%PRINT DRIVERS: Teaching SAS® Software to Speak
The Many Languages of Document Publication
Paul Wehr, Parke-Davis Pharmaceutical Research, Division of Warner-Lambert Company, Ann Arbor, MI

ABSTRACT
This presentation introduces a facility for creating reports for which the output device can be selected retroactively. That is, the programmer can create a report, in a DATA step, then at submit time decide to generate standard text output (as you would with PUT statements), PostScript® output, Rich Text Format (RTF) output, or Netscape™, simply by specifying "DRIVER=(driver-name)" in the macro that initializes the output service. This service also allows the programmer access to text formatting features such as centering, boldfacing, underlining, large and small type, line drawing, and the creation of tables.

Introduction:
Creating output drivers for SAS reports allows a user to select an output format at run time. This process is analogous to the print drivers found in most word processors, which allow the printing of a document on more than one kind of printer. This paper will be of interest to anyone involved with publishing the results of an analysis, presenting the results to people without access to the SAS system, or transporting the information to other applications or systems. This is especially true for those who must provide these services for an unusually large volume of reports.

While this topic is conceptually very simple, its implementation is somewhat technical. In order to accommodate a wide audience, the paper is divided into three sections. The first section is intended for a general audience. The second and third sections will require a little more technical background to be well understood.

The first section will provide some examples of the system, and illustrate some common reporting situations for which the Print Driver system holds some distinct advantages over standard SAS reports. It will also investigate the basic principles behind providing an abstract reporting system.

The second section will describe the interface between the Print Driver system and the DATA step. This section will describe the "middle ground," or abstract meta-language that mediates the conversation between a user's document concept and the digital implementation.

The final section will describe some of the technical details on how the Print Driver system has been implemented by the author. This section also includes some suggestions to developers on techniques for efficiency that the author discovered during multiple iterations of the Print Driver system.

What's the Problem?
One of the common questions seen on SAS-L® and comp.soft-sys.sas® is "How can I convert my SAS report to ________?" Most often this blank is filled with "WordPerfect®" or "Microsoft Word", but periodically requests have been made for PostScript, TeX, Microsoft® Excel, and more recently, HTML. Unfortunately, there is no universal, automated answer to this question. Because of the nature of the PUT statement—as I will describe later—this type of document conversion flexibility is simply unobtainable. What is needed is a replacement for the PUT statement that allows the programmer to design a document that can be represented in a wide range of formats. At the most fundamental level, the Print Driver system was created to provide such replacement for the PUT statement that allows this type of document flexibility.

What About the Drivers in SAS's GPROCS?
SAS Institute does provide an extensive set of device drivers in SAS/GRAPH® software, which include the procedures GPRINT and GPLOT. None of these drivers, however, provide an output format that maintains the description of the report's content in the same way that we are interested in here. This means, more importantly, that no translation is available for converting this output to other popular word processing formats, or standard document description languages such as HTML and SGML. Although one can create reports with these drivers which are very pleasing to the eye when printed on a page, the technique is really just a higher-resolution version of the PUT statements that one uses in a DATA step.

EXAMPLES
Before presenting the details of the Print Driver system, I would like to first present some examples of what the drivers are trying to do.

In Figure 1, there is an example of a greatly simplified demographics, presented in a fixed-pitch, space-delimited, ASCII-based text file. This is the type of output that is generated by default using the SAS system. In this case, however, the report was generated using the Print Driver system, by selecting the "Text" driver.

![Figure 1: Driver output using the Text driver](image)

The text driver provides a convenient output format when we are interested in immediate results. Unfortunately, as we know from our experience with traditional SAS output, we know that it is less than ideal for use in publishing software. Since many documents are ultimately prepared using personal computer word-processors, any tool that could create this report as a word-
Information Visualization

processing table would be extremely useful. Figure 2 shows the output of the same report that generated Figure 1. The only difference is that in the second example, the "DRIVER" parameter was set to "RTF".

![Figure 2: Output using the RTF driver as interpreted by a word processor](image)

by setting the DRIVER parameter to POSTSCRIPT. This driver allows the user to generate a presentable report directly from the SAS application to a PostScript-compatible device. This driver also provides a mechanism for generating a PDF (Adobe's Portable Document Format) file, which is gaining popularity in the complex document storage and management community. The image shown in Figure 3 is, again, the same report, this time using the POSTSCRIPT driver.

The last example, shown in Figure 4, is an image of an online display of the same report, generated with the driver parameter set to "NETSCAPE". Although the driver is designed to accommodate Netscape's idiosyncrasies, a few minor modifications would make the driver HTML 3-compliant, giving it the power to write documents directly to the World-Wide-Web.

![Figure 3: Print Driver output using the PostScript driver, as interpreted by a PostScript printer](image)

Benefits In Everyday Use

The Print Driver system is designed to be more than a set of output device drivers for printed output. It is a bridge linking existing islands of technology. The system is most effective when used as a conduit through which information may flow from the SAS system (designed to store and analyze data) to document formats that are designed to interact directly with people. Below is a list of some of the situations where we are finding the Print Driver system to be extremely useful:

- Recently, report programmers at our site have been experimenting with some of the font options supported by our mainframe printer. Although it has improved the appearance of many reports, the timing is poor. As the company begins a down-sizing effort, the announcement has been made that the mainframe printer that these programs depend on, will soon be unavailable. This renders all these report programs virtually unusable. The Print Driver system is now being used to replace the printer-specific sections of these programs in order to solve the problem not only now, but for printer and platform changes that may occur in the future.

- A programmer in our data management department created an application designed to automatically identify anomalies in a clinical database. Unfortunately, the recipients of these reports insisted on them appearing in the same way they did when they were manually created in WordPerfect. Since the default SAS output was unable to re-create the WordPerfect version, the programmer used the Print Driver system to generate the report in PostScript format. Furthermore, because the report was created using the Print Driver system, this program has automatically gained access to all the other drivers available with this system, including the Rich Text Format (RTF) driver. Discussions are in progress about having the SAS job generate the same reports in RTF, and automatically sending them as e-mail attachments to the study monitors in the field, bypassing a large part of the current process. The monitors could then edit this report by filling in the blank "Response" section of the form, and e-mail it back to the data management specialist responsible for that data.

- Most recently, we used the Netscape driver to create an online representation of "case report form tabulations," which are reports representing all the information we have collected in a study, by patient. This use of Netscape as an "IntraNet" document browser allowed us to furnish a paper-less report that would have otherwise filled 5,000 pages for each of the four copies needed. Besides saving a great deal of paper and printing equipment cost, the reports were accessible to anyone in the Clinical Research division, not just the four primary reviewers. Furthermore, the convenience of quickly accessing only the...
information desired, instead of leafing through 100 lbs. of paper, and the ability to print only that section of the report, met with overwhelming enthusiasm from even the non-technical community.

In addition, the use of the Netscape driver to generate SAS reports is a key link to creating a whole suite of Web-browsable clinical reports directly from SAS. Once this facility is available, incorporating some user-interaction through use of the forms feature of HTML 2, one could theoretically build the content and interface of a Computer Assisted New Drug Application (CANDA) system, with very little non-SAS programming, and a great deal of platform-independence.

A LITTLE BACKGROUND

Why has this been so hard in the past, and why is it easy now?

The fundamental principle that gives the Print Driver system the power to create such a wide range of report formats is the introduction of an abstract document description language. I use the term "abstract" here, and later in this paper, because we are referring to a conceptual image of the report, not the printed or electronic representation of it. By the time the report reaches an actual implementation, the structural information is lost. The ability to describe reports in general terms is what makes the Print Driver system fundamentally different from any document format conversion process that begins with plain SAS output.

Unfortunately, the Print Driver system is applicable only to those reports created in a SAS data step. Although there are apparently plans to give SAS procedures the flexibility that the Print Drivers provide, the human interface for information reports is currently limited to the "plain text" that SAS generates by default. Because any SAS programmer, including the author, does not have access to the processing code of standard SAS procedures, we are unable to create a structured report from any of the SAS procedures. For this reason, the technical implementation of the Print Driver system is limited exclusively to DATA step generated reports. We will therefore limit our discussion to this context.

The problem of creating device-independent, transportable documents is not exclusively SAS Institute's. This question is one that the computer industry in general has struggled with for some time. The problem lies in trying to create a representation of a document in the computer in such a way that a paper copy can be created, while simultaneously maintaining the report in a form that's easy to edit, manipulate, and import into other document managers.

Because there are hundreds of formats for the same electronic document—including formats for proprietary word-processing, document publishing, document transport, document storage, and even more formats for printing—it is almost impossible to translate between formats consistently. This is especially true since the features of one format may not be available in another, and vice-versa.

Standard Generalized Markup Language (SGML)

In an effort to provide a once-and-for-all solution to this problem of electronic document representation, the federal government (Department of Defense), and many organizations around the world have adopted an ISO standard (ISO 8879): SGML. While this paper is not about SGML, it is useful to examine the convention it defines, since it provides a frame of reference for a discussion of the abstract reporting language introduced below. If SGML is of interest to you, there is more information available on the World Wide Web.

Although SGML has the advantage of being a superset of all electronic document formats, as well as all formats to come, it is too general to be directly of any use to SAS programmers. In fact, SGML is a language that describes the language that will be used to describe a document. That is, SGML provides only a formal syntax from which a document descriptor language can be created. An actual document in SGML requires two parts, the Document Type Definition (DTD), which describes the types of things that will be found in the document (paragraphs, quotations, hyper-links, pictures, etc.), and the content section that describes the document itself. For example, one well-known SGML DTD is the Hyper-Text Markup Language (HTML), the language of the World Wide Web. It is important to note that HTML is not a programming language, at least as far as a having flow control structures, variables and subroutines is concerned. It is, rather, a document descriptor language which provides an interface between a document concept, and the hardware and software that will be used to display or print that document.

Hyper Text Markup Language (HTML)

In the case of HTML, the DTD does not appear in the web document, but is implied by the tags that it contains. The HTML DTD, and its revisions, are maintained by the Internet Engineering Task Force (IETF). At the time of this paper, the IETF is finalizing version 3 of the HTML DTD (HTML 3). Although the HTML DTD is a separate text document, to the casual user, the only visible facet of the DTD is in the performance of the web browser. That is, the DTD is a list of text types that can be specified under HTML, and therefore should be recognized and handled appropriately by the browser. It is clear to anyone who has created their own web page that the interaction between a document coded in HTML, and the HTML browser can be used effectively without any knowledge of SGML or the HTML DTD.

What Does This Have To Do With The Print Driver System?

The Print Driver system is similar to HTML in that the implementation has been simplified to appear as an interface between the programmer's report concept, and the hardware and software that will be used to render it. There is, however, a lot more going on behind the scenes, and not just in the programming. The Print Driver system is based on an arbitrary document descriptor language, for which an SGML DTD could, in theory, be created. This descriptor language that the Print Drivers imply, has been designed to provide a minimal set of keywords for printing— it is almost impossible to translate between formats consistently. This is especially true since the features of one format may not be available in another, and vice-versa.
Information Visualization

HOW THE PRINT DRIVERS APPEAR TO THE USER

How a Report is Created

When developing a report that will be created from a SAS data set, we start with an abstract concept of the report in our minds. A sketch of the mental representation of the report that we used to create the examples at the beginning of this paper might appear as shown in Figure 5.

Figure 5: Sketch of report layout

If we were to create this report using SAS the standard way, we would probably evaluate the SAS procedures to see if there was one that would conveniently meet our requirements (PROC REPORT might be appropriate in this case). If not, we would turn to the infinitely flexible DATA step, and use PUT statements in order to have absolute control over the appearance of the report. If we chose the latter approach, we would have to calculate the starting position of the title in order for it to be centered, the positions of each cell of data in the table in order to get certain columns to center, and others to decimal-align. After these calculations have been done, we would begin coding a DATA step, instructing SAS exactly where to place text on the page, as shown in Figure 6.

Figure 6: Text report created using traditional DATA step approach

It is important to note at this point, that we have lost all hope of creating a transportable document from the output of this program. By giving SAS specific information on how to format the report, we fail to communicate the structure of the report. For example, by using the "*" feature of the PUT statement, we have already assumed a fixed-pitch font. Furthermore, if we use "*65" to center a value in a table cell that begins in column 59, there is no way for SAS to realize that the programmer is centering the text, and not just beginning a new table column. In any attempt to convert a report from this format to anything else, the user will have to re-enrich the text by somehow describing the document structure after the program's processing has completed. Much like the bleaching and enriching of bread flour, one might ask "why not leave the enrichment in there in the first place?"

The importance of not specifying too much information about the actual implementation of a report can not be emphasized enough. The more general the description, the more portable your document is allowed to be. Although the ultimate result of each file format appears nearly the same, the digital representation of the format may vary greatly. The program in Figure 6 will generate a file similar to that in Figure 1 but the RTF driver will generate output that appears like that in Figure 7, the PostScript driver like that in Figure 8, and the Netscape driver like that in Figure 9. It should be obvious from these examples that any attempt at conversion from the format in Figure 1 to any of these formats would be ambiguous at best.

Figure 7: Output of Print Driver system using the RTF driver

Figure 8: Output of Print Driver system using the PostScript driver

Figure 9: Output of Print Driver system using the Netscape driver

Basic Design

The Print Driver system—in contrast to the standard DATA step—picks up the task of creating a document as close to the programmer's mental image as possible. Instead of declaring the
fixed positions of text on a page (or in a text file), we specify what the text is, and in what context it should appear. It is then the responsibility of SAS—and the Print Driver system—to make the decisions on how that content should be represented on the output device or file format that we have selected. As an example, the DATA step that created the reports used in the examples at the beginning of this paper is shown in Figure 10.

Figure 10: Source code for examples using the Print Driver system

Effectively, we are using the Print Driver system as an interface to SAS, in order to tell SAS the same thing we are thinking in our minds as we create the report layout: "I want a table, with the title..., with 4 columns, the headers of which will be..., this information will appear in the cells..., with this style and justification..." The Print Driver system was designed so that ideally even non-technical people could look at the programming of a report, and have a pretty good idea what that report would generate. One of the spin-off benefits of the Print Driver system is ease of maintenance this document specification language provides. If a non-technical person can have some understanding of the code, then an experienced SAS programmer should find it intuitive.

The most impressive aspect of this coding system is found in the DRIVER parameter of the %PRINT macro on line 5 of Figure 10. This is the parameter that determines the type of output that the entire report generates. It is currently configured to generate PostScript output, but by changing the word "POSTSCRIPT" to "RTF", the same program will generate a table ready to be imported into any popular word processing program.

Syntax Specifications

The Print Driver system was designed to be as intuitive and familiar to the SAS programmer as possible. To achieve this, the interface was created to appear as a sophisticated version of the PUT statement. This analogy is best understood if we consider the PUT statement not as an end in itself, but instead as a request to the SAS system to place text—either on a page or in a document—on our behalf. Similarly, we can think of the Print Driver system as an interface between the programmer and the SAS system in which the programmer makes requests of the Print Driver system, which, in turn, makes requests of the SAS output system (PUT statements) on our behalf, appropriate to the driver we have selected.

Because these enriched PUT statements will be responsible for a wide range of tasks, each task is identified by the procedure name specified in the PROC parameter of the %PRINT macro. Again, these procedures are analogous to the elements, or tags, of an SGML document. A list of the procedures currently available appears in the table below. Although this set of procedures adequately services the majority of our reports, it is not necessarily authoritative, and could be enhanced or modified at any time.

<table>
<thead>
<tr>
<th>PROC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUTE</td>
<td>Re-direct output during processing</td>
</tr>
<tr>
<td>LINK</td>
<td>Create a hypertext reference to another part of the document</td>
</tr>
<tr>
<td>TABLE</td>
<td>Begin table reference (Instantiate a table)</td>
</tr>
<tr>
<td>TABSET</td>
<td>Declare tab stop positions and type</td>
</tr>
<tr>
<td>TAB</td>
<td>Move to next tab stop</td>
</tr>
<tr>
<td>PUT</td>
<td>Add text to the current context (default)</td>
</tr>
<tr>
<td>SPECIAL-CHAR</td>
<td>Generate a character not available in the standard character set</td>
</tr>
<tr>
<td>TITLE</td>
<td>Create a document title</td>
</tr>
<tr>
<td>LINE</td>
<td>Add a horizontal ruled line (Tables use the LINES parameter; see the table below)</td>
</tr>
<tr>
<td>TITELINE</td>
<td>Specialized sub-title</td>
</tr>
<tr>
<td>NEWLINE</td>
<td>Release the current output line</td>
</tr>
<tr>
<td>NEWPAGE</td>
<td>Release the current page</td>
</tr>
<tr>
<td>NEWRROW</td>
<td>Release the current row (Table mode only)</td>
</tr>
<tr>
<td>ENDTABLE</td>
<td>Close table mode (end a table)</td>
</tr>
</tbody>
</table>

For each of the procedures that are available, the user may specify some optional parameters. These parameters are used to qualify the request made by the procedure specified, and are implemented as macro parameters in each call to the %PRINT macro. Figure 12 lists the parameters available to the %PRINT macro call that control the results of the procedure specified in the "PROC" parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Positional)</td>
<td>specifies content (usually text) of the print request</td>
</tr>
<tr>
<td>COLUMN</td>
<td>specifies horizontal positions and table columns</td>
</tr>
<tr>
<td>JUST</td>
<td>controls text justification</td>
</tr>
<tr>
<td>STYLE</td>
<td>controls text appearance</td>
</tr>
<tr>
<td>LINES</td>
<td>specifies line styles and table borders</td>
</tr>
</tbody>
</table>

Figure 12: Parameters of the %PRINT macro
Information Visualization

One Line at a Time

The processing of the Print Driver system maintains the line-oriented output paradigm that is standard in DATA step based reports (excluding the # modifier of the PUT statement). This means that a line of text is created and issued before the next line is created. Once a line of text is issued, it cannot be later modified. Although this leads to forcing the programmer to anticipate some aspects of the report's output, it removes some ambiguities, and improves program efficiency. Unlike the PUT statement, however, the Print Driver system handles output lines by default, until they are explicitly released with a NEWLINE procedure.

HOW DOES IT WORK?

There are, of course, any number of ways to implement the driver system described in this paper. This section describes the approach used to develop this system, some of the things that have been done to optimize it, and some of the improvements that have been made, as it has gone through a number of iterations.

Virtually all processing in the Print Driver system is done within the %PRNTINIT macro. Although invocations of the %PRINT macro serve as the conduit through which the user's print requests are routed, the %PRNT macro simply resolves to a LINK to the processing logic that was included during the invocation of the %PRNTINIT macro. It is the %PRNTINIT macro, therefore, that contains the core of the Print Driver system.

The first block is the request parser, responsible for taking print requests from the user, identifying them, then calling the appropriate procedure in the driver code. The second block is the actual driver specified by the user in the DRIVER parameter of the %PRNTINIT statement (as in line 5 of Figure 10). This code block is responsible for the actual implementation of the requests the user has made, in the format the user specified. A diagram of this processing flow is shown in Figure 13.

The Request Parser

The request parsing block is given the responsibility of breaking a print request into its constituent parts. Although this section of code will not produce any output, it can be thought of as the place where the abstract document definition is realized. Figure 14 shows a small segment of the request parsing block which contains the code for processing the procedures "Title", "NewLine", and "NewPage".

![Figure 13: Diagram of processing flow of Print Driver system](image)

Functionally, when the user invokes the %PRNTINIT macro, it includes and executes the DATA step code that initiates the Print Driver environment. This includes variable and array declarations, length and retain statements, global macro variable definitions, and other initialization logic. The macro then %INCudes two blocks of DATA step code, but does not execute them. Instead, the %PRNTINIT macro uses a "GOTO label" construct to "skip over" this section of the code. The blocks are skipped over because they contain the subroutines that handle the print requests that will occur later in the code.

![Figure 14: Segment of the request parser](image)

The request parsing block does not have to be dynamically included, as all drivers use the same parsing block. However, this implementation of the Print Driver system includes a "Debug" driver which does not produce a specific image of the report, but instead is used to identify logical errors in the %PRINT macro calls. For example, it does not make sense to end a table unless one has been created, nor can one enter text into column 8 of a 5-column table. Because some of these types of logical errors can be found only during instruction parsing, a separate request parsing block was implemented to address the Debug driver's special needs.

The Drivers

The other section included during the %PRNTINIT macro invocation is the driver code itself. The driver consists of "LABEL", "RETURN" blocks that act as subroutines to handle each type of print request. Each driver has the same set of subroutine labels, although some of the subroutines may be without code because the device either does not support that feature, or it may be implied within the context. This compliance with universal subroutine labels is what allows the Print Driver system to freely interchange drivers at will. The code block LABELs are like the connectors on an interface card on your PC. Each interface card has the same number of potential connectors, although many cards do not use them all. Figure 15 shows a small segment of the TEXT driver.
Follow the Bouncing Ball...

The best way to illustrate the operation of the Print Driver system is to follow the processing logic of a single %PRINT request. This tour of the code uses the call to the "NewLine" procedure that appears on line 32 of Figure 10 as an example. Lines 30 and 31 of Figure 10 contain code in which we created a "line" of text with two calls to the PRINT macro's PUT procedure. In the next line, the "NewLine" procedure releases that line of print in order to begin describing the next one. This %PRINT macro call would resolve to a LINK to the request parser that was included in the %PRINTINIT macro call. The processing continues down the request parser's list of available procedures until it comes to the section of code assigned to process the NewLine request, as shown on lines 3-19 of Figure 14.

In lines 4 and 5 of Figure 14, the "NewLine" code in the request parser block will check the first parameter to see if more than one (the default) new line will be created with this request. Lines 6-9 process the "NewLine" request within the context of a table (this segment would be called on to process the "NewLine" request generated by line 13 of Figure 10). In this case, lines 10-18 would be called upon to process the request. Line 11 LINKs to _SHOWTXT (not shown, ironically), a section of the request parser that is responsible for rendering the completed output line. The _SHOWTXT subroutine LINKs, in turn, to the _OUTPUT subroutine of the specified driver. Each driver has its own version of the _OUTPUT subroutine. The _OUTPUT subroutine for the TEXT driver is shown on lines 20-32 of Figure 15.

After the current line is issued, line 12 of Figure 14 LINKs to the _NEWLINE subroutine in the appropriate driver. The TEXT version appears on lines 10-14 of Figure 15. In the case of the TEXT driver, the implementation is pretty straightforward. We simply issue as many PUT statements as were specified by the user in the first positional parameter of the original %PRINT macro call.

Why all the LINKing and Indirection?

It may appear that the use of %INCLUDED code, and the extensive use of LINKS is more complex than useful. The original implementation of the Print Driver system was more straightforward. It would parse and process each print request within the %PRINT macro. Furthermore, all the drivers were contained in the same macro, and the appropriate code was selected using %IF %THEN logic. This approach proved perfectly acceptable when only one driver was available, but as more drivers were added, the source code became unmanageable. Besides the simple code management issue, there was an efficiency issue created by including the source code for all available drivers, although only one driver could be used in any one report.

Why use separate %INCLUDE blocks?

In an effort to try to make each driver more independent, I separated the drivers into individual files that could be included at run time. Although this made the code for each driver more manageable, the result was a great deal of redundancy between the drivers themselves, as each driver was responsible for parsing the print requests itself. This led to the problem of having to copy any modifications to the parsing code to each driver. This leads me to extract the generic code responsible for the interpretation of the print requests, and put it in a single file referred to above as the "request parser".

Where is all the error-checking code?

Although error checking code could have been built into the request parser and the drivers, the Print Driver system is generally used in a batch-based, "production" environment. This means that the drivers are often used to generate reports that have already been verified to be syntactically correct, so little advantage is realized in having the applications impeded by a lot of error-checking code. In the interest of maximizing efficiency in an application that may potentially draw a large amount of computer resources, the error-checking code was stripped from the standard processing code and assigned to a separate, specialized driver (the "Debug" driver). The drivers themselves, therefore, are relatively short. The drivers shown in this paper average only 192 lines of formatted, 72-character code. The standard request parser is 322 lines.

The Scope of the Print Driver System

Of course, developing a system of drivers would be a lot of work if you were only creating one type of report. The true payoff of the Print Driver system would not be realized unless it was used as a resource in an environment where a wide range of reports are necessary, and each of these reports could end up in any number of electronic formats.

Even if you could use the drivers in a wide range of reports, it may still seem complex and cumbersome to create a single driver, and even more so to create a set of drivers. Fortunately, it is actually much simpler than it first appears for two reasons. First, the drivers do not need to be able to independently generate every conceivable report. That is the job of the specific reporting application. Second, any driver developed needs only to implement (support) those procedures deemed necessary.

The Print Driver services are much like the utilities that are provided in most homes. Just as it would be impractical for each home to generate its own electricity, it would be impractical for any single reporting application to be expected to be able to generate reports in the variety of file formats that are available with the Print Driver system. Also, just as the utilities are not responsible for creating, maintaining, and using electrical home appliances, neither is the Print Driver system responsible for independently creating, maintaining and using any single report.
Information Visualization

The Print Driver system is only responsible for supplying the "power" to a programmer to conveniently implement the report that they need.

CONCLUSION

The most important point to take from this paper is the fact that generalized document conversion is simply not possible using the tools currently available in the SAS system. While there have been some excellent procedures developed for converting the text that SAS produces to a specific type of output, there will never be an automated tool that can convert output to an arbitrary set of output devices.

SAS is working on making their procedure output more transportable. Meanwhile, the Print Driver system offers an extensible means for providing this transportability for reports generated within a DATA step.

Print Driver System: The Swiss-Army™ Chainsaw of Reporting Tools

The real power of the Print Driver system is found in its extensibility. The drivers can be expanded in both flexibility (Swiss-Army), and in power (Chainsaw). If more flexibility is needed, that is, a service not currently provided, that service can simply be added to the list. In some cases, this can be done without modifying any of the specific drivers. For example, the current drivers have no procedure for representing the concept of a table heading, or of page headers or footers. They also still require the programmer to calculate column widths, and re-format [flow] text in advance. Adding these capabilities, as well as any other requirements, is simply a matter of implementing the interface with the driver system.

If more power is required, meaning the ability to generate output for a device not currently available, a programmer needs only implement a driver for that device, and the report then inherits that functionality. If needed, drivers for popular output languages such as TeX, PDF, SGML, Windows DDE, and PC-printer languages like PCL-5 and Epson, could all be implemented at any time. In fact, once a driver is created, all reports created using the driver technology—even those that have not been touched in years—automatically inherit the ability to generate reports in that new format.

The power to expand, in addition to the drivers and interface already in place, really does serve to teach SAS to speak virtually any document publication language.

TRADEMARKS

SAS and SAS/GRAPH are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

PostScript is a registered trademark of Adobe Systems Incorporated.

WordPerfect is a registered trademark of Novell Inc.

Microsoft is a registered trademark of Microsoft Corporation.

Netscape and the Netscape Navigator are trademarks of Netscape Communications Corporation. All rights reserved.

Swiss Army is a registered trademark of Victorinox.

CONTACTING THE AUTHOR

Copies of this paper and source code for the Print Driver system are available from the author:

Paul Wehr
2502 Traver Road
Ann Arbor, MI 48105-1249

home: (313) 665-2505

wally@eworld.com

wehrp@aa.wl.com

Reprints of this paper are available from http://pages.prodigy.com/paul_wehr/sugi.htm

REFERENCES

1 e-mail discussion list SAS-L send the message "sub SAS-L your name" to listserv@vtvm1.cc.vt.edu.

2 USENET news.group.comp.soft-sys.sas URL=news:comp.soft-sys.sas.


