

Using the Process Flow Diagram Object to Communicate Information

Frank C. DiIorio, ASG, Inc., Cary NC

INTRODUCTION

Many routine tasks in the workplace require analysis, or at least awareness, of trends in large volumes of data. The application development tools in the SAS System are justly renowned for their ability to both manage large, disparate data sources and present individual data points and summary measures.

The presentation of data and statistics has changed significantly over time. As programming tools become more powerful and computer hardware costs drop, graphically-oriented analytical engines have come within reach of most users. The power of these tools is enormous — one can view a multivariable display of thousands of data points, quickly get a feel for trends in the data, and then isolate and obtain information about individual points.

When correctly designed, these sophisticated analytical tools can exploit the strengths of both the graphical and character-based worlds. But what happens when a user's requirements cannot be met with off the shelf tools such as SAS/INSIGHT, JMP, or SAS/LAB software? It becomes time to dig into the SAS/AF toolbox and "roll your own."

This paper describes the evolution of a graphically oriented solution to an engineering problem. It describes the background of the problem; presents the alternative solutions we considered; and outlines the preferred solution in some detail. Although the setting is the semiconductor industry, the methodology should be applicable to other venues.

Notes: [1] All Figures are displayed following the text of the paper. [2] All data are simulated.

THE SETTING

Mitsubishi Semiconductor America Inc. (MSAI) manufactures chips in its Durham, North Carolina facility. It is a state of the art, complex, and highly automated manufacturing environment run by teams of design engineers, quality assurance (QA) analysts, and computer systems personnel.

Over the last three years the Information Systems Development (ISD) group at MSAI has developed a "near real-time" SAS-based data capture and analysis system to support the engineering and quality assurance staff. The system is event-driven in the original sense of the phrase — as cassettes, or lots, of wafers are moved within the fabrication area (a.k.a., the "fab") they are measured for QA and other purposes. This information is passed from equipment in the fab to an ORACLE database. The population of the database triggers a series of SAS programs, which generate Shewhart quality control charts using SAS/QC. If the data sent to SAS signals a condition requiring intervention (values drifting away from target specs, a value out of specification limits, etc.) the

system pages one or more engineers and in some situations provides other types of feedback.

The system is characterized by its extensibility. As usage increases, so does the number of requests made by the engineers for new reports, data, and screens. The request that spurred development of the subsystem described in this paper came from Photo Etching engineers. This step of the fabrication process is repeated numerous times before a chip is ready for packaging. The wafers containing the chips are grouped in lots. Each lot remains at one of 12 photo units for about an hour. When a lot completes its stay at the photo unit, numerous pieces of information are sent to the database. Among these are time into the unit, time leaving the unit, lot number, and various measurements useful for quality assurance. Ideally, once one lot completes another begins. Temporal gaps between lots are due to equipment problems and/or the amount of work to process.

THE NEED

The engineers wanted a visual display of completions for each of the photo units over a 24-hour period. They also wanted to understand the reasons for "negative space," or gaps in production. Both completion and gap information already existed, but could only be found in multiple data sources, which were in turn displayed in multiple, character-based screens. The proposed application would integrate the screens and present information on completions and gaps together for the first time.

The engineers also wanted the display to be interactive. Each event (either a completion or gap) would be labeled with a lot identifier. Clicking on an event would retrieve information. For both types of events, start and end time would be captured. For completions only: lot numbers and chip type. For gaps only: equipment status and down time cause(s).

Other requirements, some of which were satisfied by the first version of the system:

- Hard-copy reports from the underlying database
- Screen print of the display
- User-selectable date. Default is current date.
- User-selectable gap size. Default is 60 minutes (i.e., any down time less than 60 minutes is not considered problematic)

Other design issues:

- The screen size was constant (scrolling issues were not raised). All target users would be using 17" x-terminals.
- There would always be a 24-hour period ending at 7a.m. (along the y axis) and 12 photo units (along the x axis)

- The user should be able to display multiple days in a single invocation of the application.

Just *how* the data were presented was left to the ISD team charged with designing and writing the application. Since the needs were somewhat different than existing tools in the system there were practically no standards to adhere to. Lingered in the background, though, were basic graphic design principles, primarily those promoted by Edward Tufte. See Appendix A, "Information Display Guidelines," for a brief outline.

The heart of the application is what appears to be, in Base SAS's PROC CHART terms, a bar chart with subgroups. The value-added by the FRAME technology, and what will be discussed extensively later in this paper, is that in the Photo Unit application what appear to be subgroups are actually hotspotted "nodes." Clicking on a node triggers display of data from one or more databases. A screen shot of a typical (in this case, simulated) day is shown in Figure 1.

SOLUTION STRATEGY

The first pass tried to use PROC GCHART so that issues of axis labeling, titling, bar spacing and the like would be handled more or less automatically. The problem with this solution was that the events, or data points plotted, needed to be labeled — this in turn required that instead of a clickable, selectable *point* there be a wider *area*, one wide enough to contain the labeling and to make selection (i.e., pointing and clicking) easier. Recall that the standard representation of an observation in PROCs CHART and GCHART is a user-selectable character. A single character isn't wide enough.

The next pass was creation of the diagram entirely from ANNOTATE and DSGI. As only an occasional user of SAS/GRAPH I find these products hard to pick up "cold." My feeling is that they are most effective when they are more or less constantly used and their intricacies and quirks become familiar. They don't readily provide the automatic hotspotting required for the drill-down component of the application.

The solution we finally arrived at was to use the Process Flow Diagram (PFD) object in Version 6.11 of SAS/AF. The rest of this paper describes how we constructed datasets input by the PFD object and demonstrates the resulting application. Along the way, we'll discuss aesthetic considerations.

AN INTRODUCTION TO THE PFD OBJECT

Before going into application development specifics it would be helpful to discuss some of the capabilities of the PFD class. In the "SAS/AF Software FRAME Class Dictionary" the PFD is described as a "diagram composed of symbols with connecting arrows and descriptive text." Basically, it's a flow chart. The object itself is an interactive drawing environment modeled after numerous drawing packages in the PC and Mac environments. The developer (and, optionally, the user) drags drawing icons

into the object. Arrows, rectangles, and other charting widgets are available. Text can be associated with most of the widgets. Each widget is named, and may be referenced by the SCL program attached to the FRAME entry. Each widget is automatically hotspotted — when the user clicks on it, a labeled section of SCL code can execute. Information about the widget can be obtained and altered by using the `_GET_INFO_` and `_SET_INFO_` methods, which load SCL lists with various attributes such as node name, label, text, and color. The PFD in Figure 1 has over a hundred rectangular nodes (some labeled, others filled), lines (the y and x-axis, the y-axis tick marks), and text (title, axis labels).

It is tempting to think of the PFD as a pure flowcharting tool. But when one takes the broader view and looks it is apparent that what it does is, essentially, drawing. If all we need for display is unconnected lines and boxes the PFD class will be the tool of choice. For the purposes of the Photo Unit application we conceptualized each gap and completion as a rectangle, or node. Clicking on the node (which is automatically hotspotted) runs code that extracts information for that event/gap from the relevant database.

USING THE PFD

One method of building a PFD is to manually draw boxes and lines, adding descriptive text to each element. Enter a new FRAME entry and specify `MAKE→Process Flow Diagram`. Once you size the object SAS/AF presents you with a screen resembling that in Figure 2.

Manual drawing is clearly impractical in applications where the number, position and even coloring of nodes varies. The key to dynamically creating the PFD is the object's ability to receive its drawing instructions from an input SAS dataset rather than from manually manipulated drawing objects. This is specified in the Object Attributes pop-up, shown in Figure 3.

Understanding the structure of the input dataset can be challenging. Each object in the diagram (arrow, text, box, etc.) is an observation. The object's type is identified by the `_TYPE_` variable — 22 is the title, 1 is an arrow, 2 is a rectangle, and so on. There many other variables that identify the object's coordinates within the drawing area, text, font, font weight, color, and the like. Unfortunately, when a PFD creates an output dataset it does not create labels for the variables. Understanding the exact nature of many of the variables is, in many cases, a matter of trial and error (and calling SAS Institute Technical Support!)

We decided to manually create a bare-bones diagram that contained at least one of all the elements needed for the full-blown application. We saved the dataset by entering the Object Attributes pop-up and specifying "Output" attributes for Type ("Data Set") and Value ("work.pfd"). This dataset was the basis for building the input datasets created by the application.

The bare-bones, but representative, diagram is in **Figure 4**. A listing of the dataset used to create it is shown in **Figure 5**.

Some items of interest when dynamically building PFD input datasets:

- The figure only shows variables created by the PFD object. Other variables may be present in the dataset.
- There must be an title observation (`_TYPE_ = 22`). You can blank the observation's value of variable `_TEXT1_` to avoid any unwanted titling at the top center of the display.
- The variable `_LABEL_` is created by the PFD object. It is used by the Photo Unit application to identify a selected node (this is done by using the `_GET_INFO_` method and populating a list. The technique is discussed in "Building the Application," below). Values of `_LABEL_` should be unique so that screen objects can, in turn, be uniquely identified.
- The `X1-Y1` and `X2-Y2` variables identify the top left and bottom right coordinates of the object, respectively. The coordinate 0,0 identifies the top left corner of the PFD object.
- `_TEXT1_` contains the text displayed with the object.
- `_ICOLOR_` identifies the color of the object (line color or rectangle fill). It may be changed via SCL. When a node is selected, the Photo Unit application turns its color to green and turns the previously selected node to its original color.
- `_FONT_` specifies the font to use when displaying `_TEXT1_`. The format varies from system to system. Valid choices may be identified by entering the VIEW → Change Font menu item found in most windows. A valid font in our HP-UX system is
`-adobe-helvetica-bold-r-*-18-*-*-p-*-*`
 A mouthful, but once you enter it in the application's "template" dataset (described below) it never has to be typed again!

BUILDING THE APPLICATION

Understanding the structure of the input dataset is the basis for building the application. Once this is grasped the rest is not necessarily easy, but becomes mostly a matter of managing a lot of details. This section describes how we constructed the diagram.

The layout of the diagram was fixed — the screen size would always be the same, the number of photo units (and thus the number of vertical bars) would always be 12. This simplified the application's development. Here are some of the steps.

Create a template dataset. This dataset contains parts of the diagram that are constant or only slightly changing from diagram to diagram. These include: the title, vertical (time) axis tick marks and labels, and horizontal (photo unit) axis labels. We used the FSEDIT and FSVIEW pro-

cedures to add and modify observations in the template dataset until its appearance was acceptable.

This template dataset can be stored permanently and used in multiple invocations of the application. Only a few modifications to a few observations are required for its use (these are described in "Combine the Datasets," below).

Create the event/gap dataset. This dataset is an extract from a production database. It contains information about event completions for each of the twelve units for the user-specified date. By default, the date is from yesterday at 7:00 a.m. to today at 6:59 a.m.

There were several design considerations here. One was representation of each event. Completion times are recorded to the nearest second. Obviously that is too thin a line to easily click on. Since we expected about one completion per hour and we knew the number of pixels used along the vertical axis we decided to represent each completion by a range of points along the y axis. That is, the completion went from being a horizontal line on the chart to being a rectangle, in this case, 15 pixels high. The horizontal coordinates of the node were easily computed. 64 pixels was an acceptable width, given the information that would be displayed in each rectangle. The starting x coordinate was a direct function of the photo unit number (happily, they were numbered simply 1 through 12). The `_LABEL_` variable's value for these observations was of the form DATAxxx (e.g., DATA001, DATA101).

Another consideration was color of the node. The default fill color was white. If a completion represented a rework of a lot the engineers requested a yellow fill. Rework status was derived directly from the lot numbering conventions employed at MSAI.

Finally, there was the issue of the absence of completions, or gaps. In theory, a gap exists if one event completes at time "x" and another does not begin at time "x+1". For the purposes of this application, however, we identified gaps if time between completions was 60 or more minutes or if a unit had no activity at all during the period of interest. To identify gaps we sorted the data in unit-completion order, then ran the data through a DATA step, noting the time difference between completions for a given unit. If the difference was greater than or equal to the 60 minute threshold we wrote an observation with a `_LABEL_` value of GAPxxx. The fill color for these nodes was red.

Combine the datasets. The PFD input dataset is simply a concatenation of the template and event/gap datasets. As the template dataset observations are read in we make some minor modifications to the `_TEXT1_` field. The title reflects the current date and the number of minutes in the gap, and the labels for the earliest and latest (bottom and top) vertical tick marks are adjusted to contain the day number (for example, "7am 19" indicates 7 a.m. on the 19th).

The PFD input dataset is complete once the modified template dataset and the event/gap datasets are concatenated.

RUNNING THE APPLICATION

This discussion has so far ignored one of the key behavioral aspects of the application. Rather than simply looking at the current day's events the engineers wanted to be able to view more than one day in a single invocation of the program.

The ideal mechanism would be a slider bar or similar selection tool on the same screen as the PFD. The user could select a date, click on a Run icon, and update the display. However, Version 6.11 did not have a `_REPOPULATE_` (rewrite, redisplay) method. So even though we could run the code needed to rewrite the PFD input dataset we could not actually update the display.

One solution was to split the application into two FRAME entries, one for date selection, the other for the actual display. The first FRAME allows specification of the day to process (other user-selectable features that would be appropriate for this FRAME are described in "Enhancements," below). The screen is shown in Figure 6.

Once the user clicks "Create Diagram" the SCL program builds a WHERE clause and applies it to the production database. If the ATTRC function with the "any" parameter returns a value of 1 there are observations to process and the template modification-event dataset construction process begins. Once the dataset is built, control is passed to the FRAME entry containing the PFD. A typical screen is shown in Figure 1.

Once the PFD is displayed the user can click on any event or gap. A word of caution for PFD users: some programming was required to avoid error conditions when non-hotspotted areas were accidentally selected. If the list filled by the `_GET_INFO_` method had a length less than three, the user did not select a hotspotted area.

If, in fact, an event or gap node is selected, the `_GET_INFO_` method retrieves the `_LABEL_` value. Recall that these are unique values of the form DATAxxx and GAPxxx. This value is used as in a WHERE clause applied against the PFD input dataset. We then FETCH the single observation identified by the WHERE clause and construct a character variable to display below the x-axis.

If an a photo unit completion is selected, the following items from the production database are displayed: key (lot number), chip type, time in, time out, and elapsed time. If a gap is selected, the following items are displayed: start of gap, end of gap, and elapsed time. In either case, when a node is selected it turns green, and the previously selected node reverts to its original color. Figure 7 illustrates selection of an event completion (the bottom node for PU04 was selected and turned green). Note the line of text below the horizontal axis.

AESTHETIC DESIGN CONSIDERATIONS

The display's design was driven by a number of aesthetic considerations.

The first consideration was simply to opt for a graphic, rather than printed, display of the completion/gap information. The stacking of events in vertical bars according to photo unit allows the engineer to see at a glance how the units are performing. Is there an even distribution of events (white nodes)? Does one unit seem to have an unusual concentration of reworks (yellow nodes)? Likewise, are gaps (red nodes) clustering in a particular unit?

A consideration that was not acted upon was to vary the width of the nodes to reflect elapsed time (the difference between time entering the unit and time at completion). Ideally, the longest elapsed time of a completion event would be represented by a full 64 pixels (the current size for all completions). The width of other, quicker nodes would be proportionally less. This display technique would not only give a quick view of *when* the event completed but also *how long* it took to complete. The "information density" of the chart would be dramatically increased. See "Enhancements," below, for further comments.

The display itself is as lean as possible to communicate the required information. Grid lines not only cluttered the display but also interfered with node selection — successful selection of a node overlapping a grid line was unpredictable.

Finally, a design consideration of the screen is that the screen itself is not adequate for all engineers' needs. The users requested supplemental, text-based reports so that all information could be seen at once: in-time, out-time, identifiers, and other information would be in the report. All this information is available in the PFD input dataset. Unlike the PFD, however, it is all presented at once rather than on demand by mouse clicks. The combination of easily interpreted, visual presentation of completions and gaps along with more traditional text based reports is a potent one — all the users' needs can be satisfied by a single data set and two means of presentation.

ENHANCEMENTS

Printing. Although this sounds straightforward in practice it is remarkably problematic. A print dialog is displayed when the user clicks the right mouse button when pointing to a PFD object. The dialog choices and printer destination selection can be a troublesome process for a novice or occasional user. Ideally, the button would be disabled during execution and printing controlled by a pushbutton or other user-defined mechanism. In Version 6.11, however, the right mouse button cannot be disabled. The only available option is providing a "Print" mechanism that is easy to use. This is possible by passing `"_PRINT_"` to the `"_SET_CMD_"` method for the PFD object and then SUBMITting a GOPTIONS statement and GREPLAY code.

Another issue, *what* is printed, is more basic. The print command prints only the contents of the PFD object. Other FRAME objects such as graphic text used for titles, annotations, and the like are *not* printed. This is why we had to go to the trouble of adding observations to the template dataset that would have been easier to imple-

ment using graphic text and other classes. The rule is simple: *anything* you want to print must be part of the PFD.

Handling Overlapping Nodes. The number of pixels in the node height reflects the anticipated 60 minute per lot throughput for each of the Photo Units. Sometimes, especially in the case of the less-demanding reworked lots, more than one lot per hour can be processed. This creates overlapping nodes. This is illustrated in Figure 8.

As a means of communicating information, this is desirable: the abnormally high activity occurring at that time is readily apparent. The problem lies in node selection and/or deselection. Sometimes the nodes are so closely overlapped that it is impossible to select the "lower" (partially visible) one. In other cases if one of the overlapped nodes is successfully selected and filled with green the program cannot revert to its original color once the node is deselected. Note that the code for overlapped boxes is identical to those that do not overlap. Several calls to SAS Institute technical support identified the source of the problem as the way AF and SAS communicate x-terminal commands to the HP Unix operating system.

Proportional-Width Nodes. The appeal of this was noted earlier ("Aesthetic Design Considerations"). Implementation is problematic. The calculation of the node width is straightforward: the lot that took the longest to complete at a unit would have the widest (64 pixel) bar. Lots completing faster would be narrower. However, since the seven character lot number had to be used for labeling the node it would be almost impossible to fit this identifying information on faster/narrower completions.

Additional Information. The information displayed when a gap or event is clicked is, by design, fairly sparse. If the users wanted more items we could add more lines of text to the message area at the bottom of the screen. The other approach would be to display it in a pop-up window. As of this writing, though, no request for additional displayed information has been made and so no decision has been necessary.

CONCLUSION

The PFD application described here is significant on two fronts. First, it provided the MSAI engineers with a visually appealing, intuitive tool which has proven to be quite popular. Second, from a SAS developers' perspective, it drives home the importance of reading and re-reading the documentation. Even if you think you know the simplest of procedures or techniques a refresher read-through of the documentation can point out techniques and options that may have been forgotten (in this case, the obscure but pivotal item was the ability of the PFD class to accept input datasets for drawing instructions).

ACKNOWLEDGEMENTS

My thanks to the AF technical support staff at SAS Institute for their expertise and patience. A special thank you is due Art Alexander for his assistance, code samples,

and large doses of moral support, especially during the early phases of this project.

QUESTIONS? COMMENTS?

Your questions and comments are valued and encouraged! Contact the author at:

ASG, Inc.
102 Westbury Drive
Chapel Hill NC 27516-9154

919.942.2028
fcdiorio@aol.com

INFORMATION DISPLAY GUIDELINES

Edward Tufte's "The Visual Display of Quantitative Information" (1983, Graphics Press, Cheshire CT) is the most lucid and compelling discussion of statistical graphics written in recent years. Among his guidelines for "graphical excellence" are:

- present many numbers in a small space
- encourage the eye to compare different pieces of data
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- induce the reader to think about the *substance* [emphasis added] rather than about the methodology, graphic design [...] or something else

The book promotes principles which enable graphics to "reveal data." This is best accomplished by graphics which give the reader "the greatest number of ideas in the shortest time with the least ink in the smallest space." Economy of presentation (reduction of "non-data" and "redundant ink") is emphasized so that the data itself can be the center of attention. This will promote the graphic's ultimate goal — "to give visual access to the subtle and the difficult, that is, the revelation of the complex."

Anyone charged with the design of graphical presentation of data should read this book.

Process Flow Diagram

Figure 2: Process Flow Diagram
object. Drawing toolbox is on the left.

SAS: Process Flow Diagram Attributes

Name:

Input

Type: Value:

Output

Type: Value:

Scrollbars

☒ Vertical
☒ Horizontal

Run-Time Options

☒ Allow edits
☒ Confirm saves

Autosave Value

Additional Attributes

Diagram options...
SUBMIT replace string...
Command processing...
Object help...
Instance variables...
Object links...
Custom attributes

Figure 3: PFD object attribute screen. Note "Input" type of "Data Set" and its value ("work.pfd"). This is where we change the default activity of the object and specify a SAS dataset that we build on the fly, based on the specified day's production data.

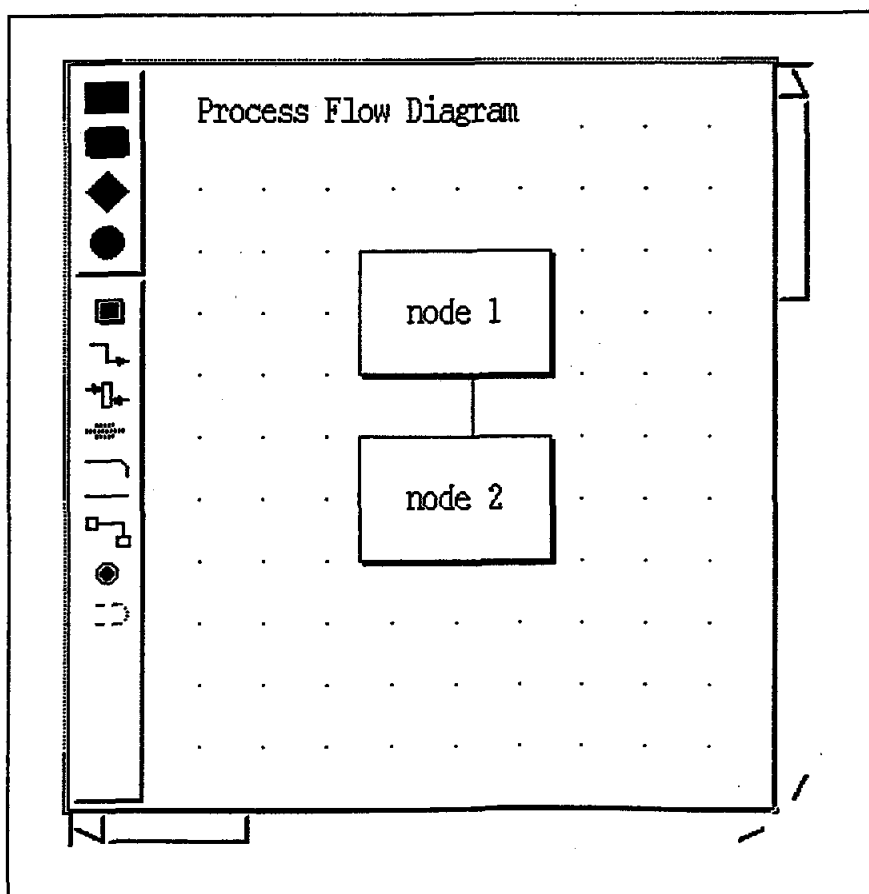


Figure 4: Build a simple PFD with the two main structural elements of the full-blown application - text-filled rectangles and lines.

11:27 Tuesday, July 8, 1997

OBS	_VUE	_SUBVUE	_LINK	_TYPE	_ID	_LABEL	_HID	_TID	_X1	_Y1	_X2	_Y2	_TEXT1	_JTEXT	_TCOLOR
1	1	0		2	1	PFD2	.	.	144	192	240	256	node 2	MM	BLACK
2	1	0		2	2	PFD1	.	.	144	96	240	160	node 1	MM	BLACK
3	1	0		22	3	PFD0	.	.	64	16	224	32	Process Flow Diagram	UL	BLACK
4	1	.		1	4	PFD3	1	2	200	192	200	160		MM	BLACK

OBS	_LWIDTH	_LSTYLE	_LCOLOR	_ICOLOR	_SCOLOR	_HEAD	_APOS	_TPOS	_STYPE	_SWIDTH	_XCOLOR	_ZCOLOR	_FONT
1	1	0	BLACK		CXD6DBD6	.	.	6	2	BLACK	CXD6DBD6		
2	1	0	BLACK		CXD6DBD6	.	.	6	2	BLACK	CXD6DBD6		
3	1	0	BLACK		CXD6DBD6	.	.	6	0	BLACK	CXD6DBD6		
4	1	0	BLACK			0	0.5	.	.				

OBS	_PERP	_OPTION	_FAMILY	_FNAME	_CSET	_POINTS	_STYLE	_WEIGHT	_WIDTH	_PRTY	_PORT	_CORNER
1	0	0	-1		-1	-1	-1	-1	-1	'00	0	0
2	0	0	-1		-1	-1	-1	-1	-1	'00	0	0
3	0	0	-1		-1	-1	-1	-1	-1	'00	0	0
4	1	0	-1		-1	-1	-1	-1	-1	'00	0	0

Figure 5: Listing of dataset WORK.PFD, the dataset used to create the display shown in Figure 4.

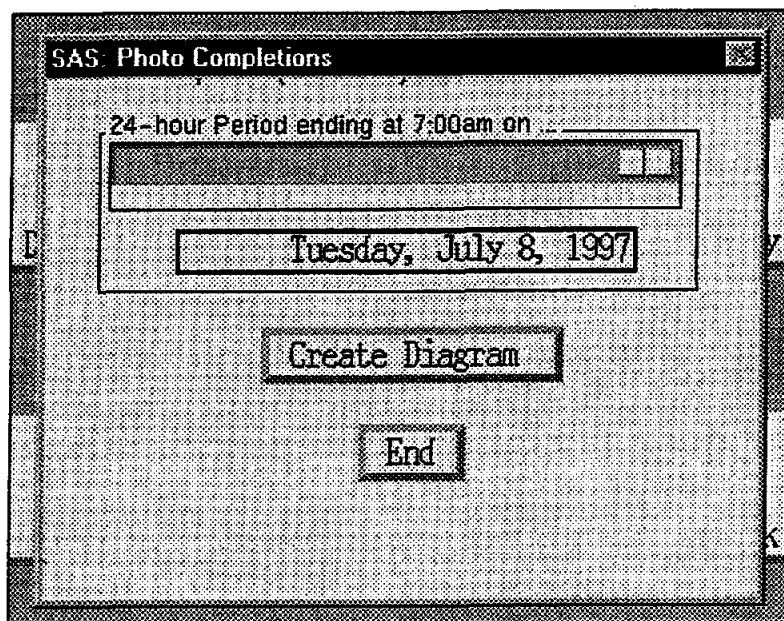


Figure 6: Driver for the application. The engineer uses the slider to select a date. When "Create Diagram" is selected, the program checks for the presence of data for the date. If there is any, the input dataset is built and a display similar to that in Figure 1 is shown, usually in about 12-15 seconds.

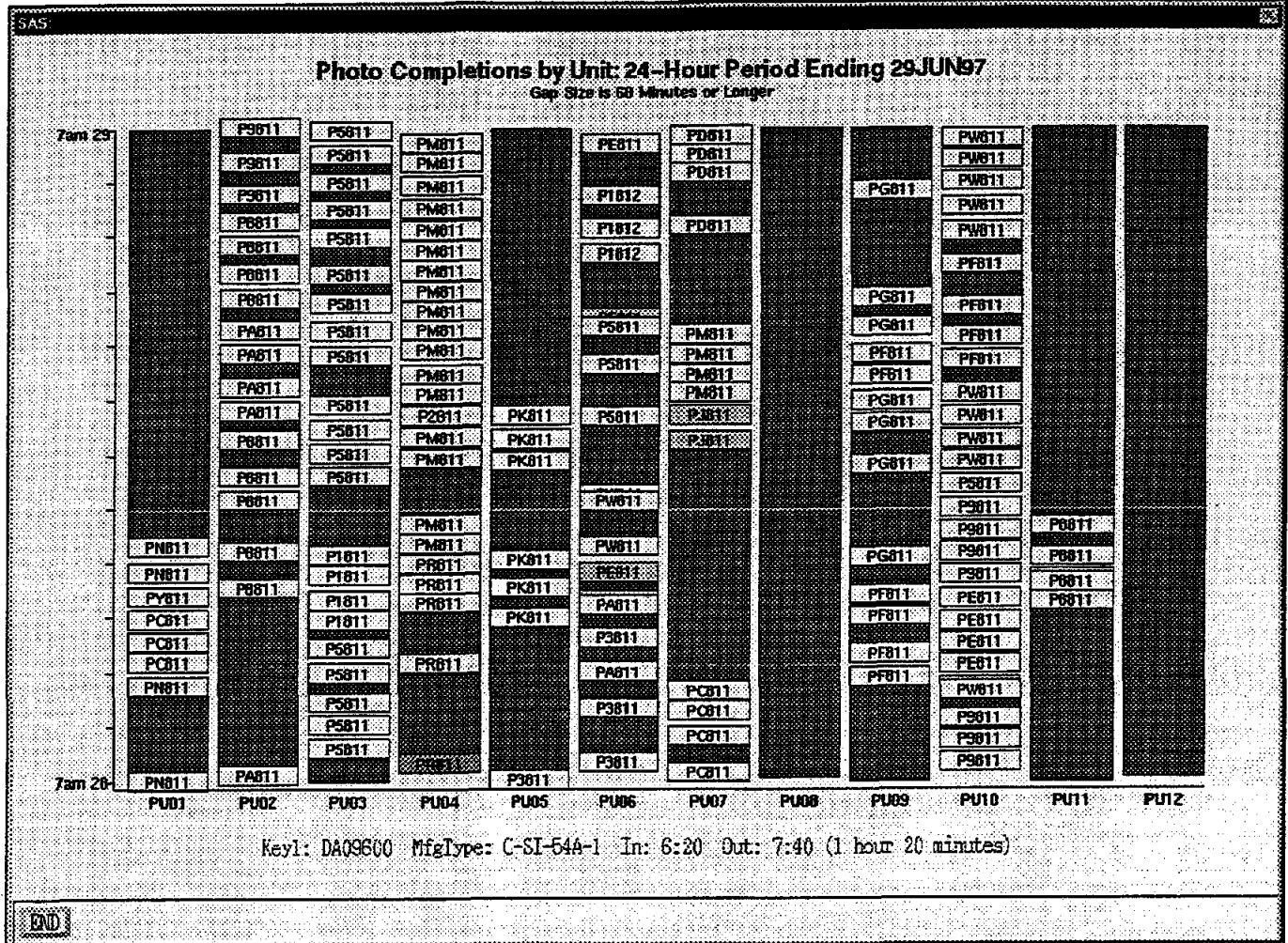


Figure 7: User selects a completion at the bottom of Photo Unit 4 (PU04). The node color is changed to green and information about the node (in this case, simulated data) is displayed.

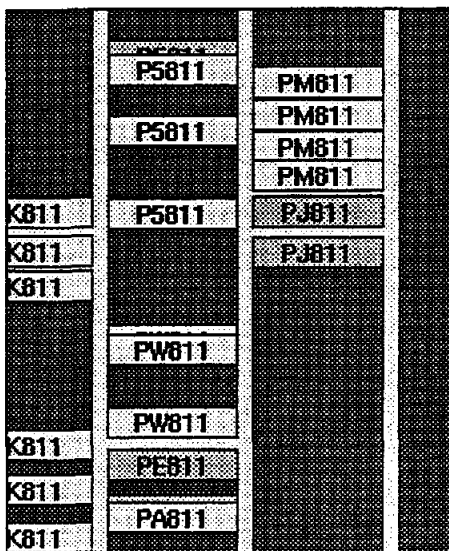


Figure 8: The Overlapping Node problem. "P5811" at the top of the display obscures another completion occurring at almost the same time. Likewise, "PW811" is also significantly overlapping another completion. Sometimes the user can select the "hidden" completion, sometimes not. Colors for these nodes do not necessarily turn green, then back to their original values.