5

Chapter Five
ANIMATION TRACE STREAMS

What is an Animation Trace Stream?

Every animation uses one animation trace stream ("trace stream" for short). Every trace stream is a sequence of *commands* (statements) that specifies dynamic events that take place on the screen. Everything that moves, changes, appears or disappears in the animation does so as the result of a command in the trace stream.

If you're creating a *post-processed* animation, the animation trace stream will be contained in an animation trace *file* ("trace file" for short). If you are building a *concurrent* animation, trace commands are passed from a running program to functions contained in a library version of Proof. Proof commands used for either form of animation are identical; hence, we will use the term trace stream to apply to either form, and will use the term trace file only when specifically discussing post-processed animation. Details on using the library versions of Proof are given in Chapter 18.

Proof reads the trace stream as an animation progresses, using time information contained in the trace stream to determine when to read and process the next command in the stream.

Structure of an Animation Trace Stream

The animation trace stream contains a sequential list of command lines. Each line contains a command that causes Proof to perform some action in the actual animation.

Proof reads each line only once. There is no pre-processing of the trace stream.

The Proof commands in an animation trace stream are all ASCII (human-readable text and numbers). The syntax is English-like and easy to read.

Comment lines may be embedded in the animation trace stream by beginning the line with an

asterisk (*).

An animation trace stream should be terminated by an end command. If you are running a non-Library version of Proof, and Proof encounters the end of a trace file with no end command, a warning message will be issued.

Creating and Editing an Animation Trace Stream

A trace stream is comprised of an ordered list of Proof commands. These commands are created by a model or program separate from Proof. Any language or modeling package to be used with Proof must include some form of output statement for formatting ASCII text and either writing it to a file or passing it to a library function. For example, in BASIC, a WRITE statement could be used; in C/C++, printf() could be used; and in GPSS/H, PUTPIC or BPUTPIC could be used. Chapter 8 contains detailed instructions for generating animation trace streams using various modeling tools and other programs. Some simulation software (such as Extend and SLX, at this writing) can be obtained with built-in capabilities that simplify the task of generating Proof trace streams. If this is not described in Chapter 8 for the software you use, contact your simulation software vendor or Wolverine for the latest information.

Because the commands are simple, few in number, and ASCII, it is easy to manually create or edit a small trace file using a text editor. You will do so in some of the early exercises in this book, using NotePad or WordPad (which both come with Windows) or whatever text editor you prefer. If you wish, you can also use a word processing program such as WordPerfect or Microsoft Word.

When using WordPad or any word processing program, be sure to save trace streams as "text document" or "ASCII" or "text only." (NotePad and programmers' text editors save as text by default.)

Once you begin using Proof for projects, you will never need or want to hand-edit your trace streams. Your model or other program will write the trace stream each time it runs.

Exercise 5-1: Exploring an Animation Trace File

This exercise explores a simple animation trace file. Start Proof and begin the exercise by choosing **File**, **Open Layout & Trace**, and selecting sample.lay from the exercise folder. Next, choose **View**, **View Proof File as Text**, **Trace File**. Proof will minimize its own window and launch a separate text editing session using another program.

The first time you use **View Proof File as Text**, you'll be prompted to enter a command that Proof will use to do the viewing. The default command is **start WordPad**, which launches

Microsoft's WordPad editor, which is provided free-of-charge in all versions of Windows. If you have a favorite editor, you can supply either the fully-qualified name of its executable file, e.g., **c:/editors/myeditor.exe**, or if the executable has been registered with Windows (as is typical), you can use a start command, e.g. **start myeditor**.

When you have concluded an editor session, you can exit your editor if you like, and then click on the Proof icon on the Windows task bar to resume your full-screen Proof session.

The contents of sample.atf are as follows.

```
Write Info Time 0

Time 25

Write Info Time 25

Time 30

Write Info Time 30

Time 35

Write Info Time 35

Time 50

Write Info Time 50

Write Info End

End
```

On the task bar choose **Run**, **Go** or click the **Run** button to view the sample animation and observe its behavior. What you'll see is a sequence of messages appearing on the screen at varying intervals. In the sections that follow, we'll ignore the inner workings of the write trace commands shown above, and focus our attention on the role of the time commands.

Simulation Time, Animation Time, and Real Time

To understand how time is handled by Proof, we must consider three clocks. First, the simulation model or program that creates a Proof trace stream has a clock. The time units of this *simulated time* clock vary from model to model. In a model of a factory, time units of minutes or seconds might be used, while in a model of a computer system, time units of microseconds or milliseconds might be used. Second, Proof has a clock. Unless you take the rare step of scaling all time values

in your simulation model before writing them to the trace stream, Proof's *animated time* clock will exactly parallel your simulation clock. Third, we must consider the clock on your wall (*real time*). Proof plays your animations at a specified ratio of animated time to real time.

How the Animation Clock Works

When the animation clock is moving, trace stream command processing is suspended. Conversely, during the brief intervals in which trace stream commands are executing, the animation clock is frozen at its current value. The transition between these two states is punctuated by time *commands.*

The syntax of the time command is:

time [**jump**] *timevalue*

The time command is used to advance the animation clock to the time specified by *timevalue*. Proof then executes the commands that follow this time command. When Proof encounters a time command in the trace stream, it first checks to be sure that the *timevalue* is greater than or equal to the current animated time. If not, the animation terminates with an error.

After verifying that the time command specifies a future time, Proof continues to display the animation as it progresses, *without reading any more lines from the trace stream*, until the animation time specified in the time command has been reached or exceeded. At that point Proof reads and processes the next line(s) in the animation trace stream. Lines are read and processed in this way until an end command is encountered.

If the **jump** option is used, a fast-forward is performed. The animation is suspended while Proof quickly processes the trace stream events until reaching a time command that is greater than or equal to the given time value. When completed, the screen is redrawn with everything up-to-date.

Figure 5-1 depicts Proof's processing of the time commands in sample.atf:

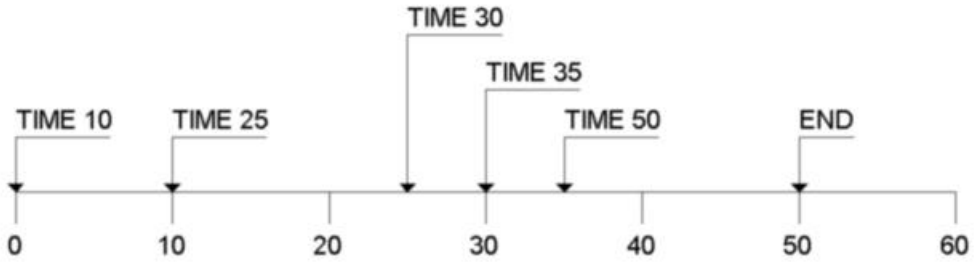


Figure 5-1. How time commands are read and processed

In sample.lay, a 5:1 ratio of animation time to real time is used. Thus, after the time 10 command is processed at time zero, no further commands are processed for two seconds of viewing time (10/5 = 2). When the time 25 command is processed at time 10, no further commands are processed for another three seconds ((25-10)/5 = 3). When the time 30 command is processed at time 25, command processing is suspended for one second. This pattern continues until the end command is read at time 50. At this time, all processing ceases and forward execution of the current animation stops.

Simultaneous and Near-simultaneous Trace Commands

As mentioned in Chapter 3 and earlier in this chapter, animated time always passes at a constant rate. As previously discussed, the ratio of this rate to real time is the *animation speed*.

How does Proof track real time? The "heartbeat" of every animation is the screen refresh rate. This is usually 60 or 70 cycles per second but can vary depending on your graphics hardware.

If two or more animation commands are processed during the same screen refresh cycle and their effects are conflicting, you will see only the result of the command that was executed after the other(s). In sample.atf at time 50, two commands update the same message area on the screen. Because of this, the text "Time 50" never appears on the screen; rather "END" appears. The same phenomenon can occur when two or more commands are executed at times that differ, but are so close together that they fall between consecutive screen updates.

Here is a concrete example explaining this phenomenon. Assume that an animation is running on hardware that operates at 70Hz (70 screen updates per viewing second). Further assume that an animation is running at a 10:1 ratio of animation time to viewing time (real time). The first screen update after time zero will occur after 1/70 second of real time, or 0.01428 elapsed seconds into the animation. Since a 10:1 viewing speed is used, elapsed time 0.01428 corresponds to an animation time of 0.1428. Now assume that the trace stream contains two commands to set an object's color. Assume that the first command, issued at time 0.03, sets the object's color to blue, and the second, issued at time 0.14, sets the object's color to red. Will the blue color ever be seen?

To understand the answer to this question, we must discuss how Proof updates the screen. Proof maintains two copies of the screen at all times – one that matches the visible screen image, and one that is hidden. All updates are applied to the hidden image, and then when it is time to update the screen, Proof tells the hardware to switch from the visible image to the hidden image, reversing the roles of the two images. Therefore the answer to the question posed above is no,

because both commands are applied to a hidden image in the same interval between screen updates. When the hidden image is made visible, the color of the Object will be red.

Debugging an Animation Trace Stream

Proof's **Debug Mode** allows you to step carefully through your trace stream if you are looking for a problem or for a special situation. You can enter Debug Mode by clicking **Mode, Debug**. When you enter Debug Mode, Proof's toolbar will appear as shown in Figure 5-2.

Further details on debugging are given in the "Advanced Debugging" section at the end of Chapter 14.

Figure 5-2. The Debug Mode Toolbar

Debug Mode is identical to Run Mode, except that the two following icons appear in the toolbar:

 E-Step

 T-Step

 Find a specified Object

E-Step executes the animation trace stream one command at a time. Each time you click on the E-Step icon, Proof displays the next line from the animation trace stream in status bar at the bottom of the screen.

When you click on the T-Step icon, Proof executes commands up to and including the next time or DT command. The individual commands between each Time or DT command execute automatically, without user intervention. Thus T-Step steps through the sequence of instants in time recorded in the animation trace stream.

Clicking on the "Find Object" icon allows you to locate any given Object on the screen. The queried Object is momentarily highlighted with a large, red cross. If a queried Object resides off-screen, the current view is relocated to make the Object visible

If you wish to change any command in your animation trace stream after observing it in Debug Mode, you must exit Proof and edit or regenerate the trace stream. You cannot make changes to a command while it is displayed in the status bar.

Exercise 5-2: Using Debug Mode

You may still be unclear about the sequence of event processing. What do we mean when we say that no further processing of the trace stream occurs until the most recently specified time is reached? We can observe this behavior directly using Debug Mode.

Open the sample animation in the exercise folder, and click **Mode**, **Debug**. Start by clicking →|E **E-Step** with the mouse. You'll see "Write Info Time 0" in the trace command display at the bottom of the screen, and "Time 0" will appear in the center of the screen. Watch the animation time display at the right end of the toolbar as you click →|E once more. You'll see the time updated twice a second until animation time 25 is reached. The "sample" animation runs with a speed of 6, so reaching time 25 will take 25 / 6 = 4.167 seconds. During this time interval, no further commands will be read from the trace file. Why?

Remember that when a time command is read, no further processing occurs until that time is reached. At time 0 (the beginning of the animation), the time 25 command is read and processed. You are looking at the state of the animation immediately after the Time 25 command has been processed.

Click →|E once more, and you'll see "Write Info Time 25" in the command window, and "Time 25" will appear in the center of the screen.

Now, click on →|T **T-Step**. Proof executes commands up to and through the next Time (or DT) command in the trace stream.

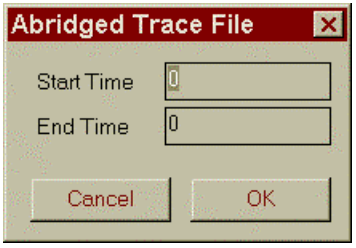
Experiment with the use of E-Step and T-Step until you are comfortable with them. If you reach the end of the trace stream, you can start over by clicking the **Time Jump** button and specifying that you want to jump to time zero.

Writing an Abridged Trace File

As you develop more complex, real-world animations, your trace streams may become very large. Often there is a warm-up period or a portion of the animation with little activity that you would like to omit during later viewings. You (or someone else viewing your animation) can always click on the **Time Jump** button in the toolbar and enter a time beyond the period of inactivity. However, this method of fast forwarding can take some time if the trace stream contains many thousands of commands.

Another approach is to have Proof create a separate .ATF file containing an *abridged* version of the large trace stream. This file will contain only an *interval* of animation time, which you specify. To create an .ATF file containing an abridged version of the currently opened trace stream, choose **File**, **Create Special Files**, **Write Abridged Trace**. (If no trace stream is currently opened, **Write Abridged Trace** will be dimmed. To open a trace stream, click on **Open Trace Only** or **Open Layout & Trace** in File menu. Then choose the highlighted, **Write Abridged Trace**.)

A dialog box of the following form will appear at the bottom of the screen:



The times being requested are in animation clock units. Fill in Start and End times for the abridged trace file and click **OK**. A standard File Save dialog will appear, allowing you to specify the name of the abridged trace file. Once you specify a file, Proof generates the abridged trace.How does this work when you are playing back the abridged trace file? The first command in the abridged trace file will be a time command that uses the jump keyword with the specified start time. When the abridged trace file is played, the time jump command will force Proof to immediately jump to the starting time you specified. Following the time jump command will be a sequence of Proof-generated commands that bring the animation to the proper state for the given start time. This portion of the abridged trace stream creates, places, and sets all Objects that exist as they should be at the given start time. Messages, Bars, Plots, and Paths are also updated.

Proof processes this portion of the abridged trace stream all at once (typically very quickly) and immediately brings up the animation at the given start time. The rest of the abridged trace stream contains the original trace stream commands until the specified end time. At that point, an end command is inserted and the commands following that end time are discarded. When you click **Go**, the trace stream proceeds as usual until the given end time is reached.

Abridged trace streams provide two benefits: first, the speed at which Proof brings the system up to the given start time and second, abridged trace streams are smaller streams, so disk space is saved.