Objects created after set class...directional or set class...nondirectional has been executed will inherit the new directionality from the Class. Examples are as follows:

set class BUS directional
set class AGV nondirectional

set class...RGP

The set class...RGP command changes the default RGP offset of the specified Class. Existing Objects of the Class are not affected. New Objects created after the set class...RGP has been executed will inherit the new RGP offset location. The syntax is:

set class *classname* RGP *offset*

The offset is always specified as a negative number and represents the units behind the hot point

(0,0) of a second point of attachment for directional Objects when traveling on Paths. For example:

set class Bus RGP -2

set class...speed

The set class...speed command changes the default speed of the specified Class. Existing Objects that are members of the Class are not affected. New Objects created after the set class...speed has been executed will inherit the new Class speeds unless individual Object speeds have been set. Note that individual Object speeds override Class speed. The syntax of the set class…speed command is as follows:

set class *classname* speed *s*

Speed is measured in Proof Animation coordinate units divided by animation time units. For example

set class widget speed 12.5

redefines the default speed of Objects of Class widget to be 12.5 units of distance per unit time.

set object...class

The set object...class command changes the Object Class on which a particular Object is based. This command is useful for changing the shape of an Object during an animation. For example, the representation of a machine could change from idle to "sparks flying" or that of a truck from full to empty.

When the set object...class command is used, the properties of the new Class are adopted by the Object. It may be desirable to adjust certain properties, such as color, after the set object...class command. Logically, set object x class is the same as destroy x followed by create x, except that Object x's location, motion status, and attachment to other objects are preserved.

The syntax of set object...class is as follows:

set [object] *objectID* class *classname*

Examples of set object…class are:

set 23 class FullAGV
set widget class FINISHED

set object...clearance

The set object...clearance command changes the clearances of an Object. This command is used if Objects need to accumulate on an accumulating Path with clearances which differ from those defined in the Object's Class.

The syntax for set object...clearance is as follows:

set [object] *objectID* clearance *fore aft*

For example,

set 53 clearance 6 7

changes the clearances for Object 53 from their previous settings to a *fore* clearance of six linear units and an *aft* clearance of seven linear units. The new values will persist unless overridden by a subsequent set object...clearance or set object...class command.

set object...color

The set object...color command immediately changes the color of an Object. The initial color of an Object is determined when the Object Class is defined. The color of an individual Object can be manipulated via set object...color at any time after the Object is created, before and/or after it appears on the screen. The syntax of the set object...color command is as follows:

set [object] *objectID* color *colorID*

The *colorID* can be backdrop, L1 through L32, F1 through F32, or class. If *colorID* is specified as class, the Object is changed to the default color scheme of its current Class as defined in Class Mode. The following are examples of set object…color:

set Drill color RED
set 102 color F6
set 1212 color class

set object...directional

set object...nondirectional

The directional/non-directional property of an Object can be changed using the following syntax:

set [object] *objectID* directional

or

set [object] *objectID* nondirectional

These commands are useful for Objects that are sometimes but not always directional (e.g. vehicles that sometimes "crab" - move sideways - or cases on a conveyor that sometimes navigate curves but other times are pushed sideways onto or off of the conveyor).

When an Object's setting is changed to non-directional as in the example below, its directional angle is "frozen" in its current state.

set crane nondirectional

When a non-directional Object is set to directional as in the example below, its directional angle is reset to zero. If the Object is moving on a Path, it may have an incorrect forward orientation until it moves onto the next Line or Arc in the Path.

set 77 directional

set object...speed

The set object...speed command alters the speed of an Object moving on a Path. If the specified Object is not currently on a Path, its new speed will not take effect until it is placed on a Path. An Object can derive its speed along a Path from three sources. Issuing a set object… speed command is the highest priority form of speed specification for any given Object. Such commands overrides the other two forms of speed specification. If no set object…speed command is issued, but an Object is a member of an Object Class for which a default speed has been defined, the Object's Class speed overrides the speeds, if any, specified for Paths on which the Object is placed. If an Object has neither an individually-specified speed nor a Class speed, .and a Path on which it is placed does have a speed, the Object's speed is set to the Path's speed. If an Object is placed on a Path and none of the three forms of speed specification have been used, the Object will remain stationary. The syntax of the set object…speed command is as follows:

set [object] *objectID* speed [*speed* | path | class]

The set object...speed command, if used with a numeric *speed* argument, causes the Object to move at *speed* on Paths *and* sets the source of that Object's speed to be the Object speed.

An example of set object...speed with a numeric *speed* is:

```
set Car12 speed 30
```

This command gives Car12 a speed of 30 linear units per animated time unit for travel on Paths.

When an Object's speed is specified in this way, the Object will move at the *speed* on all subsequent Paths until it undergoes another set object...speed (or a set object...travel command, described below). In the example above, Car12 is now considered to have its own Object speed, so the value of 30 will be used as the speed of Car12 on all Paths until overridden.

The path or class keywords may be used in a set object...speed command, to specify the source of an Object's speed as the Path speed or Class speed, respectively. For example:

```
set Car12 speed path
```

This command establishes (or re-establishes) that the source of Object Car12's speed is the speed of whatever Path Object Car12 is on at the present time or is placed upon in the future.

set object...travel

The set object...travel command has the same effects as set object...speed has when used with a numeric speed. The Object's speed is calculated by dividing the remaining distance along its current Path by the specified travel time, and the source of the Object's speed for motion along the current and subsequent Paths becomes this calculated speed value.

The syntax is:

```
set [ object ] objectID travel traveltime
```

The *traveltime* must be greater than zero.

Here is an example of set object...travel:

```
set 125 travel .25
```

This will cause Proof Animation to divide the remaining length of the Path by .25, and then move Object 125 at the resulting speed.

Because set object...travel establishes a speed that is based on the current Path but has a permanent effect on the Object's speed, the Object will normally have to undergo another set object...speed or set object...travel before getting on another Path.

set object...userinfo

The set object...userinfo command attaches descriptive text to an Object. This text appears in a pop-up dialog box when, and only when, the Object is selected by right-clicking on it. The descriptive text begins with the second character after the userinfo keyword.

The syntax is:

set [object] *objectID* userinfo *descriptive_text*

Here is an example of set object...userinfo:

set 125 userinfo United Airlines 327 DCA to ORD

set object...usertitle

The set object...usertitle command is used in conjunction with the set object...userinfo command. This command allows you to specify a title for the pop-up dialog box that appears when an Object for which auserinfo has been specified is selected by right-clicking on it. The title text begins with the second character after the usertitle keyword.

The syntax is:

set [object] *objectID* usertitle *title_text*

Here is an example of set object...usertitle:

set 125 usertitle UAL 327

set path...lag

The set path…lag command changes the lag time of Objects traveling on an accumulating Path. When Objects accumulate at the end of an accumulating Path, and the first Object is removed from the Path, the second object, if any, on the Path waits until the Path's lag time has elapsed before moving. Each object behind the second, if any, undergoes a similar delay. Lag times allow approximation of the way cars resume motion when a traffic light turns green. The

syntax of set path…lag is as follows:

set path *pathname* lag *lagtime*

For example

set path conveyor lag 3

establishes a 3 animation time unit delay before movement can resume following the removal of a downstream blocking condition.

set path...speed

The set path…speed command changes the speed of a Path. It can change the speed at which *all* Objects on the Path, if any, are moving. The syntax is:

set path *pathname* speed *speed*

For example,

set path subpath1 speed 0

will cause all Objects on the Path named "subpath1" to come to a stop.

When set path...speed is used, all Objects on this Path and future Objects that will be placed on this Path will move at the specified speed *assuming the individual Objects are deriving their speed from the Path speed* (see the discussion of sources of speed under set object...speed above). Objects that inherit Class speeds, or that get their own speed via set object...speed, will not automatically pick up the new Path speed unless set object...speed path is used for each such Object.

set path...travel

The set path…travel command provides an alternative way to specify the speed of a Path. The syntax of the set path…travel command is as follows:

set path *pathname* travel *traveltime*

The following example sets the speed of path subpath2 to its length divided by 12.5:

set path subpath2 travel 20.5

When set path...travel is used, all Objects on this Path and future Objects that will be placed on the Path will complete travel along the Path in the specified *traveltime*. The speed at which the Objects travel is derived from the Path length divided by the *traveltime*. The set path ...travel will only affect Objects traveling on the Path that do not have Class or Object speeds set.

set view

The set view command selects a named view during the execution of the animation trace stream. The syntax is:

set view *viewname*

The *viewname* may be any named view defined in the layout file, or the system views "(Home)" or "(Class)". View names are case sensitive, and in the case of the system views they must be referenced with quotes and parentheses as shown in the first two examples below. Names of named views need not be contained in quotes unless they contain embedded blanks (see the third and fourth examples below).

```
set view  "(Home)"
set view  "(Class)"
set view  shipping
set view  "shipping dock"
```

Changing the View from the trace stream can be disruptive to someone watching the animation. This technique should be used only for situations in which something takes place in the model that focuses attention on a particular part of the animation. Otherwise the person viewing the animation should be in control. (If the goal is to merely show a collection of different views, Presentation Mode should be used.)

set viewing_speed

The set viewing_speed command changes the viewing speed of an animation from within the trace stream. Viewing speeds are interpreted as animation clock ticks per viewing second. You control the meaning of an animation clock tick. For example, in an animation of a manufacturing system, one click of the animation clock might represent one second in the real system, while in a model of a telephone switching system, one clock tick might represent a millisecond in the real system.

If a simulated system is entering a period of little or no activity, set viewing_speed can be

used to temporarily increase viewing speed. For example, in an animation of a production line that runs at normal levels of activity for two shifts and at a minimal level of activity during the third shift, it might be helpful to speed up animation of the third shift and resume normal viewing speed when the first shift starts for the next day. Note that set viewing_speed commands will override any viewing speeds set via Proof's menus.. The syntax of set viewing_speed is as follows:

set viewing_speed *speed*

The *speed* specified must be a numeric constant greater than zero, e.g.,

set viewing_speed 10

sound

The sound command was used in earlier versions of Proof for playing simple tunes or adding alarm effects to an animation. Since Windows 95 and Windows 98 do not provide an interface to a PC's tone generator, the sound command no longer works. However, the play command provides the capability of playing .WAV files, which can contain complex music, sound effects, or narration.

syscall

The syscall command is used for invoking other programs from within Proof Animation. Before invoking a specified program, the Proof window is minimized and placed on the taskbar. After the specified program completes its execution, you must reactivate Proof by clicking on icon in the taskbar or by means of Alt-TAB task switching. The syntax of the syscall command is as follows:

syscall *program name* [*program arguments*]

For example,

syscall d:\slx\se d:\slx\myprog

runs D:\slx\se, passing it "d:\slx\myprog" as an argument.

time

The time command is used to advance the animation clock to a new value. Animation trace streams always contain time or dt commands. In the absence of time or dt commands, nothing

would happen on the screen. Use of the time command is far more common than use of the dt command. The syntax of the time command is as follows:

time [jump] *timevalue*

The *timevalue* must be nonnegative and must always be greater than or equal to the most recently specified *timevalue*. Both integer and floating point values are accepted, as seen in these examples:

time 5
time 10.367

In the following example, the JUMP option is used, and a fast forward is performed until time 106 is reached:

time jump 106

dt and time are the only animation trace stream commands that can cause viewing time to elapse. All other commands are processed instantaneously until the next dt or time command is encountered

write

The write command modifies the contents of Messages, which are textual annotations that can be dynamically modified during the course of an animation. Messages may defined for Object Classes. If Messages are defined for an Object Class, each Object created from the Class has its own unique copy of the message. Thus, annotations can be individualized for particular objects. The syntax of the write command is

write *messagename textstring*
write *messagename (objectID) textstring*

The first form is used to modify Messages in a layout, and the second form is used to modify Messages in Objects of a given Class. The *messagename* refers to the name of a Message that has been defined and positioned in the layout file or Object Class. The *messagename* is case sensitive.

The *objectID* refers to the Object in whose Class the Message is defined.

The *textstring* is the actual text that is to appear on the screen when the write command is executed during the animation.

Blank characters are allowed in the *textstring*. Proof Animation assumes that the string begins with the character that follows the first space after the *messagename*, and ends just before the carriage return/linefeed at the end of the line. The following example of the write command

```
write status The machine is down
```

will display the text "The machine is down" at the location of the Message named "status."

```
write BOX(20) 20
```

Will display "20" in Object 20's Message named *Box*. The Message named BOX must be defined in the Object Class of which Object 20 is currently a member.

When the write command is executed, the Message's current text, if any, is first erased by overwriting it in the Erase color defined for the Message.