Use of the transform exit sequence in printing and as a framework for the solution of complex problems

by S. D. Mastie

This paper is both a review of previous research and development and a description of new work for the IBM PSF/2 and PSF/6000™ printing products. These products are best characterized as complex queue drivers, originally designed to allow fast printers to be driven directly from the existing print subsystems of OS/2® and AIX®, respectively. The new work extends the capability of these products by providing a framework that broadly redefines the way in which "print queues" can relate to printers, other queues, and other programs, without requiring any changes to the spooling system. This technical advance is based on the twin ideas of the transform exit and the transform exit sequence, which are examined in detail. Using this technology, both PSF/2 and PSF/6000 treat the largest mainframe-attached printers as "local desktop printers," allowing LAN communities to print directly on mainframe printers from LAN applications. Other interesting uses of this technology include routing print jobs to the printers appropriate for their data types, improvements in printer throughput, automated print-job archiving, and improved system management, each of which is covered in some detail in the paper. Although some of these solutions can also be provided by extending the capabilities of the spooling systems and some have been addressed in particular environments and applications, the transform exit sequence framework does not depend on changes to the spooling system, allowing it to be used in a wide variety of current and future operating systems.

Introduction

Historically, printers and print queues have often enjoyed a simple relationship: one printer per queue, and one queue driver to bind the two together by sending print jobs from the queue to the printer. As long as the data type of the job on the print queue is one that the printer

0018-8646/97/$5.00 © 1997 IBM

Table 1 Abbreviations used in the paper.

AIX:	Advanced Interactive Executive (IBM UNIX)
AFP:	Advanced Function Presentation (architecture specifying document structure and page content)
API:	Application Programming Interface (interface to application program)
ASCII:	American Standard Coded Information Interchange (a code for unformatted print files)
IPDS:	Intelligent Printer Data Stream (two-way printer dialog and page-description language used by IBM printers and PSF products)
LAN:	Local area network
JES:	Job entry subsystem (IBM MVS print job queuing system)
MVS:	Multiple virtual storage (mainframe operating system)
NT:	New Technology (a Microsoft operating system)
OS/2:	Operating System/2 (an IBM operating system)
PDL:	Page Description Language (describes data format)
PS:	PostScript (page description language developed by Adobe)
PSF:	Print Services Facility (queue driver for IPDS printers)
RIP:	Raster image processing (creating images from PostScript)
SNA:	Systems Network Architecture (often used for cross-system distributed printing)
VM:	Virtual Machine (mainframe operating system)
VSE:	Virtual Storage Extended (mainframe operating system)

can handle, this paradigm allows printers to be unaware of the specifics of the operating system, application program(s), or queuing system. Not surprisingly, this paradigm is very pervasive: One finds it on all of the major IBM operating systems, including MVS, VM, VSE, AS/400*, AIX*, and OS/2*, and most major non-IBM operating systems as well, including Windows**, WIN-95**, NT**, NetWare**, Banyan Vines**, and many varieties of UNIX**. (**Table 1** provides a list of abbreviations common to the printing literature.) In the case of the IBM operating systems, sophisticated Intelligent Printer Data Stream* (IPDS*) printers could be attached by the use of a Print Services Facility* (PSF*) software product as the queue driver. This allowed continuous two-way printer communications for the first time but still left the paradigm intact: There was usually one queue per printer, and PSF bound the two together.

The PSF/2 introduction of the transform exit and transform exit sequence extended this. This new technology enabled PSF/2 to provide important new solutions to otherwise unsolved printing problems, and a clean way to address issues such as forms scheduling, which were previously solved only in specific operating system environments. It has already been ported to AIX in the PSF/6000* product and can be applied to other operating systems as well.

The transform exit provides four essential functions: data-type detection, open API support, substitution variables, and a "terminal" designation. Data-type detection (often called "sniffing") allows conditional processing by data type. Open API support allows practically any program or filter to be executed by the transform exit when print jobs of the appropriate data type(s) arrive on the print queue. Substitution variables allow APIs to be invoked in an operating-system-independent way, by externalizing information as variables that are dynamically resolved for each print job by the transform exit before calling the API. Designating a transform exit as "terminal" allows the API invocation to be one-way, completing the queue's processing of the data in lieu of sending the data to a physical printer.

The transform exit sequence provides rules for linking individual transform exits into a sequence to be executed, so that the output of one transform exit can become the input to another. This allows individual transform exits to be linked in an object-oriented manner as building blocks of complex solutions, unaware of the broader context in which they are imbedded. The transform exit sequence is a new programming language, in which the individual "instructions" are transform exits that are called in sequence. The input to a transform exit sequence program comprises the print jobs that arrive on the print queue.

• *Organization of paper*

First we present a historical overview of the printing paradigm and relevant prior art, in order to establish the terminology and framework for evaluation of the new work.

The main body of the paper, which follows, is composed of three broad sections: the transform exit, the transform exit sequence, and applications of the technology. The first of these sections, on the transform exit, focuses on the technology, terminology, and functions of individual transform exits. The expressive capability of the transform exit is shown in terms of problems that can be solved by the use of a single transform exit. The second section, on transform exit sequences, advances the ideas of the transform exit by providing the rules and framework for linking multiple individual transform exits. The final

section, on applications of the technology, presents sample applications of the transform exit and transform exit sequence.

The concluding section suggests areas needing further study.

• *Historical overview of the print paradigm, and prior art*

While many of the concepts and terms used throughout the paper are nearly universal in nature (an example is "print queue"), the OS/2 desktop is used wherever possible to specifically illustrate the concepts. The reader is invited to generalize beyond this, to other environments, as appropriate.

Printing can be decomposed into three high-level steps:

1. Formatting the data.
2. Moving the data to the print queue.
3. Moving the data from the print queue to the printer.

The first step is generally performed by an application program, which uses a print driver to help create data the printer can understand. The second step is done by the print driver and the spooling system of the particular operating system to get the print job onto the print queue. The final step is the delivery of the print data to the printer by a queue driver. The queue driver is controlled by the spooling system. For example, if the queue is held, the queue driver is typically not called until after the queue is released. Pictorially, the print driver, queue, queue driver, and dataflow relationships can be represented as shown in **Figure 1**. Each of these steps occurs during printing, but they can be overlapped and interrelated, reflecting the great variety among printers, print and queue drivers, and application programs.

We first consider the printer. From its perspective, sometimes called the "printer's-eye" view, information arrives which comprises printing controls and data to be processed and printed. Printers can understand only some of the many possible formats of data, generally called page description languages (PDLs). Among the more common ones are HP:PCL** (Hewlett-Packard's Page Control Language, for HP** LaserJet** and compatible printers), PS (Adobe's PostScript** language), and IPDS (IBM's Intelligent Printer Data Stream). Generally, these PDLs organize the pages and the data within the pages so that the printer knows how to create the page images. Each page image is composed of pixels (also called pels, for picture elements). Some printers support only one PDL; most newer printers support a wide variety of them, along with common forms of simple, unformatted data without explicit page boundaries such as ASCII files and MVS 1403 data. Unfortunately, no printer can support all of the types of data that exist; the universe of possibilities is immense and growing constantly. If one sends data to a printer in a PDL that it does not understand, it will probably produce many pages of garbage, even for a small file, as anyone who has sent the wrong type of data already knows.

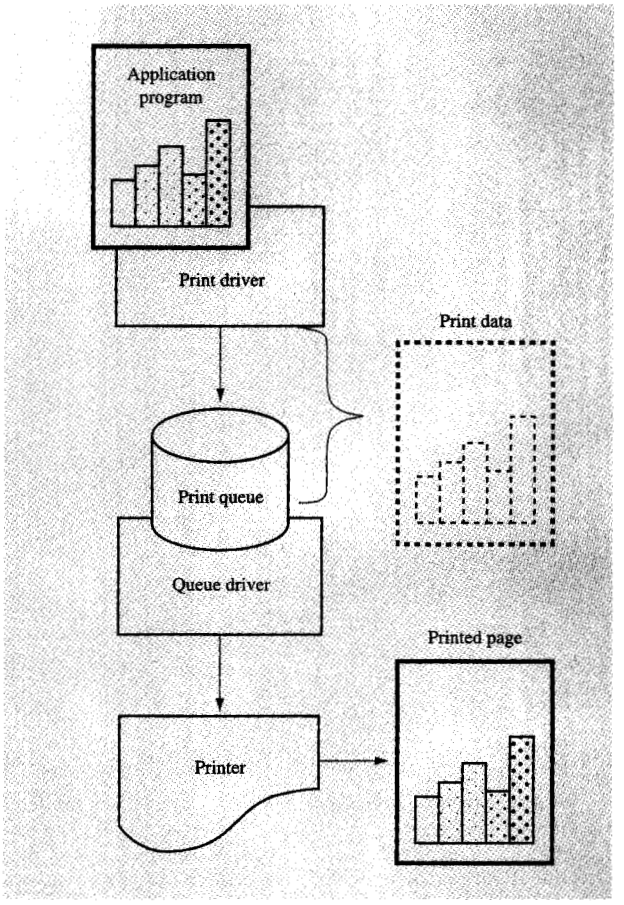

Figure 1

Basic printing process.

The fact that each printer understands only a limited number of PDLs has profoundly shaped the historical printing paradigm. Because the universe of possible data types is continually increasing and printers need structured PDLs to render the pages properly, the evolution of most queue drivers has led to a very simple approach: They merely take print jobs from the queue and send them directly to the printer, even if the printer cannot understand the PDLs of some of the print jobs. Figure 1 shows such a queue driver and how it fits into the overall printing process.

Not that all queue drivers are simple, by any means. PSF/MVS, for example, can be viewed as a queue driver for the MVS JES (Job Entry Subsystem) spooling system, connecting to IPDS printers via S/370* channel, SNA, or

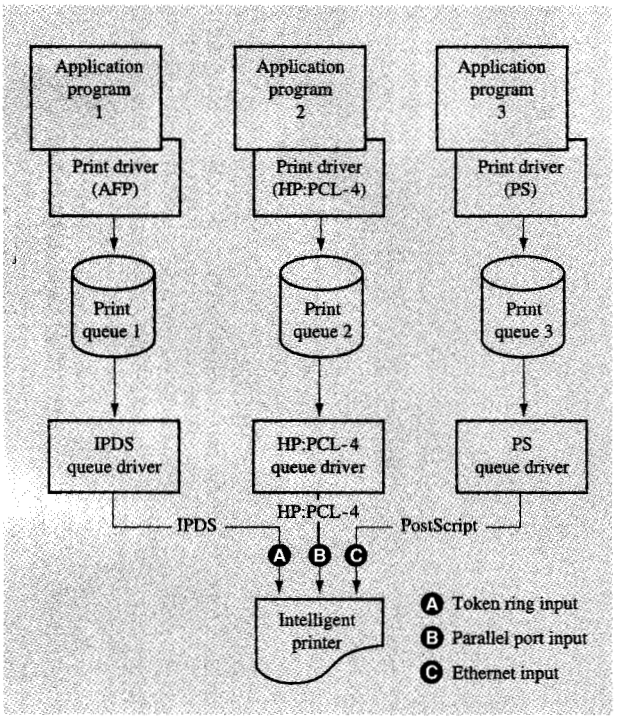

Figure 2

Intelligent printer receiving three kinds of print data from three different applications, print drivers, and print queues. The data arrive via three different communication technologies and three different drivers. Any single application-to-printer path is the same as shown in Figure 1.

TCP/IP protocols. Recognizing the importance of having the data formatted correctly on the printer, the PSF/MVS designers established a new way of dealing with the universe of possible data. In addition to *preventing* data from printing incorrectly whenever possible, PSF/MVS introduced post-spool formatting of data; i.e., the queue driver may change the data before delivering the data to the printer, as opposed to pre-spool formatting by the application and print driver, which is still more common. This technology allowed unstructured data such as fixed-length, 72-column records to be composed into more aesthetically pleasing output, without requiring any changes to existing mainframe application programs that generated the original data. Not only does this allow print format changes without application program changes, but it also allows the printer to receive a single type of PDL (IPDS in this case) from the queue driver, even though multiple forms of data arrive on the queue. This important advance, driven by the printer's-eye view, added great utility to printers, while keeping the printers themselves relatively simple [1]. Still, it generally conformed to the old paradigm: a single queue (JES) and a single queue driver (PSF) feeding a single printer (IPDS), so that the printer can maintain its myopic printer's-eye view of the universe of possible data types. Also, any data type not recognized and handled by the queue driver still results in either no output or incorrect output.

Printers have evolved and advanced greatly since the development of PSF/MVS, and newer printers generally accept and understand many PDLs. In addition, the "smartest" printers have a limited ability to detect the PDL type of each print job and adjust their emulation modes accordingly. Known as AES (Automatic Emulation Switching) [2], this advance has been extended to allow some printers to handle multiple PDLs arriving simultaneously from different input sources. **Figure 2** is a diagram of such an intelligent printer. Note that this allows multiple print queues to feed a single printer, but each queue is still bound to a physical device, as is each queue driver. Other advances in spooling systems allow a single queue to spread jobs across multiple queue drivers [3]. This one-to-many relationship between a queue and physical printers is generally called a "printer pool." A printer pool does allow an application program and print driver to print to any of a collection of printers, but each queue driver is still bound to a physical device, with few or no intelligent criteria to control the distribution of the jobs: Each of the n printers gets every nth print job, even when one of the printers becomes disabled, and there is no capacity to add adjustable, user-defined criteria such as allowing large jobs to go only to a particular printer, or all jobs of a certain data type to a particular printer that can handle that data type. These ideas are refined significantly by the ISO DPA 10175 work, which adds significant intelligence and capability to the spooling system itself [4]. Implementations of this work such as the IBM PSM (Printing Systems Manager) product for AIX, or the Xerox PrintXchange product, are able to do intelligent print pooling based on flexible routing criteria. ISO DPA 10175 actually extends the printing paradigm significantly by redefining what the functions of the spooling system can be, in order to permit interoperability and exchange of print jobs between heterogeneous printing systems. However, even this new work leaves the model for the queue driver portion of the paradigm intact: From the perspective of any single queue driver, there is still a single queue of print jobs and a single printer to send them to. This attests to how profoundly the need to print, and the ensuing queues, queue drivers, and PDLs, have shaped the way our computing systems look and behave.

The reach of this printing paradigm for queue drivers extends even beyond the queue. As has been stated above, in general the data that are put on the queue are sent directly to the printer by the queue driver. Therefore, inappropriate data types that reach a print queue still fail.

This has propagated the responsibility for producing valid print data back to the clients and application programs, which has led directly to the development of the *print driver*. A print driver generates properly formatted data for a particular printer directly from the output of an application program. Some applications still do such formatting themselves, but with advanced LAN operating systems such as Windows and OS/2, a framework is provided that allows all print drivers to be available to all of the application programs running on the desktop. This means that print drivers now offer consistent PDL generation from practically any application program: The printer manufacturer creates an appropriate driver for each new printer, and the operating system provides the underpinnings that allow this print driver to be loaded and used consistently from any application program. This has certainly advanced the state of the art, since it allows all desktop application programs to format and print correctly on a particular printer. However, just a quick glance at the sheer numbers of LAN print drivers for Windows or OS/2 tells a critical tale: Each of these drivers is subtly different in order to create a PDL that is acceptable to its particular printer. It is practically universal today that when the carton of a new printer is opened, a diskette is found inside that provides the new print driver that is needed. The good news is that this paradigm advance has freed the application-program writers from worrying about the printers; all of the print drivers are available to all of the application programs via the homogeneous printing interfaces provided by the operating system. In theory, this enables all printers to be available from all application programs. In practice, however, it still falls to the user to select a print driver that is appropriate for the target printer or get garbage output.

The mainframe analogies to this are very clear as well. At one time, all mainframe application programs did their own internal formatting, which required the application programs themselves to be changed for each printer or page-layout change. This problem was mitigated by the addition of post-spool formatting [for example, by a queue driver such as PSF using an AFP* (Advanced Function Presentation*) Pagedef and Formdef], to protect mainframe software programs from change by insulating them from the details of formatted page layout. Known as conditional processing, the ability to change page layout and support new printers *without* requiring changes to the software applications is to mainframe software what print drivers are to workstation software. In each case, application programs can make use of new printers and formatting options without being changed themselves.

This concludes the overview of the print paradigm, with enough history to allow the evaluation of the new technology of the transform exit and transform exit sequence and how these have been implemented in PSF/2 and PSF/6000. That the basic paradigm for printing holds for nearly every computing operating system, and that it allows print jobs to be printed successfully on devices from low-end dot matrix printers to high-end laser printers testifies to its flexibility. However, pervasive problems remain in getting correct output, and the technology of the transform exit sequence is a new approach to dealing with many of these print-related problems. The transform exit sequence technology can be implemented completely within the queue driver as it has in PSF/2 and PSF/6000. This makes it widely useful as a technology, since it can be implemented within existing operating systems and spooling systems. This also makes the technology completely complementary with pre-spool conversion techniques such as the filters of UNIX and AIX, and broader spooling system extensions, such as the ISO DPA 10175 standard.

The transform exit sequence addresses the problem of inappropriate print data being sent to a printer by allowing such data to be detected and handled gracefully: If one sends PostScript to an HP:PCL printer by way of a PSF/2 or PSF/6000 queue driver, the PostScript is converted by means of a transform exit sequence into the HP:PCL data that the printer can handle. The conversion can be done either locally or on a remote machine, as preferred. Although the filters of UNIX and AIX provide broad data transformation capability, they are different from the transform exit sequence. Filters are normally invoked explicitly from the command line to do a particular data conversion before the data are enqueued for printing. This should be contrasted with the post-spool nature of the transform exit sequence, which allows appropriate conversions to be done automatically and *conditionally* when a print job of a particular data type arrives on the print queue. Also, the transform exit sequence can be applied to operating systems where filters are unavailable.

Another variation of data-type awareness is exposed by the following problem: How can a print administrator set up a single queue for a networked user community so that the queue routes print jobs of different data types to the physical printers capable of handling them? Addressed in ISO DPA 10175, this problem may also be solved by means of a transform exit sequence in PSF/2 or PSF/6000, so that all PostScript jobs that arrive are sent to one printer, all HP:PCL jobs that arrive are sent to a second printer, and so forth. Since the transform exit sequence operates post-spool within a queue driver, it has no dependencies on spooling system changes or the specific capabilities of a single operating system, making this class of data-type routing problem solvable in a wide variety of existing environments.

Printers have a small number of input bins (also called trays). Many printers support dozens of different media

sizes and weights, but only one medium type at a time can be loaded in a given bin. This fact leads to another illuminating problem: How can a print administrator make available to the user community all of the media that a printer actually supports, while making sure that if a particular medium is not loaded when it is needed, the job that needs it is held (until the medium becomes available again) rather than printed on an incorrect medium? A closely related scheduling problem is this: How can a print administrator assess the number of print jobs that are held awaiting the availability of a particular medium that is *not* currently loaded in the printer (in order to efficiently schedule media changes) or obtain an enumeration of all of the media types required by pending print jobs? These related problems are outside the scope of what can be addressed by means of filters. They can be addressed by means of an ISO DPA 10175 approach, but they can also be solved by the use of transform exit sequences to define multiple logical queues for the single physical printer. This allows administrators to manage devices effectively by releasing only those queues that match the media actually loaded in the printer, even in environments where a new spooling system is inappropriate. Note also that the arrival of a print job in a particular queue can be used as the criterion for an alert to the administrator, providing an automatic indication that a medium change will be required to handle a particular print job. This application is discussed in more depth in the last subsection on multiple logical queues later in the paper.

Most significantly, the transform exit sequence is an extensible framework that permits additional solutions in the future. Current uses of the technology include conversion of print jobs and then uploading them from desktops to mainframe printer queues, conversion of PostScript files in the background to avoid printing delays on the physical printer, and calling application programs so that the user may view the print jobs before they are printed or in lieu of printing them at all. Each of these solutions is examined in the section on applications. For some of these solutions, a "print queue" becomes merely a front-end queuing system to the transform exit sequence framework and is used to enqueue, convert, and redirect data for purposes not related to printing *per se*.

Transform exit

Since the transform exit can be viewed as an extension of the concept of filters, we begin by reviewing filters. Filters, which allow the conversion of data from one type to another, are widely used in UNIX, AIX, and other workstation environments to enhance the utility of particular printers. Assume that a filter called "AtoB" converts data type A into data type B. If a particular printer accepts data of type B only, this AtoB filter can be used to transform (or "filter," in the vernacular of UNIX) data of type A into type B before the data are enqueued, allowing the data to print. If we assume that such a printer is attached to local port LPT1 and the data of type A are located in the file c:/typea.out,[1] the data can be printed if the following command is issued:

AtoB c:/typea.out > LPT1.

This invokes the AtoB filter, supplies the file of type A data as input, and pipes the result of the filter (type B data) directly to local port LPT1.

The transform exit extends the ideas of filters by allowing filters and other programs to be invoked *selectively*, on the basis of data-type criteria, whenever an appropriate job arrives on a print queue. The specification of a transform exit has the following form:

transform exit = *name, terminal,*

(*data-type qualification*;), *body*.

The *name* of the transform exit must be unique. Although it is not strictly required, it makes keeping track of multiple transform exits simpler.

If the transform exit is flagged as *terminal*, no further processing will be done on the print job after the body of the transform exit is executed. The Boolean terminal indicator is necessary to solve some classes of problems. It is false by default, meaning that the queue driver will normally attempt to print the results of the execution of the transform exit body. If it is set to true, the transform exit is called a "terminal transform," and the queue driver will not attempt to print the job after executing the body of the transform exit.

The *data-type qualification* is an optional logical expression that, if present, causes the transform exit body to be executed for a print job if and only if the data type of the print job matches the expression. The expression can be a single data type or multiple data types combined using logical AND(&) and OR(!) operators. The data-type qualification is expressed with the use of tokens to represent the kinds of data that must be detected. Within PSF/2, the following tokens are valid: AFP, ASCII, PS, PCL, and METAFILE. These can be combined using logical operators, for example, to define a transform exit which is to be executed if a print job is either of two different data types. If any data-type qualification is given, a delimiter must follow and precede the body.

The *body* is the filter or program to be executed by the queue driver if any data-type qualifications are met for a particular print job. For example, this might be a filter to convert the data to another form. With regard to the filter example AtoB introduced earlier, AtoB requires the fully qualified file name as input. Since the actual file will

[1] We use UNIX notation here, since filters are widely used in UNIX.

vary with each new print job on the spooling system, substitution variables are provided by the transform exit to allow tokens to represent data that vary. These tokens are resolved by the transform exit before the body of the transform exit is executed.

As an example of use, we define a transform exit to put the existing AtoB filter to work, so that it will be called automatically by the queue driver for any PostScript job that arrives on a print queue but will not be called for any non-PostScript print job. If it is called, we want to print the result of the AtoB transformation. To call AtoB, we need two substitution variables: one ("%i" in PSF/2) to tell AtoB where to access the current print job if it matches the PS-only criteria, and another ("%o" in PSF/2) to tell AtoB where to put the filtered data so that the data can be printed. The substitution variables %i and %o are for input and output, respectively. Then, if AtoB is located in the c:\filters subdirectory, the following transform exit specification can be used:

transform exit = AtoB for PostScript Only, False, PS;

c:\filters\AtoB %i > %o.

Once a transform exit is defined, it may be associated with multiple print queues—it does not have to be redefined for each queue.

A transform exit is run when a print job arrives on a queue with which the transform exit is associated. All transform exits must be executed by the queue driver before any data are sent to the printer, because any transform exit can change the data, or even pass control of the printing process to a different application program or print queue if it is a terminal transform.

This actual execution of a transform exit involves four essential parts, discussed below: data-type detection, open API support, substitution variables, and terminal transforms.

- *Data-type detection*

During transform exit execution, data-type detection is the ability to inspect a print job on the print queue, associate it with some member of a set (all members of the set constitute the universe of all known data types), and compare it to the data-type qualification (if any) of the transform exit definition. This concept requires that all print jobs be classifiable into some taxonomic hierarchy of data types. For reference, the following are the data types that PSF/2 version 2.00 can detect:

- AFP ≡ any Mo:DCA-P print job [5].
- ASCII ≡ any ASCII print job.
- MF ≡ any OS/2 metafile print job [2].
- PS ≡ any PostScript print job [6].
- PCL ≡ any HP:PCL-4 or HP:PCL-5 print job [7].
- OTHER ≡ none of the above.

Then,

U = {AFP, ASCII, MF, PS, PCL, OTHER},

where U is the universe of all possible data types. The data type OTHER includes all data types other than those for which data-type-detection heuristics or algorithms have been implemented.

Classification according to some sets U, including this one, involves the resolution of ambiguity, since some print jobs can be associated with multiple data types. PostScript print jobs, for example, are often ASCII files that contain the PostScript programming language [8]. Therefore, when both ASCII and PostScript are members of the set U, the ambiguity must be resolved. In general, ambiguity between any two or more members of U must be resolved as part of data-type detection.

The approach implemented in PSF/2 was to remove any ambiguity by associating each print job with one and only one member of the set U. Another approach is to allow classification of a print job as a subset of the members of U, for example, {PS, ASCII} in the case of an ASCII print job that contains PostScript. For PSF/2, however, print-job accounting requirements (the total number of pages printed for each data type) and other transform exit applications were best served by associating each print job with only a single data type. Therefore, the implications of subset classification are left as future work, and henceforth, data-type detection is assumed to be the classification of a print job as one and only one member of the set U.

An ASCII file that contains PostScript can be treated as PostScript (the PostScript program should be run to produce the appropriate images) or treated as ASCII (which results in a listing of the PostScript program). If explicit information is available indicating which of these two output formats is desired, PSF/2 makes use of that information in lieu of detection. In the absence of such information, any job on the print queue that tests positively as both ASCII and PostScript is classified as PostScript only.

In summary, a given transform exit can be unconditional (applicable to all data types) or data-type conditional. Unconditional transform exits can be executed immediately without data-type detection. Data-type-conditional transform exits require that the data type of each print job be determined and compared to the data-type qualifications of the transform exit, and any ambiguity in the classification must be resolved before this comparison is made. Resolution of the ambiguity between the members of a set U which is different from that used by PSF/2 is left as future work.

• *Open API support*

An API, or application programming interface, is defined as an interface that allows data and/or control to be passed from one application program to another. The transform exit provides a framework for making use of any available API (i.e., invoking any available program) from the print queue. OS/2, for example, allows most application programs that have been compiled to run under DOS, Windows 3.1, or OS/2 to be started by simply invoking them from any OS/2 window. OS/2 also supports invocation of REXX or command language (*.CMD) programs. Therefore, the support for transform exits in PSF/2, which is based on OS/2, allows any of these types of application programs to be invoked from a transform exit, and a "print" queue can now represent almost any desktop application.

As one example, consider the AFP Workbench*, which runs under either Windows or OS/2, and allows documents to be viewed, among other document-related functions. According to default naming conventions, c:\fld\fldwinvw.exe would be the Windows version of the Workbench program. The command

```
c:\fld\fldwinvw c:\myfile.out
```

would start the Windows Workbench in a WIN-OS/2 context from any WIN-OS/2 or OS/2 window and would cause the file c:\myfile.out to be viewed. Therefore, this same command can be the body of a transform exit to build a viewer queue for previewing print jobs.

Similarly, all system commands, such as COPY, PRINT, and TYPE, as well as other application programs, can be included in the body of a transform exit.

Finally, we observe that the transform exit provides an excellent framework for customization. Consider the task of writing a new accounting program that is to add up the number of bytes of the PostScript jobs that are printed on a particular printer. The transform exit approach allows the accounting program to be coded in whatever language the individual prefers, added to a transform exit, and be called for PostScript jobs only.

• *Substitution variables*

Substitution variables are tokens that a transform exit makes available to the body of the transform exit. They represent information about the print job that might be needed by the body of the transform exit. These variables are resolved dynamically for each print job and are automatically substituted into the application program calls in the body of the transform exit. Because the transform exit makes these substitution variables available, many application programs can be accessed via a transform exit much more easily than if they had to be aware of the underlying queuing system that manages print jobs. For example, on OS/2 and most other queuing systems, the identifier assigned by the queuing system varies for each print job. Also, it is often necessary to call an application program from a transform exit in order to do something for every print job (or perhaps every print job of a certain data type). For such cases, a substitution variable is made available to represent the concept of the job (the particular job being printed at this moment), which allows the body of the transform exit to be coded just once and remain constant. The transform exit then resolves this variable appropriately for each print job and dynamically substitutes the correct value into the call to the application program, making the transform exit work consistently for multiple print jobs having different identifiers. Substitution variables let transform exit definitions be specified in terms of variables that are resolved dynamically by the transform exit execution.

• *Terminal transforms*

By definition, a terminal transform is a transform exit that, once executed, results in no further transformations of the data being performed and in the original source print job being removed from the print queue.

A terminal transform can be absolute or conditional. An absolute terminal transform is terminal regardless of the data-type classification of an individual print job, while a conditional terminal transform is terminal for only the type of data specified by the data-type qualification.

The terminal transform, when combined with the other capabilities of a transform exit (the data-type detection, open APIs, and substitution variables), allows "print" queues to represent almost any application program, whether or not it has anything to do with a physical printer. That is because a terminal transform cleans up the queue after the transform exit has passed control of the "print" job to an entirely new application, allowing sophisticated data management function to be made available for print jobs with no cluttering of the spooling system. Thus, the concepts of an archive queue, Lotus Notes* queue, AFP Workbench viewer queue, and pre-flight PostScript viewer queue become possible, each involving transformation and redirection of the data to something other than a printer. All of these queues could have been functionally implemented with the transform exit capabilities examined previously, but without the terminal transform capability, a copy of each print job would be held in the print queue. This characteristic is generally undesirable, as it leaves such print jobs in the queue until they are cleaned up, by manual intervention or a periodically executed cleanup program. While this may be adequate for some tasks, others require that a print queue be "self-cleaning" after print jobs are passed on to the appropriate application program. The terminal transform allows this, by letting the transform exit itself control whether there is any further processing to be

done on the (possibly transformed) print data. A simple example illustrates this: Some applications of transform exits require print jobs to be archived *while* printing, yet others require that print jobs be archived *instead of* printed. The two transform exits required for these disparate functions can be identical syntactically in how they invoke the archive process from the body of the transform exit, but the second one requires the transform exit to be terminal. Being terminal or not is therefore an essential element of a transform exit.

This concept is explored further in the transform exit sequence discussion that follows and is featured prominently in the sample applications section.

- *Transform exit summary*

In summary, a transform exit is defined using a new syntax, is associated with a print queue, and is executed when print jobs arrive on that queue. The actual execution of a transform exit consists of four essential capabilities: data-type detection, open API support, substitution variables, and terminal transforms. Comparisons to the function provided by filters indicate that a transform exit extends the usefulness of filters by allowing a filter to be run for only certain data types, and insulates a filter from the variations among the print jobs on the queue. This replaces the old print paradigm by allowing "print" queues to be defined for practically any task, even tasks having nothing to do with any physical printer, such as archiving or viewing print data. There is prior art for building customized queue drivers for particular tasks not related to a physical printer: for example, building a "fax queue." Also, prior art exists for building "print drivers" that create and file data rather than actually printing data. This new work, however, is different: It allows not just one particular task to be built into a customized print driver, but is instead a framework that allows *any* API to be invoked for appropriate print jobs. The data-type detection allows these APIs to be invoked conditionally for only certain type(s) of data, and the substitution variables allow application programs to be invoked without knowledge of the queuing specifics of the queuing system being used. This permits Windows applications to be called from an OS/2 print queue, for example, without their requiring any knowledge of the OS/2 print job queuing system (or even that the file passed in as input must be a print job). Finally, the idea that a transform exit may be a terminal transform permits the definition of functional queues that clean up the print queue after each print job or keep a copy of each print job, as desired.

Transform exit sequence

An essential idea from object-oriented programming—the reuse of functions by several applications—applies here as well. Why not extend the idea of a transform exit to allow multiple transform exits to be used as functional kernels that can be linked together to form entirely new programs?

To provide this capability, the idea of the *transform exit sequence* is introduced. The transform exit sequence is perhaps best viewed as a new programming language, the individual instructions of which are transform exits. This extension allows transform exits to become building blocks for more powerful applications. A transform exit sequence has the following attributes:

- A transform exit sequence is made up of 0 or more individual transform exits.
- The output of one transform exit in the sequence is the input to the next.
- Conditional transform exits (i.e., those with data-type qualifications) are skipped if their conditions are not met, allowing the output from one transform exit to skip past transform exits that are not applicable and become the input to a later transform exit that is applicable.
- A transform exit sequence is terminated upon completion of the last transform exit in the sequence or upon encountering a terminal transform exit (including a conditional terminal transform exit whose condition is met).

This extension allows filters to be imbedded within transform exits and easily combined with other filters (also in transform exits), thus creating powerful transformation capabilities. Filters can be chained together without transform exits, of course, but putting them together in a transform exit sequence avoids any problems due to inappropriate data types reaching any of the filters. For example, consider the following four transform exits, for which the names, data-type qualifications, and bodies are shown, along with brief comments describing them:

```
AtoB: A; c:\filters\AtoB %i %o
                        /* Convert data type A to type B */

BtoC: B; c:\filters\BtoC %i %o
                        /* Convert data type B to type C */

CtoD: C; c:\filters\CtoD %i %o
                        /* Convert data type C to type D */

DOIT: D; e:\apps\doit %i
                        /* Call doit, with data type D */
```

Note that the name of each transform exit is a mnemonic that represents the transformation or function. Transform exit AtoB, for example, calls filter c:\filters\AtoB if and only if the input data type is A. The result of this filter has data type B, which is returned to the next transform

exit in the sequence. Similarly, transform exit DOIT calls application program e:\apps\doit if and only if the input data type to this final transform exit is D.

These four transform exits, organized into a transform exit sequence in the order given, should be contrasted with a similar solution using linked filters. For example, consider the four filters being linked in the following manner:

AtoB input.fil > BtoC > CtoD > doit.

This calls the filter AtoB to process the input input.fil, the result of which is then piped into BtoC, which in turn pipes its result into CtoD, which finally pipes its result into the doit application program. If the input were of type A, this would perform the desired series of filter operations very well. However, if the input were *not* of type A, but instead of type B, C, or D, in all likelihood, none of these specific filters could handle inappropriate data types. Consequently, the standard solution using the technology of filters would be the use of subsets of this filter chain, one subset for each possible input data type. For example, the subchain BtoC input.fil > CtoD > doit would handle data of type B and would avoid the need for filter AtoB to handle input of type B. In contrast, in the framework provided by the transform exit sequence, all input data types are handled, and none of the filters is called for incorrect data types.

The transform exit sequence allows individual functions to be brought together into integrated solutions. Consider, for example, a printer that handles only HP:PCL-4 input. Because PSF/2 provides PostScript-to-AFP conversion and AFP-to-PCL conversion, the HP:PCL-4 printer has the ability to print PostScript files with no need for any additional PostScript option card or memory. All PostScript print jobs are delivered to the printer as HP:PCL-4 data. Similarly, the usefulness of application programs can be extended by means of a transform exit sequence. The AFP Workbench, for example, supports only AFP and ASCII data, but when invoked in the proper context from a transform exit sequence, the AFP Workbench is an excellent "PostScript viewer."

Another example of the usefulness of the transform exit sequence is seen when considering the task of redirecting print jobs to a remote print queue. Although many data types might be enqueued on an OS/2 (or other) print queue, only a narrow range of data types can be exported to a particular remote printer and printed successfully. For example, an IBM AS/400 system printer typically supports only SCS (SNA character set) and AFP data, which creates problems when simple redirection implementations send PostScript and other inappropriate print jobs to the printer. A transform exit sequence, however, allows multiple transform exits to be linked together to provide more efficient redirection by converting any inappropriate data types (or at least identifying them if there is no appropriate conversion available) *before* they are sent.

Finally, consider that a transform exit sequence allows a LAN administrator to harness functions from many different vendors and sources. Because the transform exit sequence is made up of individual transform exits, each of which can make use of any application program that can be invoked from a command line, a transform exit sequence allows otherwise disparate functions from different products and vendors to be "wired together" to form an appropriate solution.

Applications of the technology

The applications described here share two traits: The first is that these are real applications of the technology, not theoretical, and at least one PSF/2 user is currently using the technology of the transform exit sequence as shown to solve real-world problems in a production printing environment. The second is that these applications were long desired, but the users had been unable to implement them in a post-spool environment before the technology of the transform exit sequence became available. Other technical solutions to some of these problems exist, but the fact that so many new printing applications could be created by the IBM Corporation using this technology attests to its flexibility and uniqueness.

- *Data-type router*

Figure 3 shows a single queue that distributes data to different printer queues according to type. PostScript is routed to one queue, HP:PCL to a second, and all other data types to a third. This is a powerful arrangement, which allows a print administrator to define a single print queue and make that single queue available to the LAN community, while using whatever printers are appropriate for the different data types. This prevents the accidental routing of inappropriate data to a printer by an individual user. It is not possible, for example, for PostScript to be routed accidentally to an HP:PCL printer. Instead, all jobs go to the single queue and are routed to the printer that can best handle that type of data. This configuration also lets the administrator change the printers being used, with no effect on where members of the user community send their print jobs.

This solution is done with a transform exit sequence that uses three terminal transform exits: one routes PostScript to its printer; the second routes HP:PCL; and the third routes whatever is left. The first two transform exits are conditionally terminal. For example, the PostScript transform exit could be interpreted to mean "If the print job is PostScript, forward it to the PS queue and remove it from this one." Files of a data type other than PostScript are not terminated but instead are handled by the next transform exit in the transform exit sequence.

The final transform exit is unconditionally terminal and sends whatever remains to the third printer.

- *Convert, upload, and print*

The application of the technology shown in **Figure 4** allows printers on an IBM MVS system or an IBM AS/400 system to be accessible by a LAN community. (The figure is a simplification of the full solution.) Because the print queues for the mainframe printers are managed from the LAN print server, they are indistinguishable from local LAN printers to anyone but the print administrator. This allows the LAN administrator to make use of whatever printers are appropriate for the LAN community, including those attached to large mainframe or midrange computers. As described previously, appropriate transform exits prevent incorrect data from being uploaded to the mainframe printers, transforming data when possible into a format that will print successfully on the target printer. Uploaded data are reblocked, as required, into appropriate mainframe format (into "5A" records on the IBM MVS system, for example) and submitted to the appropriate mainframe print queue. The uploading of the data is performed by the preferred API for the particular system (often based on Client-Access redirection for AS/400, and either CM/2 or TCP/IP LPR to upload to an IBM MVS system) built into a terminal transform exit.

- *Off-line PostScript*

The next solution allows overall printer throughput to be maximized for high-volume PostScript applications. As shown in **Figure 5**, two logical queues are defined, one driving the high-speed printer, and the other handling the raster image processing (RIP) of all PostScript jobs in the background. This configuration prevents a fast printer from being slowed down by the PostScript RIP; the printer can be printing other data while the PostScript RIP is performed. (A variation of this application aids scheduling by delaying large print jobs submitted during peak load times until a more suitable time, such as second shift.) When the PostScript RIP is complete, the converted job is sent to the real print queue, where it can be printed very quickly on a high-speed laser printer—much faster in most cases than the original PostScript could have been printed. This solution requires only a single transform exit on the queue for the physical printer, to route any PostScript to the background print queue. The background queue has a transform exit sequence that does the appropriate PostScript RIP and uses a terminal transform exit to send the resultant converted file to the real print queue, where it is printed.

- *Many logical queues for a single physical printer*

A certain LAN community identified 14 different media that were needed at various times, yet the only available

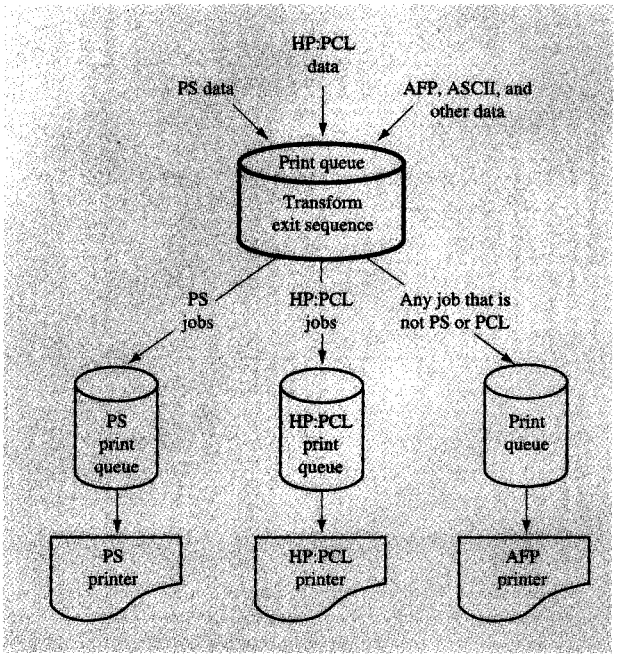

Figure 3

A single print queue distributes jobs to printers according to job data type. A transform exit sequence in the queue driver allows data to be routed to appropriate printers.

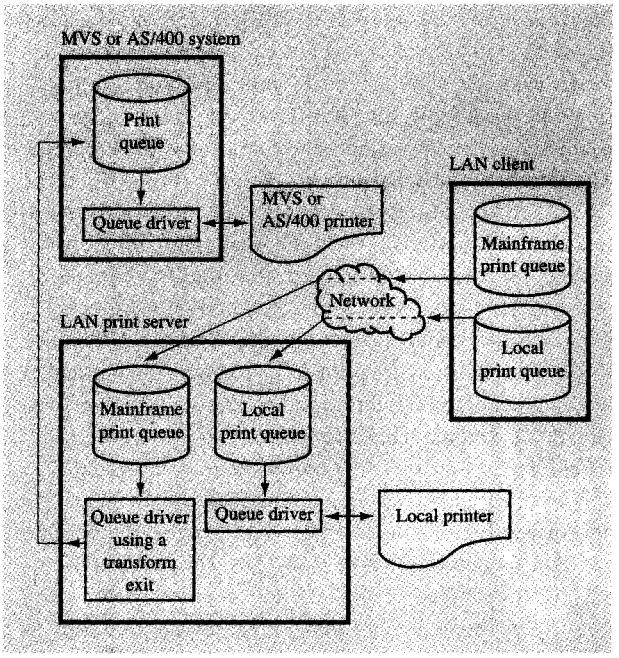

Figure 4

Application of transform exit sequences in which mainframe printers appear in LAN print queues to a client in the LAN.

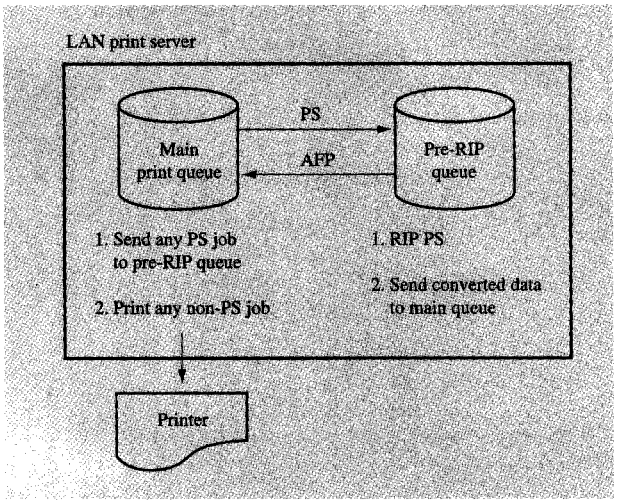

Figure 5

A background PostScript RIP can improve printer performance. By not sending PostScript jobs to the pre-RIP queue, the main print queue does not delay the physical printer with the PostScript RIP. The pre-RIP queue completes the PostScript RIP in the background and sends the converted data back to the main print queue when the job is ready to print.

laser printer had only a single input bin. A solution other than purchasing many more printers was required. **Figure 6** shows the solution to this problem based on the transform exit sequence technology, which simulates 14 media available simultaneously from a printer with only a single physical input bin. A different logical print queue is defined for each medium and feature that is required. The mnemonic names (such as "A4" for A4-sized paper, "green" for green paper), not a part of the transform exit sequence solution, help the LAN community direct print jobs to the correct print queues with minimal training. Arriving print jobs that require the medium currently loaded in the printer (A4 in the figure) are sent directly to the printer. Arriving print jobs that require another medium generate an alert signal to the operator that a change of medium will be required, and are moved to a held queue that is visible only to the operator and will be released when the medium has been changed appropriately. Looking over the held queues, the operator can quickly determine how many jobs are pending for each of the media not loaded and can schedule the physical medium changes (and corresponding print queue holds and releases) accordingly. Though manually intensive if frequent requests are made for unavailable media, this solution allows a print administrator to extend the utility of a single printer by building logical queues for all media that have to be made available to the user community.

Technically, this application is a straightforward use of the transform exit sequence technology, with individual transform exits routing jobs, alerting the operator, preconverting PostScript and other data types, and changing print-job attributes, as appropriate. Each of the user-accessible print queues has a transform exit sequence that sends the print job to the corresponding operator-controlled print queue after any appropriate data conversions have been made. If this corresponding queue is held, an alert signal is sent to the administrator as a result of a transform exit calling an application program that generates the alert signal. Also, certain features such as "three copies" of the medium currently loaded can be implemented without alerts or operator intervention. For example, within PSF/2, the transform exit body "APRINT %i dest = oper_queue copies = 3" will submit the print job to the queue called "oper_queue" and cause three copies to be printed [9].

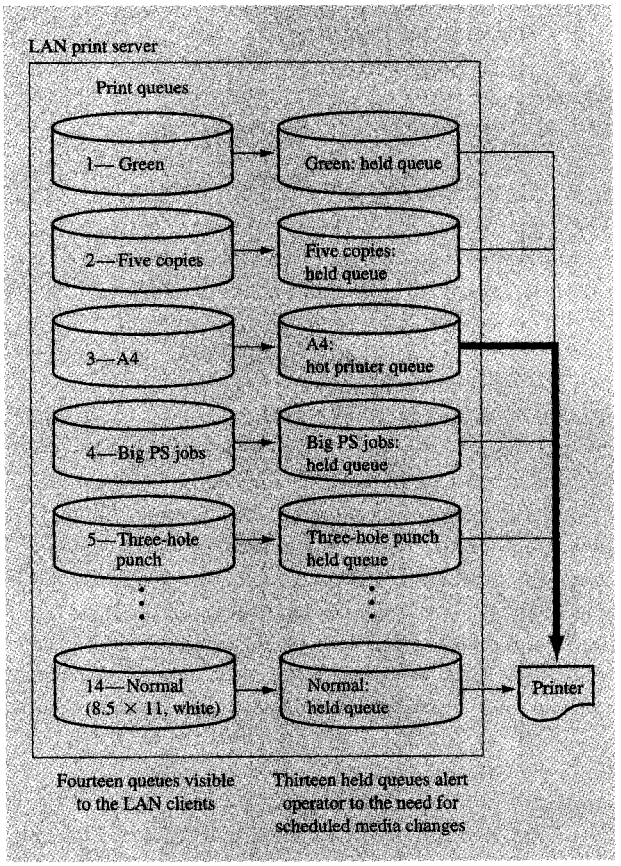

Figure 6

Support for virtual media in a single-bin printer using transform exits.

The effect of this from the end user's perspective is to magnify the actual capabilities of the printer: Although only one medium is ever actually loaded at a time, the printer appears to support all required media types at all times. From the printer operator's perspective, any user request that requires a medium change results in an alert signal, and the use of a number of held queues allows greatly improved scheduling and management of the printer. Note too that this solution is easily extended to include additional physical printers, should total workload require additional printers to be added to the configuration. Also, the solution could be extended to explicit media selection (as opposed to implicit by print queue name) in the future.

Conclusions

The transform exit and the transform exit sequence have enabled new classes of solutions without requiring any changes to the operating system or spooling system. This new technology allows APIs to be imbedded within transform exits, allowing the APIs to be called automatically when certain types of jobs arrive on a print queue. Multiple transform exits can be linked sequentially as building blocks to create a transform exit sequence, which is executed when a job arrives on a "print" queue; however, the flexibility of the transform exit sequence enables a broad range of solutions having nothing to do with printing *per se*.

This technology does not eradicate all existing print problems, but it has been proven to provide solutions to many problems in LAN printing environments, as implemented within PSF/2. This is not the only approach to solving these problems, and intelligent spooling systems based on the ISO DPA 10175 standard hold promise for making many of these solutions more widely available in the future. On the other hand, because the transform exit sequence technology does not depend on any spooling system or operating system, it can be used by queue drivers today in a wide variety of existing environments. It is also a framework on which additional function can be provided, suggesting that future improvements based on this work can be expected.

Of particular interest for future work would be an improvement in data-type-detection algorithms and heuristics. These might even be replaceable themselves, instead of being bound to the transform exit framework. Currently, a transform exit definition can express actions to be taken on the basis of data types but cannot influence which heuristics or algorithms should be used for making the data-type determination. Detecting and handling errors—syntactic or execution—can also be difficult within a transform exit sequence framework, since it can be difficult to detect whether the API call failed (i.e., the body of the transform exit was syntactically incorrect) or the application program itself had a problem during processing. Applicability of this work to operating systems other than OS/2 and AIX also deserves to be examined, since it should be generally applicable beyond its current scope of implementation.

Acknowledgments

Thanks to Art Roberts, for always finding a way to do the *right thing*, and to the many others who have contributed so much to the PSF/2 project over the years. Thanks also to my family for the support that made these advances possible.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Microsoft Corporation, Novell, Inc., Banyan Systems Inc., X/Open Co. Ltd., Hewlett-Packard Co., or Adobe Systems, Inc.

References

1. R. J. Howarth and B. G. Platte, "The Continuing Evolution of AFP," *IBM Syst. J.* **32,** No. 4, 665 (1993).
2. B. Curran and D. Kerr, *OS/2 Warp Administrator's Survival Guide*, 1st Ed., ISBN 0-672-30744-8, Sam's Publishing, 201 W. 103rd St., Indianapolis, IN 46290, 1995.
3. *Ibid.*, pp. 511–515, 525.
4. "Information Technology Text and Office Systems Document Printing Applications Part 2: Protocol Specification," *ISO/IEC DIS 10175-2*, International Standards Organization, 1995; available from the American National Standards Institute, 11 W. 42nd St., New York, NY 10036.
5. R. Hohensee, *Mo:DCA-P Reference*, IBM Publication SC31-6802-03, IBM Information Development, Dept. 588, IBM Boulder, P.O. Box 1900, Boulder, CO 80301, 1996.
6. Adobe Systems, *PostScript® Language Reference Manual*, 15th printing, ISBN 0-201-10174-2, Addison-Wesley Publishing Co., Inc., Reading, MA, 1990, p. 156.
7. *PCL 5 Printer Language Technical Reference Manual*, Order No. 5961-0509, Hewlett-Packard Co., P.O. Box 1145, Roseville, CA 95678, 1992, p. 42.
8. Adobe Systems, *PostScript® Language Tutorial and Cookbook*, 1st printing, ISBN 0-201-10189-0, Addison-Wesley Publishing Co., Inc., Reading, MA, 1986.
9. *A Guide to Using PSF/2*, 1st Ed., IBM Publication G544-5225-00, IBM Information Development, Dept. 588, IBM Boulder, P.O. Box 1900, Boulder, CO 80301, 1995, p. 125.

Received June 4, 1996; accepted for publication August 5, 1997

Scott D. Mastie *IBM Printing Systems Company, 6300 Diagonal Highway, Boulder, Colorado 80301 (mastie@us.ibm.com).* Mr. Mastie holds a Master's degree in computer science from the National Technological University and a Bachelor of Science degree in computer science from the University of Michigan Honors College. He has been working in IBM AFP (Advanced Function Presentation) software development for twelve years. Mr. Mastie has worked in diverse printing areas, including mainframe, workstation, cross-systems distributed printing, PostScript, and POD (print-on-demand). His current technical interests include enhancing digital printing technology to address the diverse requirements of high-volume offset-press applications. His author credits include a chapter on PSF/2 in the *OS/2 WARP Administrator's Survival Guide*, contributions to the *IBM Technical Disclosure Bulletin*, and XPLOR global conference proceedings. Mr. Mastie is a member of XPLOR and ASQC.