16

Chapter Sixteen
DATA IMPORT/EXPORT
&
LAUNCHING OTHER PROGRAMS

Proof allows you to import and export data in the following formats:

- Proof can import CAD drawings (.dxf files).
- Proof can export layouts as .dxf files.
- Proof can import bitmap files for use as slides.
- Proof has a built-in screen-grabber that makes it easy to export bitmap images.
- Proof can export linkage (.lkg) files, which describe a layout's Paths.
- Proof can generate .avi video files.

In addition, an animation can invoke other software programs or issue operating system commands by using Proof's syscall command. In the sections that follow, we'll discuss the various forms of data import/export Proof supports, and we'll conclude with a discussion of syscall.

Importing Layout Geometry

The most common reason for importing geometry data is that someone else may have created a drawing that you can use as the background of your animation. You may still need to define Paths and clean up or simplify imported data before you can use imported layouts.

You might have available to you:

- A CAD layout generated on a PC
- A CAD layout generated on some other system
- A hard copy printout

Importing from a PC CAD File

In all versions of Proof, a built-in CAD import/export feature can be found through the **File** menu. This feature imports CAD drawings stored in the .dxf file format defined by AutoDesk, makers of AutoCAD, into Proof layout files. Most PC CAD programs support .dxf as an output format. Only .dxf files can be imported. .DXB and .DWG files cannot be imported. Proof layout files can also be exported and saved as .dxf files.

The DXF file format was once intended to be a universal CAD interchange format. However, Autodesk modifies the DXF format definition from time to time when new versions of AutoCAD are released. The format supported in Proof Release 4 is the Release 12 DXF format. Files saved under newer versions of the DXF format may import easily. But if there are problems during the import process, try using a CAD program to save the drawing as R12 DXF.

To import a .dxf file, click **File, DXF Files, Import DXF**, and select a file.

Once imported, the entire drawing will appear on the screen. (Proof automatically performs a zoom-to-fit.) For three-dimensional .dxf files, Proof ignores Z-axis information. Proof extracts Lines, Arcs, Text, Fills, and all Classes from the DXF file. All polygons are converted to individual lines. Text will show up in Proof's fonts with every attempt to preserve spacing and size.

Once the file is imported, a dialog box similar to the following will appear:

You can also access this dialog box by clicking **File**, **DXF Files**, **View DXF.**

Initially all layers and all line styles will be displayed. You can examine the individual layers used in the drawing by clicking on the **Next** and **Previous** buttons in the **Layers** box, and you can examine all line styles used by clicking on the **Next** and **Previous** buttons in the **Line Styles** box. To delete an entire layer, or all lines and arcs drawn in a specified line style, click on the respective **Delete** button.

Proof distinguishes different layers *only* during the CAD import process. Once an imported file is saved as a layout file, all layer information is discarded. Many CAD drawings use dedicated layers to contain annotations such as dimension information. Such information is probably inappropriate for use in an animation. The ability to delete such layers with a single mouse click greatly facilitates the process of cleaning up a CAD drawing for use with Proof – but again, you must perform these steps before saving the Proof layout following the import.

Similarly, Proof distinguishes different line styles only during the CAD import process. Line style information is lost once an imported drawing is saved as a layout file. All non-continuous line styles that are not deleted are drawn as continuous lines.

What's Wrong? My Screen is Blank After Importing a .DXF File!

This is not uncommon. Almost always, the reason is that your .dxf file has one or two drawing elements that have erroneously been placed a large distance away from the main part of your

drawing. These outlier elements can go unnoticed until Proof does a zoom-to-fit to assure that your entire drawing is visible on the screen. Accommodating a single outlier may result in the majority of your drawing being shrunk to a tiny dot on the screen.

To cure this problem, first see if you can find that tiny dot. If you do find such a dot, try zooming in on it (using the Zoom Box) to see if it really is the main part of your layout. If it is, do another zoom-to-fit, and use Box Edit to cut away the areas containing outlying elements.

If you can't see where the main part of your drawing is, save the current layout and edit the layout file. Take a look at any Lines, Arcs, or Text in the layout and write down the (X,Y) coordinates of a few of them. Bring your layout file back up under Proof and go into Draw Mode. Use the coordinate information you wrote down as a guide for where to look for your drawing. Remember, Proof's .dxf import feature works. The drawing is there. You just have to find it.

Importing a CAD file from another system

Many CAD users use non-Autodesk CAD systems for generating and storing plans and drawings. Proof's input capability is through the PC-based .dxf format. If you have drawings available to you in a nonstandard format, it may still be possible to use them.

CAD file formats differ in their effectiveness at interchanging data. However, with a little research (perhaps involving contact with your CAD vendor), it should be possible to determine if and how your file format can be translated to .dxf, which can then be imported to .LAY.

Generating a Layout File from Hard Copy

If you have a plotted, printed, or even professionally hand-drafted picture of your layout, but no computer file, you still might be able to get it into Proof's .LAY format. Again, the .dxf format comes into play.

There are many low- and moderate-cost packages for performing "raster-to-vector" (bitmap-to-vector) conversion. Many of these software packages support .dxf output. So, if you have access to a scanner and can locate conversion software that meets your needs, you can import scanned drawings into Proof.

Exporting Layout Geometry

Proof also includes a DXF export option, which saves Proof layout information into a DXF file for other uses.

To export a layout file into a .dxf file, choose **File**, **DXF Files**, **Export .DXF.** This will invoke

a standard file save dialog for specifying the name of the target DXF file.

Note that only geometry – Line segment, Arcs, and Text – can be exported. There is no standard file format that can support Proof's animation-related constructs such as Messages and Paths. Thus, you cannot form a closed loop by exporting then importing Proof data. Animation information will be lost!

Nevertheless, there are two reasons you might want to export geometry data. First, you might want to print your layout in vector format. Proof itself does not support direct printing. However, if you can export the geometry, you can print using inexpensive PC CAD or other drawing programs that accept .dxf file input.

Second, you might reach a point in a simulation project where you, instead of the facility layout people, "own" the most acceptable version of the system geometry in your .LAY file. This can happen when a facility design project goes through a simulation-driven phase of intense activity and massive design changes. In these cases, it may be advantageous for you to generate a computer-readable version of your geometry for import into someone else's CAD system.

Remember, as mentioned above, that you cannot currently expect to interchange files with a facility designer on a regular basis, because important animation information is lost each time you make the transition from .lay to .dxf, and CAD information (e.g. dimensioning) may be lost when you translate from .dxf to .lay.

Importing Bitmap Data

Proof can import bitmap data for use as slides in Presentation Mode (see Chapter 17). In Presentation Mode, you can read in and display static slides that have been saved in the .bmp or .pcx format. Instructions for displaying such a bitmap file are given in Chapter 17.

Proof runs in 256-color mode, so any imported bitmap files must be stored in 256-color (8-bit color depth) mode. Proof will reject bitmap files stored using more than 256 colors.

The adoption of the .bmp format in Proof means you can generate sophisticated bitmap images using other PC graphics programs, and use them as slides in Presentation Mode. Proof will read and use palette information, and can display up to 256 colors.

You may encounter occasional difficulties in importing .pcx files generated by programs that do not properly adhere to the published format. You should perform tests by displaying some sample files generated by the program you have chosen before you commit to that program for creating .pcx files.

If at all possible, you should create bitmap files that are going to be imported into Proof using a

size that is compatible with the resolution at which you intend to run Proof. Of course, you may not have control of run-time resolution, especially if you're preparing animations to be viewed on computers owned by other people. Proof's SLIDE command (see Chapter 17) provides a "stretch" option for .bmp files. Use of this option allows you to stretch or expand bitmap files to fit the screen. Proof employs a rather unsophisticated stretching algorithm, so you may find the results of using it unacceptable.

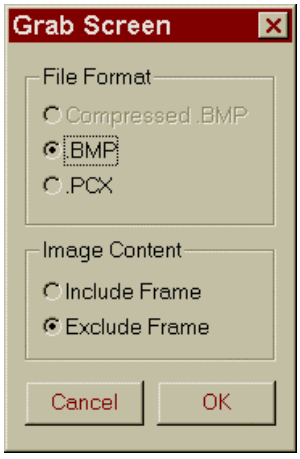
Finally, remember that importing bitmapped files directly into Proof only works with slides and Presentation Mode. You cannot directly generate a layout file by importing bitmap data! See the earlier section in this chapter on generating a layout file from hard copy for discussion of the raster-to-vector conversion required to turn a bitmap into a layout file.

Exporting Bitmap Data

There are two reasons you might want to export bitmap data. First, you may want to print it. This is a very different alternative from vector-based printing. Vector-based printing can capture the fine details of a zoomed-out layout on a single page, if you use a sufficiently high-resolution printer. However, bitmap printing can be very convenient when you want to print in "quick and dirty" fashion or when you want to include "screen snapshots" (possibly from a running animation) in a report.

The second reason you might want to export bitmap data from Proof is to save the screens for later re-display on the screen. This can be an important component of a Proof presentation (see Chapter 17). For example, you can zoom in on various parts of the layout and highlight them as separate slides prior to showing the animation running.

You can capture a Proof screen by clicking **File**, **Grab Screen**. When you do so, a dialog box of the following form will appear:



Proof can write .bmp (Windows bitmap) and .pcx (old PC Paintbrush) files. Most software tools that can accept .bmp files. .pcx files were popular in the early days of PCs, but now not even the latest version of PaintBrush accepts .pcx files. .pcx files do have the advantage of being remarkably small.

You can choose to include or exclude the Proof application frame. If you include the frame, you will save a complete screen image. If you are running at 1024 x 768 resolution, the saved image will be exactly 1024 x 768 pixels. If you choose to exclude the Proof application frame, only the client area of Proof's main window will be saved. Output .bmp files can be imported, displayed, printed, and even edited by a variety of PC software.

Exporting Path Descriptions Using a Linkage File

You can direct Proof to create a *linkage file* (.lkg) containing information about each Path in the layout file. This file describes the structure of Paths contained in a layout. The .lkg file format is designed to be easily read by other software (possibly directly into a spreadsheet or using some stand-alone programming tool, but more typically by using logic added to your simulation model). By reading Path information into a simulation model, rather than "hard-wiring" all Path-related information, you can be 100% certain that a model's Path information is identical to the corresponding Proof information. For example, if you lengthen a Path by 10 feet using Proof's drawing tools, export Path information into an .lkg file, and read the file into your simulation model, you won't have to manually update all parts of your model that depend on the given Path's length.

To create a linkage file, choose **File**, **Create Special Files**, **Write .LKG file.** If your model depends on the current contents of this file, then you must do write the linkage file manually

each time you modify your layout in a way that may change the geometry or path definitions. The linkage file is *not* written each time you save the layout file.

The linkage file consists of a series of Path Descriptions, one for each Path. Each Path Description contains a first line that contains key information about the entire Path, followed by zero or more segment information lines.

Depending on how your model or program is structured, you may be able to get it to read and parse a linkage file directly without having to manually edit the linkage file. Otherwise, you can extract the necessary information from the linkage file and use some intermediate form to get it into your model. The problem with extracting the information "by hand" is that the process must be repeated if the geometry changes.

The Linkage File Format

Format of the First Line of a Path Description

The first line of a Path Description in the linkage file has the following format:

PATH *pathname alphapart numericpart nsegs*

length traveltime speed

ACCUMULATING|NON-ACCUM CIRCULAR|NON_CIRC

All of these values are on a single line in the linkage file. The alignment is fixed-format, which can make the file easier to read using some computer programs.

The *pathname* is the name of the path.

If there are one or more digits at the end of the pathname, then *alphapart* contains the part before the digits and *numericpart* contains the digits, respectively. If the pathname ends in a letter, then *alphapart* is the full path name and *numericpart* is -1. This division into two parts is intended to help those who want to read path data directly into a simulation model that uses numeric values to manage a network of Paths.

The value of *nsegs* equals the number of segments in the Path. Most of the time you will not need segment information from the linkage file. However, if you want to use a program to process the linkage file, the value of *nsegs* tells you how many lines to skip between path entries.

The *length*, *traveltime*, and *speed* are as reported in Path Mode.

The last two fields tell whether or not the Path is accumulating or circular, respectively.

Length vs. Traveltime

If you do not plan to change the speed at which your model thinks Objects will move along a given Path, then you can use the *traveltime* directly.

If you think your speeds might change later, you should use the *length* and let your model calculate the traveltime. Then, if your speeds do change, you will still need to change the Path speeds in the animation, but you will not need to revisit the linkage file.

Format of the Segment Lines

Each of the segment lines in a Path Description in the linkage file corresponds to one segment of the Path. Most simulations do not reference individual Path segments, but if yours does, you will find the segment lines useful. The segment entries are in order (the first entry is the first segment of the Path, etc.). The format of each of the segment entries is:

LINE|ARC *cumulativedistance x1 y1 x2 y2*

The first word, LINE or ARC, defines whether the segment refers to a Line or Arc. The cumulative distance defines the distance from the beginning of the Path to the end of the segment. The x and y values identify the portion (of the Line or Arc) that comprises this segment.

The *cumulativedistance* reported with the last segment of each Path will always match the *length* in the Path entry.

A sample .lkg file is shown in Figure 16-1. Some of the data has been squeezed slightly to fit on single lines. If your modeling software supports only fixed-format input, take a look at an actual .lkg file to determine the columnar spacing. Lkg files are written in a fixed format.)

```
PATH LOOP LOOP -1 12 27.7080 30.0000 0.9236 NON-ACCUM CIRCULAR
ARC       3.1416      0.0000     -2.0000      2.0000      0.0000
LINE      4.1416      2.0000      0.0000      2.0000      1.0000
ARC       7.2832      2.0000      1.0000      4.0000      1.0000
LINE      9.2832      4.0000      1.0000      4.0000     -1.0000
ARC      10.8540      4.0000     -1.0000      3.0000     -2.0000
LINE     13.8540      3.0000     -2.0000      0.0000     -2.0000
ARC      16.9956      0.0000     -2.0000     -2.0000      0.0000
LINE     17.9956     -2.0000      0.0000     -2.0000      1.0000
ARC      21.1372     -2.0000      1.0000     -4.0000      1.0000
LINE     23.1372     -4.0000      1.0000     -4.0000     -1.0000
ARC      24.7080     -4.0000     -1.0000     -3.0000     -2.0000
LINE     27.7080     -3.0000     -2.0000      0.0000     -2.0000
PATH BIGCIRCLE BIGCIRCLE -1 1 9.4248 10.0000 0.9425 NON-ACCUM CIRCULAR
ARC       9.4248      0.0000     -1.5000      0.0000     -1.5000
PATH SMALLCIRCLE SMALLCIRCLE -1 1 6.2832 5.0000  1.2566 NON-ACCUM
CIRCULAR
ARC       6.2832      0.0000     -1.0000      0.0000     -1.0000
```

Figure 16-1. Sample .LKG File

Exercise 16-1: Exploring a Linkage File

Bring up the KANBAN animation (in the SAMPLE subdirectory). Choose **File**, **Create Special Files**, **Write .LKG file**, and specify temp.lkg as the name of the linkage file.

Use a text editor or word processor to examine the contents of the temp.lkg file. Write down the total length, travel time, and speed of each of the first few Paths. Then return to Proof, choose Path Mode, and verify that the values you wrote down are correct.

Generating .AVI Files (Making Animation Movies)

Use .avi files utilize a widely accepted format for storing video data. Proof can capture portions of an animation as .avi files. The use of .avi files has several advantages:

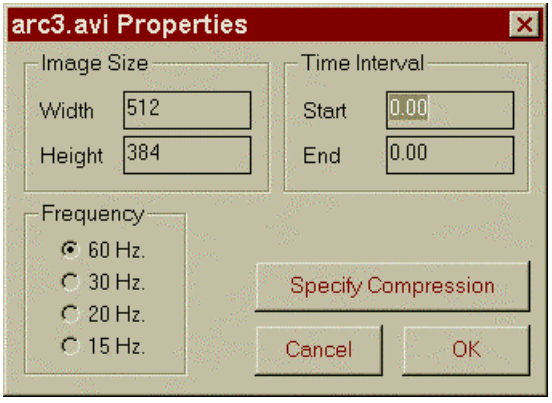
- .avi files are very portable. Any PC equipped with the Windows Media Player can play .avi files. This software is included in Microsoft Windows, and the very latest versions can be downloaded from Microsoft, free-of-charge.
- Proof software is not required for viewing an .avi file. If you'd like to place a demo on your web site and not have to worry about providing the Proof Demo Viewer, you can generate an .avi file and be confident that anyone who accesses or downloads it will be able to view it.

The use of .avi files has several disadvantages:

- .avi files can easily be very large.
- Software used to view .avi files may not always be fast enough to view a Proof-generated .avi file as smoothly as the animation runs under Proof.
- Because of the two preceding issues, it can be cumbersome to generate and manage full-screen .avi files. We recommend the use of at most quarter-screen images (half-size horizontally and vertically).

There are many circumstances in which the advantages of .avi files outweigh the disadvantages. You can decide for yourself. If you do decide to use .avi files, the actual generation process is very straightforward.

To capture a portion of a Proof animation as an .avi file, bring up your layout and trace files, and click **File**, **Create Special Files**, **Capture .AVI**. When you do so, a standard file selection dialog box will appear, allowing you to specify the name of the file to be generated. After you have specified a file name, a dialog box of the following form will appear:



This dialog box allows you to specify the size (in pixels) of the image to be captured, the starting and ending times of the interval of animation to be captured, and the frequency of image capture. The highest frequency available results from including every Proof animation frame in the .avi file. The second highest frequency results from including every other frame, and so on.

Uncompressed .avi files can be huge, so AVI files are typically compressed internally. Proof will require you to specify a compression algorithm. (If you click OK without doing so, Proof will tell you to specify the compression first.) Use "Microsoft RLE" unless you have a good reason for choosing another option. We've had good luck using Microsoft RLE compression

with a default image quality of 85%. Also, keeping the image size down reduces the AVI file size, as mentioned above. For distributing AVI files, you can often compress them further using a ZIP utility.

When you're satisfied with the specifications shown in the dialog box, click on the **OK** button, and Proof will generate the .avi file. During the generation of the file, Proof will show your animation in a window of size identical to the requested image capture size. Proof cannot generate images that are larger than the client area of Proof's application window. Don't worry if the animation has a jerky appearance. The overhead of writing the .avi file slows things down during file generation. Proof captures images at the frequency you've requested, so the output file will be as inherently smooth as the requested capture frequency allows (note: the numbers in your Frequency options may be different than those shown above due to differences in graphics hardware).

Launching Another Program from a Trace File

You may want to execute another program from within a trace file. To do this you use the syscall command.

The syntax of syscall is:

syscall *command or program name*

For example, to launch myprog.exe, you would use

syscall myprog .exe

Each time the syscall command is issued, Proof relinquishes control of the screen and mouse while the other program is running or the operating system command is executing. Upon completion of the command or other program, Proof regains control and continues processing the trace file starting with the command following the syscall. If the program issued by syscall changes or rewrites the currently executing trace or layout file, execution errors could occur.