PPA Printer Software Driver Design

The software driver for the HP DeskJet 820C printer performs many functions that were formerly performed in the printer, including swath cutting, data formatting, and communications. The driver also includes a PCL emulation module for DOS application support.

by David M. Hall, Lee W. Jackson, Katrina Heiles, Karen E. Van der Veer, and Thomas J. Halpenny

The software driver for the new HP DeskJet 820C printer includes many new functions that need to be performed on the host computer because of the printer's Printing Performance Architecture (PPA). In older PCL (Printer Control Language) printers, these functions were performed in the printer. Fig. 1 shows the differences. These functions include:

- Swath cutting
- Data formatting
- PPA communications
- PCL emulation for DOS application support.

This article provides an overview of the changes necessary for supporting PPA and then discusses each of the functions listed above in more detail.

Driver Overview

Under the Windows® operating system, printer drivers are responsible for supporting a specific API (application programming interface) known as the DDI (Device Driver Interface). This interface gives the driver fairly high-level drawing commands. It is up to the driver to take those commands and produce a bitmap that can be encapsulated in a language and sent to the printer.

Typically, within a Windows printer driver, a rendering engine takes the DDI commands and produces a rendered bitmap. A halftoning algorithm is performed on the rendered bitmap and a halftoned bitmap is produced. This halftoned bitmap is typically in a format that can be encapsulated in a language such as PCL and then given to the printer.

For the HP DeskJet 820C, this halftoned bitmap has to be put through additional processing as shown in Fig. 1 to create data that is ready to be printed by the printer's electronics directly. This additional processing includes swath cutting and sweep formatting.

Since the HP DeskJet 820C does not understand PCL (Printer Control Language), a PCL emulation module is necessary to provide support for DOS applications. The DOS application data stream is captured by a DOS redirector and passed to the PCL emulator, which produces a halftoned bitmap ready for swath cutting.

PCL versus PPA

Fig. 2 shows the printing model for PCL printers. For PCL printers, the process of encapsulating the halftoned bitmap is fairly straightforward. Raster data from the halftoned bitmap is compressed, PCL wrapped, and then sent to the I/O module. The reason that this is a simple process is that PCL printers are designed to receive data in the same format as the halftoned bitmap. PCL printers unwrap the data into an internal buffer and perform the necessary swath cutting and data formatting internally.

Fig. 3 shows the printing model for PPA printers. For the HP DeskJet 820C, the PCL encapsulator is replaced with an SCP data encapsulator. SCP (Sleek Command Protocol) is an HP-proprietary command language. This module contains swath cutting functionality, data formatting, SCP language encapsulation, and printer status management.

Raster data from the halftoned bitmap comes into the SCP data encapsulator, goes through the SCP manager, and eventually arrives at a raster block within the swath manager. The swath cutting state machine examines the data and determines the appropriate sweep to generate. A sweep is a collection of rasters appropriate for the printer mechanism to print while it sweeps the printhead over the paper.

Once the sweep is generated, it is given to the sweep formatter. The sweep formatter is responsible for taking the sweep data and putting it into the appropriate format for the HP DeskJet 820C internal hardware. Then the data is compressed, wrapped in SCP, and handed off to the I/O layer.

The I/O layer is responsible for communicating with the printer by wrapping the data stream in VLink and IEEE 1284 protocols. VLink is an HP-proprietary link-level protocol and IEEE 1284 is an industry-standard physical-layer protocol.

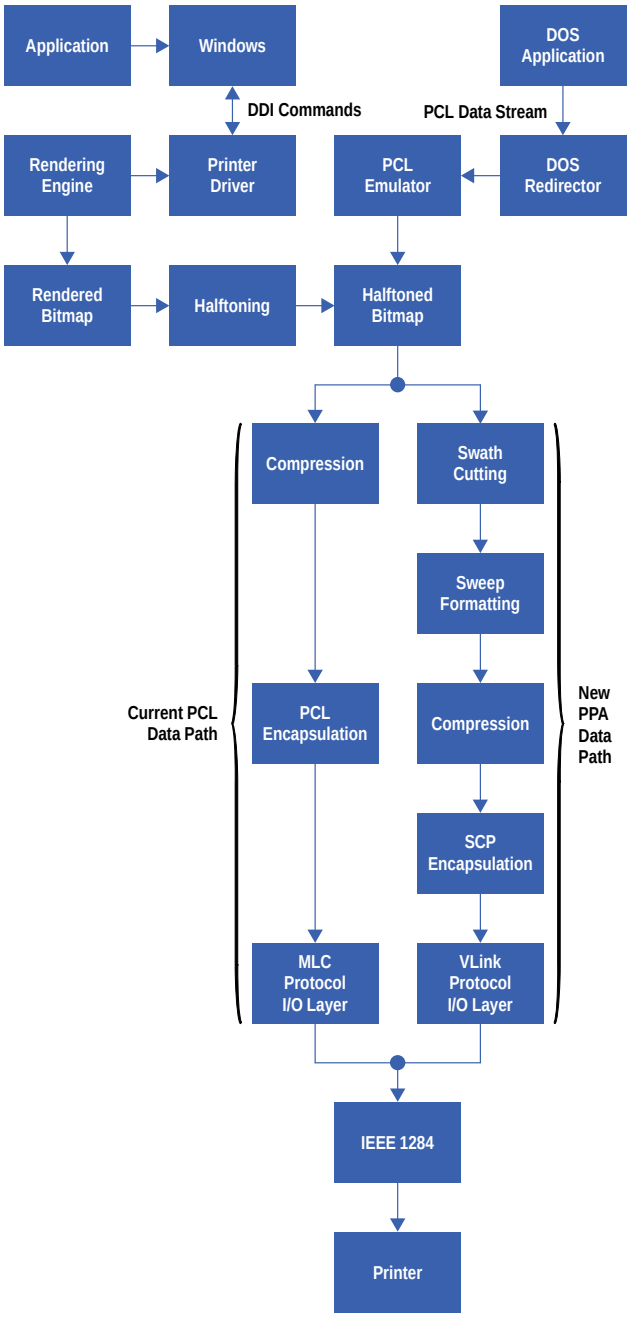

Fig. 1. *Printer driver functional block diagram, showing differences between PCL and PPA data paths.*

Performing Swath Cutting on the Host

Swath cutting is the process of taking a page of halftoned raster data and producing sweep data appropriate for the carriage electronics to print as the printhead is sweeping across the page. Swath cutting has historically been part of printer firmware, but in the HP DeskJet 820C printer, it is part of the software driver running on the host computer. Typically, a swath manager encapsulates a swath cutting engine and receives as input a bitmap representation of the page to be printed. The swath manager is responsible for determining how the pens and paper should be moved and when and how the pens should be fired to produce the printed page. The swath manager must balance the often conflicting goals of printing with the highest possible print quality and printing as fast as possible. The swath manager must be aware of certain printer-specific attributes such as printhead alignment and strategies to minimize line feed error. In PPA, swath management is performed on the host computer.

The process of swath cutting can be readily modeled using a state machine. Consider the example shown in Fig. 4. A state machine capable of processing this page would need to contain five states: Top of Page, Blank Skipping, Black Text Printing, Color Graphic Printing, and End of Page. Thus, we can create the state machine shown in Fig. 5. A particular instance of a state machine exists for each print mode the swath manager supports. For example, there could be a print mode for pages that only have

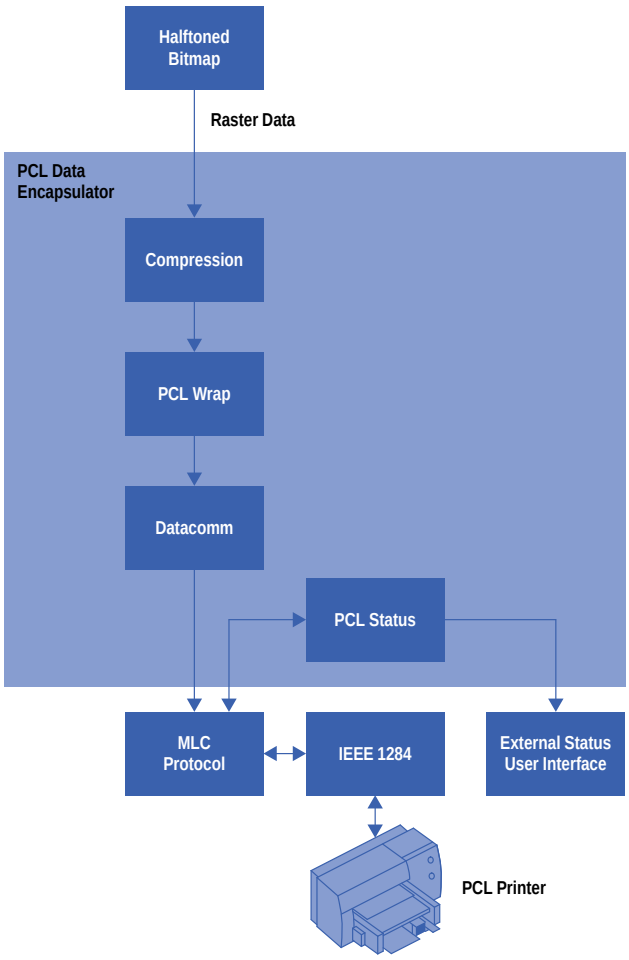

Fig. 2. *PCL printing model.*

black text on them, another print mode for pages with black and color, and yet another print mode for pages with complex graphic images.

As the state machine begins to examine the data on the page, it starts in the Top of Page state. The first data it comes to is a series of blanks. This would cause it to move to the Blank Skipping state. During this transition the swath manager would typically load the page. While in the Blank Skipping state, the swath manager would advance the paper. Next, it would encounter a black text region and move to the Black Text Printing state. Depending upon the type of printing being done at that time, this transition may produce a sweep.

Assume that for this print mode, the data on the page is being printed by making two sweeps for each line. Thus, in making the transition from Blank Skipping to Black Text Printing the printer could print the first pass of the black text region with the bottom half of the printhead, advance the paper half a printhead height, and then enter the Black Text Printing state. During the next sweep generated, the Black Text Printing state would finish the lines that were printed during the transition and continue printing the black text region (see Fig. 6). The data on the page would continue to be consumed and transitions made between states until the End of Page state is reached.

Obviously, this example is a simple one. The number of states and the number of transitions to consume data for a real page can be quite large. Using PPA we have the opportunity to perform the resource-intensive task of swath cutting on the host. This allows greater flexibility in developing machines with unique print modes, which provides the opportunity for higher print quality and throughput as well as reduced mechanism costs.

PPA Data Formatting

The HP DeskJet 820's Printer Performance Architecture requires the host to perform the majority of the data manipulation. The data that is sent to the printer must be in a format that is very close to the final form used to fire the printheads. The main difficulty in formatting the data for the printhead lies in the fact that the data doesn't come out of one position on the carriage mechanism. Instead, there are two columns for each of the four pen colors. Each column is at a different vertical and horizontal offset from a relative zero carriage position. To minimize the cost and complexity of the electronics in the printer mechanism, the data sent from the host to the printer must be ordered so that it is ready to go directly into these offset printheads in the appropriate order so that it is fired at the correct locations on the page. This ordering is based on:

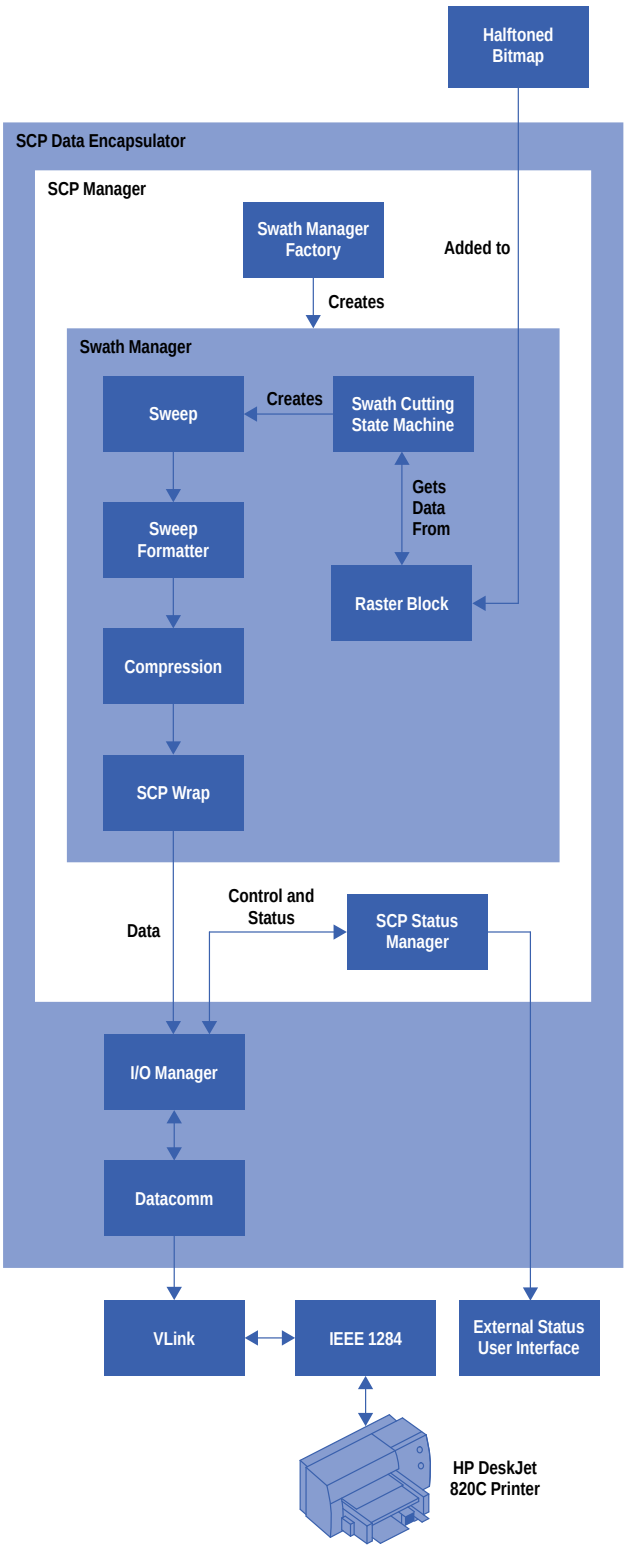

Fig. 3. *PPA printing model.*

- The starting page position of each color
- The *servant* architecture in the printer hardware (described later)
- The printhead (see Fig. 7).

To print a page, it is necessary for the carriage mechanism to move back and forth across the page, firing drops of ink as it moves. Each movement of the carriage across the page is called a print sweep. When the driver receives a page to print from some application, it renders the page into a halftoned bitmap. At this point, a PCL printer driver would send compressed and encapsulated PCL data directly to the printer. The PPA printer driver uses the swath cutting state machine to generate a

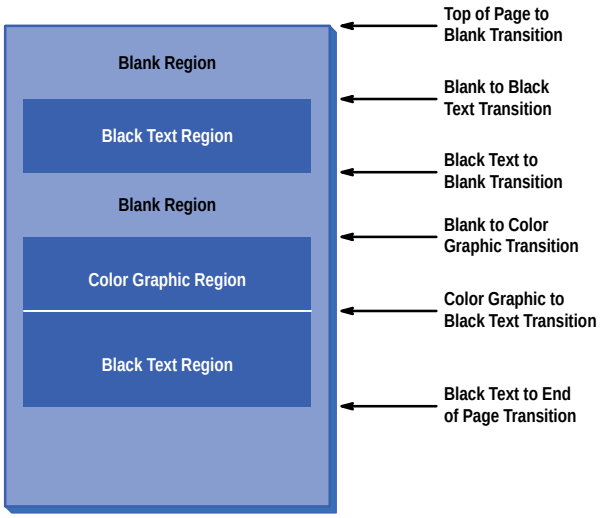

Fig. 4. *Swath cutting state machine transitions for a typical page.*

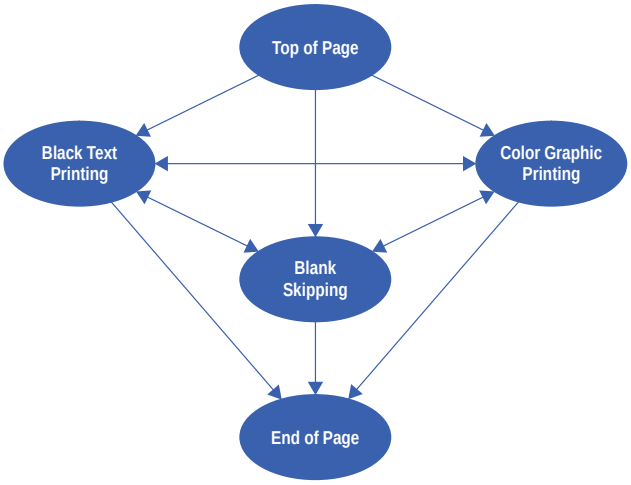

Fig. 5. *Swath cutting state machine.*

Fig. 6. *(a) In making the transition from Blank Skipping to Black Text Printing, the printer prints the first pass of the black text region with the bottom half of the printhead, advances the paper half a printhead height, and then enters the Black Text Printing state. (b) During the next sweep generated, the Black Text Printing state finishes the lines that were printed during the transition and continues printing the black text region.*

swath of data that can be printed by a single pass of the pen carriage. The resulting swath of data is passed on to the sweep formatter, which manipulates the data into a buffer that can be copied directly to the printheads. The print sweep formatter uses knowledge of the pen carriage, hardware, and firmware architecture to prepare and reformat the data into a print sweep.

The number of print sweeps required on a given page is dependent upon:

- The amount of data on the page (text or dense graphics)

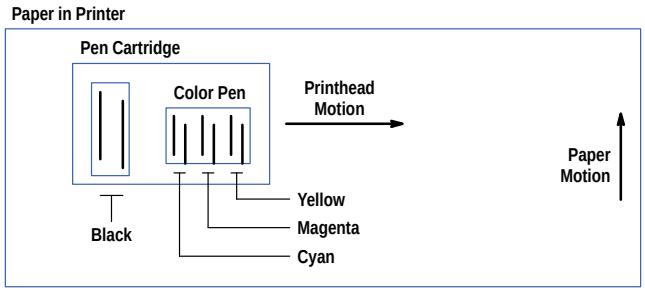

Fig. 7. *HP DeskJet 820C print cartridge layout. The lines correspond to nozzle columns and their general configuration on the printer carriage.*

- The print mode selected by the user (best, normal, or econofast)
- The paper type (plain, glossy, transparency, or special).

For each print sweep, the host sends two pieces of information to the printer. The first is the PRINT_SWEEP data, a buffer of image data sent before the PRINT_SWEEP command, which contains an entire sweep of swing buffer data blocks in the correct order. The second piece of information is the PRINT_SWEEP command, the mechanism by which the driver tells the printer where and how to place the print sweep data on the page. A PRINT_SWEEP command contains minimum and maximum positions for each pen column, the print direction, print speeds, and NEXT_PRINT_SWEEP information.

The PRINT_SWEEP command information is calculated by the printer driver based upon:

- Which pens are active (black, cyan, magenta, yellow)
- The starting and ending locations on the page for each pen color
- The direction of the print sweep
- The servant architecture:
  - The distances between pens
  - The distances between odd and even columns within a pen
  - The 0,0 position in relation to the pen columns.

Servant Architecture

The servant hardware (see ***Article 4***) is composed of a pair of buffers, called *swing buffers*, for each column of the printhead (two columns per color). To build a print sweep, the driver must:

- Separate the image into CMY planes, or primitive data blocks
- Separate the primitive data blocks into swing buffer data blocks
- Order the swing buffer data blocks into a servant image.

A primitive data block (a bitmap image of each plane for each color) is created by the driver. Each primitive data block needs to be split into two separate swing buffer data blocks: an odd block and an even block. This is necessary because of the pen design, which consists of two offset columns, as pictured in Fig. 8.

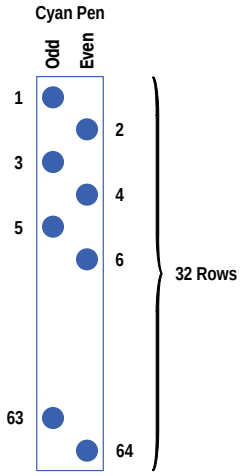

Fig. 8. *Each color pen has two offset columns of nozzles.*

Each column on the color pen has 32 nozzles. The color pen has a height of 64/300 inch. For any given column of data, rows 1, 3, 5, ..., 63 will be part of the odd column and rows 2, 4, 6, ..., 64 will be part of the even column.

The even and odd swing buffer data blocks are each 8 bits wide, the width of servant RAM, and each is the height of a printhead nozzle column. Swing buffer data blocks are cut for each primitive color and for either the even or odd nozzle column. Thus, each swing buffer data block contains every other row from the primitive data block.

Fig. 9 shows a simplified example of a primitive data block. Each byte is a buffer of data that is one byte (8 pixels) wide by N rows high, where N is the number of nozzles in a printhead column. For the example in Fig. 9, N is 6, while N is 32 for the HP DeskJet 820C color printheads.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
60	61	62	63	64	65
66	67	68	69	70	71
72	73	74	75	76	77
78	79	80	81	82	83
84	85	86	87	88	89
•	•	•	•	•	•
•	•	•	•	•	•
•	•	•	•	•	•
120	121	122	123	124	125

Fig. 9. *Primitive data block organization for a printhead that has two columns of six nozzles per color. Byte n (n = 0, 1, 2, 3, 4, 5) is a buffer of data 8 pixels wide by 6 rows (nozzles) high. The HP DeskJet 820C printheads have two 32-nozzle columns per color, as shown in Fig. 8.*

Each column of the primitive data block in Fig. 9 is divided into four swing buffer data blocks with bytes relocated to the positions shown in Fig. 10. Only the cyan pen is shown, and only two of the swing buffer data blocks for each column of Fig. 9 are shown. The drawing would be similar for the magenta and yellow pens.

Swing Buffer Data Blocks, Byte 0		Swing Buffer Data Blocks, Byte 1	
CO:0	CE:0	CO:1	CE:1
0	6	1	7
12	18	13	19
24	30	25	31
36	42	37	43
48	64	49	55
60	66	61	67

CO:x = Cyan Odd Printhead Column: Primitive Data Block #
CE:x = Cyan Even Printhead Column: Primitive Data Block #

Fig. 10. *Swing buffer data blocks for the example primitive data block shown in Fig. 9.*

Once the data is in the form of even and odd swing buffer data blocks, the blocks must be ordered and sent to the printer. This ordering is done with knowledge of the column spacing on the printhead and knowledge of the order in which the servant architecture will require the data. The printer driver controls the order in which the columns will trigger via fields in the PRINT_SWEEP command. The ordered swing buffer data blocks are then sent down as PRINT_SWEEP data ready to be loaded into the primitive swing buffers in the printhead.

Each primitive swing buffer consists of two 8-bit columns, separated by a *swing trigger point*. While the servant print process is unloading one side of the odd column swing buffer, the other side of the odd column swing buffer is being loaded by the servant load process. Once the byte is loaded, the servant print process fires one bit by 32 rows at a time for each pen column in the color pen. When the servant print process has unloaded all eight bits, it crosses a swing trigger point, and the

servant print process switches to the other swing buffer and triggers the servant load process to load the empty swing buffer. The pen fires one bit by 32 rows at a time for each pen column. The servant (printer) is responsible for any complexity involved below the byte level.

When all of the swing buffer data blocks have been consumed by the printhead, the carriage mechanism uses the NEXT_PRINT_SWEEP information to position itself for the start of the next print sweep.

Because the PPA printer relies upon the driver to format the data appropriately, the architecture does not require the printer firmware to have any knowledge of the operations just described. Thus, the cost and complexity of the electronics in the printer mechanism are significantly reduced.

PPA Communication

One of the goals of the HP DeskJet 820C printer is to provide continuous feedback to the user during any printing operation, and to guide the user during problem solving. To accomplish this, the driver requires a mechanism to ask the printer for information and to allow the printer to notify the driver whenever something happens (the printer is out of paper, the user opened the cover, etc.). The mechanism used by the PPA driver to communicate with the printer is called *status messaging*.

To notify the user to align the print cartridges when a print cartridge has been changed, that the top cover is open, or that something else needs attention, a bidirectional link with the printer is required. Two new HP-proprietary protocols allow the driver to communicate bidirectionally with the HP DeskJet 820C: VLink packet protocol and Sleek Command Protocol (SCP). Previous HP DeskJet printers used an I/O packetizing protocol called MLC (Multiple Logical Channel) and a proprietary HP printer command protocol. For PPA, VLink replaces MLC, and SCP replaces both PCL and the old printer command protocol.

While giving users error messages might seem to be a luxury they could do without, the real reason to have a protocol like VLink is that it is useful to figure out what is wrong when, for example, the printer's input buffer fills up, the printer stops accepting data, and the host is unable to send even one more byte. This often happens and is temporary, but in the days before bidirectional protocols, the driver would sometimes wait and wait to be allowed to send again, and it didn't know whether the delay was because the top cover had been opened, a print cartridge had failed, or a fatal error had occurred. It is helpful to know whether to abort the job or ask the user to insert a print cartridge or close the door. With a bidirectional protocol, the printer tells the driver exactly what the problem is, and the driver can decide what action to take next.

A bidirectional link is not required for printing or to have limited status feedback from the printer. However, unlike PCL printers, which can accept either PCL data wrapped in MLC or raw PCL data, PPA printers can only interpret data wrapped in VLink and SCP. Thus, while MLC is an option that can be added when a bidirectional link exists, VLink must handle printing with and without a bidirectional link as well as printing to a file.

Based on VLink's channelization features, there are two paths the data can take to the printer. One is for image data (the dots that will go on the page), and the other is for command data. Command data includes commands sent to the printer, such as "Print this sweep," requests for information, or queries, such as "What print cartridges are installed?", and status information, termed *autostatus*, such as "The top cover is open." Sending image data is easy from an I/O standpoint—if the printer has room in its buffer, the driver will send the data. Since command data must be sent and also received (autostatus may come in at any time), it is by nature a more complex affair.

As shown in Fig. 11, data that comes in from the front end of the driver goes through the data encapsulator, like PCL printer drivers, but from there it goes through several new objects. The SCP manager wraps the data in SCP and sends it to the I/O manager, which provides an interface to the datacomm objects. The VLink layer wraps the data in the VLink protocol and sends it to the IEEE 1284 layer and out to the printer.

Data that is sent by the printer, such as notifications that something is wrong, are put in the printer's output buffer. The driver spawns a hidden executable at the beginning of each print job called the *port sniffer*, which checks the port every half second to determine if the printer has sent any data. If so, the data is routed through the IEEE 1284 layer to the VLink layer, which then posts a message to the I/O manager's hidden status window.

The status window uses a callback to call into the SCP manager, which translates the status information, and if the message is something that should be displayed to the user, puts it on the event list. The event list prioritizes the messages on it so that the most important message gets sent to the HP Toolbox, which displays the dialog box to the user. If the message is an error, it may get resolved (for example, the user puts paper in the printer and presses the **Resume** button). The message is then routed up through the same path and deleted from the event list. The Toolbox takes the dialog box down and displays the next most important message, if there is one.

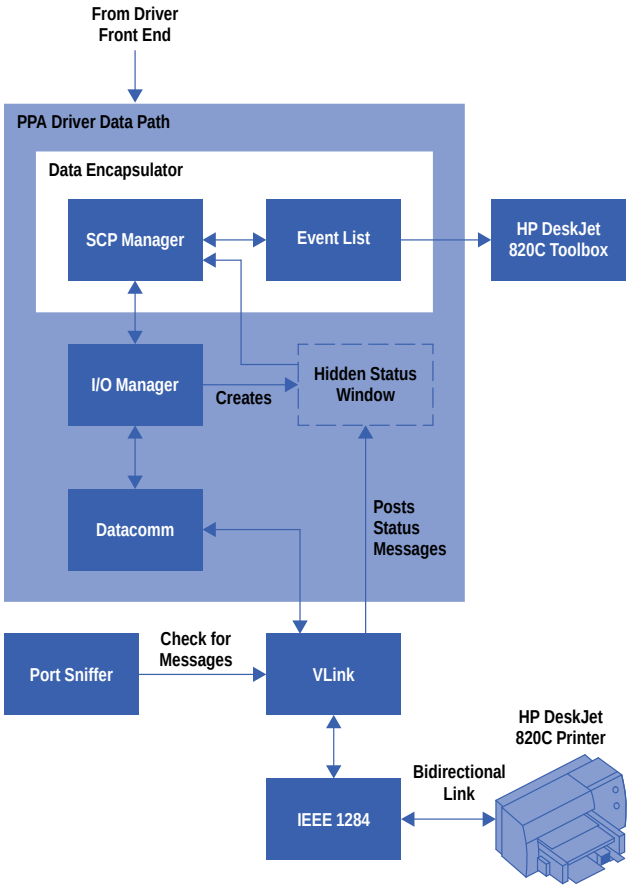

Fig. 11. *PPA status messaging architecture.*

Internal Objects in PPA Status Messaging

PPA status messaging involves several high-level modules and objects: the SCP (Sleek Command Protocol) manager, the I/O manager, the VLink module, and the event list (see Fig. 12).

SCP Translator. The function of the SCP translator object in the SCP manager is to encode data into the SCP format and decode messages received in the SCP format from the printer into query replies and event information. The SCP translator does not send SCP data directly to the I/O manager, since memory management for the data buffers is done in the SCP translator's clients, which are the swath manager and the status manager. The client of the SCP translator passes in a pointer to the data, an empty buffer, and the maximum data length. Once the data has been packaged, if the SCP translator finds that the data is larger than the buffer, it will return an error. Otherwise, it will pass back the actual SCP data length. The goal in designing the SCP translator was to encapsulate the Sleek Command Protocol so that changes in SCP in the firmware affect clients of this module as little as possible.

Commands in SCP use the format shown in Fig. 13. The command specifier field identifies the SCP command. The length field indicates the number of bytes in the data field. The data field does not exist for every command.

Priorities. Priorities allow the printer to execute commands in a different order than received. This may be necessary when a command cannot complete execution and it is desirable for the printer to process queries so the driver can find out what the problem is. Priority levels are defined in the SCP translator and the clients can set whatever priorities they like. Standard priority levels are defined as shown in Table I.

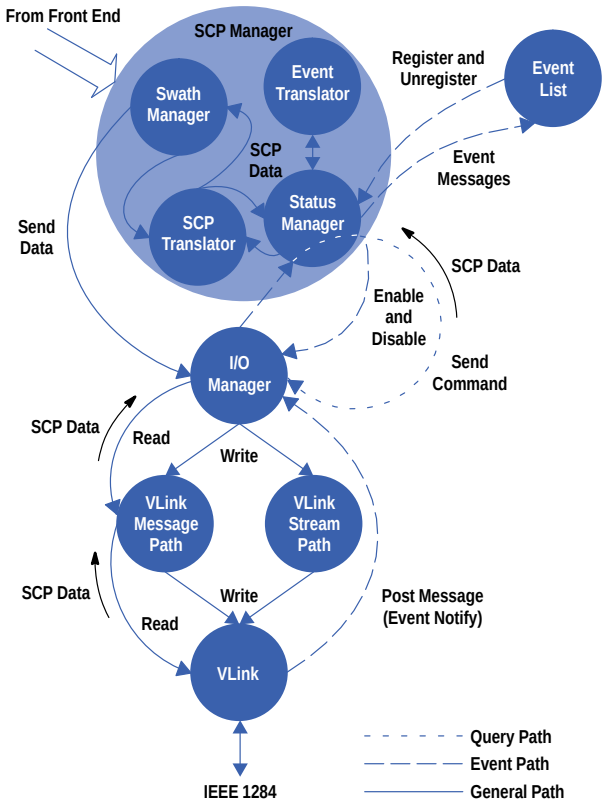

Fig. 12. *Calls between status messaging objects.*

Fig. 13. *SCP command format.*

Table I
Command Priorities

Command	Priority
Printing Commands	Low
Queries	Medium
Initializing and Deinitializing the I/O Link	High
Recovering from Errors	Recover
Canceling	Cancel
Restarting the Printer	Restart

It is assumed that the swath manager will send all of its printing commands (LOAD_MEDIA, PRINT_SWEEP, EJECT_MEDIA) at the lowest priority. Any queries it needs to make will call into the status manager. All queries should be at the same priority and higher than printing commands. It is up to the clients to set priorities.

Status Manager. The status manager manages messages to and from the printer. These messages can be broken into two categories: events and queries. Events are unsolicited notifications by the printer (i.e., autostatus) that something has occurred to change the state of the printer, such as “the door is open.” Queries are requests for information made by the driver to the printer, such as the pen IDs of the installed pens. The status manager tracks the state of the printer and creates

events when state changes occur. For example, when the **Resume** button is pressed, an internal state change occurs. This state change is recognized by the status manager and reported as an event to the event translator.

When the status manager receives notification of an event, it determines what has changed and whether the event is something the event translator has requested to know about. If it is, a callback in the event translator is called.

Upon starting a print job, the status manager queries the printer to get the current state of events. No event notification will be received until an event occurs in the printer.

Event Translator. This module exists between the event list, which is Windows-specific, and the status manager. The event translator translates the bit-field data, which is returned to the status manager by the printer in autostatus, into events. New events are added to the event list by the status manager, and events that are no longer valid (e.g., the door was open but the user shut it) are removed from the list. The event list orders the events reported to it according to their importance to the user, and tells the status monitor which dialog box to display. From most important (1) to least important (10), the following event priorities are used: (1) I/O errors, (2) paper jam, carriage stall, or maximum thermal limit, (3) pen failure, (4) wrong pen, (5) low or out of ink, (6) pen missing, (7) out of paper, (8) cover open, (9) dry timer, (10) new pen.

I/O Manager. This module is intended to glue the VLink module, which is Windows-specific, to the SCP manager, which is shared. Handling for events, queries, and buffer management must be performed by the I/O manager in addition to sending data to the printer as quickly as possible.

Events. The I/O manager creates a hidden window so that when the printer sends unsolicited event notification, Windows messages to that effect can be posted to this window by the VLink module. When the I/O manager processes this window message, it will read the SCP data buffered by VLink and call a callback in the status manager, passing in the SCP data.

Queries. To get replies to queries, the inquiring module calls VLink, specifying a buffer in which to place the reply. VLink checks this query reply buffer to see if anything has been returned in response to the query. If so, it immediately returns with the SCP data. If not, it polls the incoming channels for a specified timeout period to attempt to retrieve the reply. If a reply is received before the timeout period expires, the SCP data is passed through to the status manager.

Datacomm Paths. The image and command datacomm paths send data to the printer as long as there is space in the buffer. If space runs out, the command datacomm path waits until more space becomes available. The image data is handled differently. If space runs out while sending image data, the image datacomm path returns to the caller, allowing it to render more swaths until more space becomes free in the printer.

VLink. The VLink module must package data in a protocol the printer recognizes, and send only as much data as the printer can take, as quickly as possible. VLink must also unwrap data from the printer and route the messages to the appropriate clients.

The VLink protocol replaces MLC (Multiple Logical Channels) for the HP DeskJet 820C. Like MLC, VLink's intent is to provide a way for the host and the peripheral to exchange data. Unlike MLC, VLink is not optional. All data going to the printer must be wrapped in its protocol. In addition, VLink is streamlined or "sleek," and doesn't have many of MLC's features. MLC supported multiple logical channels, while VLink supports two outgoing and three incoming channels.

Outgoing Channels. The printer accepts data in either its input buffer or its command buffer. The VLink module specifies which type of data it is sending through a field in the VLink packet header. A template of a VLink packet is shown in Fig. 14.

Fig. 14. *VLink packet format.*

Image data is sent to the printer's input buffer on the image data output channel. Commands and queries are sent to the command buffer on the command data output channel.

Incoming Channels. Since a bidirectional link cannot be guaranteed, all incoming data is optional. This is necessary for file dumps and bad cables, and miscellaneous communication problems.

The printer periodically notifies the host how much buffer space is left in the printer. This is known as *credit*, and the printer sends notification for both the command and input buffers on the credit input channel. The VLink module will not send more data than the available credit.

VLink accepts two types of data packets from the printer in addition to credit packets: query replies, which are expected on the status input channel, and a collection of bundled items regarding printer status (such as out of paper), called autostatus messages. Autostatus messages ultimately map to events.

An autostatus message from the printer consists of a bit collection of several long words representing the current state of the printer. For example, when the door is opened, the door open bit in the collection is set to true. A report is generated on the autostatus input channel when any of these bits are toggled.

When the VLink layer receives some data, the data is identified as either credit, a query reply, or an autostatus message. Credit is interpreted and handled within the VLink module. A query reply or an autostatus message is buffered internally so that the clients can read it later.

If a received message is an autostatus message, the VLink layer posts a Windows message to the I/O manager indicating that an autostatus message is waiting to be read. When the I/O manager processes the Windows message, it reads the buffered autostatus message. Posting a message is necessary so that VLink can be free to poll the data lines for more incoming data from the printer.

Once the buffered message has been read, it is deleted. Only one query reply and one autostatus message can be buffered at a time. If a new message comes in before the original message can be read, the new message replaces the old one. It is for this reason that no additional printer queries should be made while waiting for a reply. No harm is done if a new autostatus message overwrites the old message because the same information is contained in each message and the newest message is the most relevant.

PCL Emulation for DOS Application Support

The development period of the HP DeskJet 820C coincided with most users rapidly transitioning away from DOS applications towards Windows applications. While we expected that most users would use the printer in its optimized design center, we recognized that we needed an adequate bridge to the few DOS applications that would continue to be used.

The HP DeskJet 550C printer was the final printer to be supported by most DOS applications, so the solution had to be functionally compatible with this printer and provide equally good print quality. We chose to provide compatibility with the HP DeskJet 660C printer, which was a contemporary printer that satisfied these requirements and provided an internal interface that enabled us to separate the PCL personality from the printer engine firmware. We planned to port the PCL personality functions to the HP DeskJet 820C printer driver, encapsulating them in a *PCL emulator* module. The required printer-engine functions would then be supplied by the rest of the HP DeskJet 820C driver. In this way, we could minimize design changes and maximize the chances of identical compatibility. If a DOS application is run from an MS-DOS prompt window, also referred to as a *DOS box*, the printer driver can intercept the PCL data stream that the DOS application sends to the PC's parallel port and redirect the data stream to the PCL emulator.

The HP DeskJet 820C PCL emulator encapsulates the HP DeskJet 660C formatter and text engine code. The design of the HP DeskJet 660C firmware was such that all interfacing to the external mechanism was done through a well-defined API internally known as the *Ed Interface* (see Fig. 15).

The Ed Interface resides between the formatter and font manager and the rest of the firmware. It is a collection of function calls to the support code in the firmware. Since we reused the formatter and font manager code, we provided the equivalent firmware functionality by mapping the Ed Interface calls into HP DeskJet 820C support objects.

The functions of the formatter and text engine firmware code were written in C, and as such are functions in the PCL emulator application (Fig. 16). The PCL emulator application provides C++ objects that encapsulate the functionality expected by the Ed Interface.

The PCL emulator application is designed to receive a file name that contains the PCL data to operate on. Interfacing between the internal PCL emulator object and the external driver is provided through a PCL personality object.

The PCL emulator is implemented as an executable application because the original firmware code expects to be a separate task, and this implementation allows almost direct porting of the HP DeskJet 660C firmware code. The PCL personality provides the handler functions and the external interface for receiving the PCL file name.

To allow DOS applications to print to the HP DeskJet 820C, it is necessary to capture the data generated by the DOS applications. This process is referred to as *DOS box redirection*. Essentially, it is necessary to capture the bytes intended for the parallel port and put them into a file so that the PCL emulator can properly interpret the data.

Under Windows 3.1, DOS box redirection is not part of the operating system, so it was necessary for us to provide a redirection solution. This functionality is provided by a redirector VxD (virtual device driver), a redirector DLL (dynamic link library), and a redirector EXE (executable), as shown in Fig. 17. These three pieces capture the data stream and put it into a temporary file. This file is then handed to the driver, and the driver hands it to the PCL emulator.

Under Windows 95 (Fig. 18), DOS box redirection is provided by the Windows printing system, so our redirector solution is not necessary for spooling to work under Windows 95. PCL printers essentially get DOS box redirection free. PPA printers need to intercept and perform PCL emulation on the DOS data stream. Microsoft provides a replaceable module called a language monitor where the data stream can be intercepted. The language monitor is a 32-bit DLL called directly by the spooling subsystem. The language monitor takes the incoming buffers, writes them to a temporary file, and passes the file name to the driver.

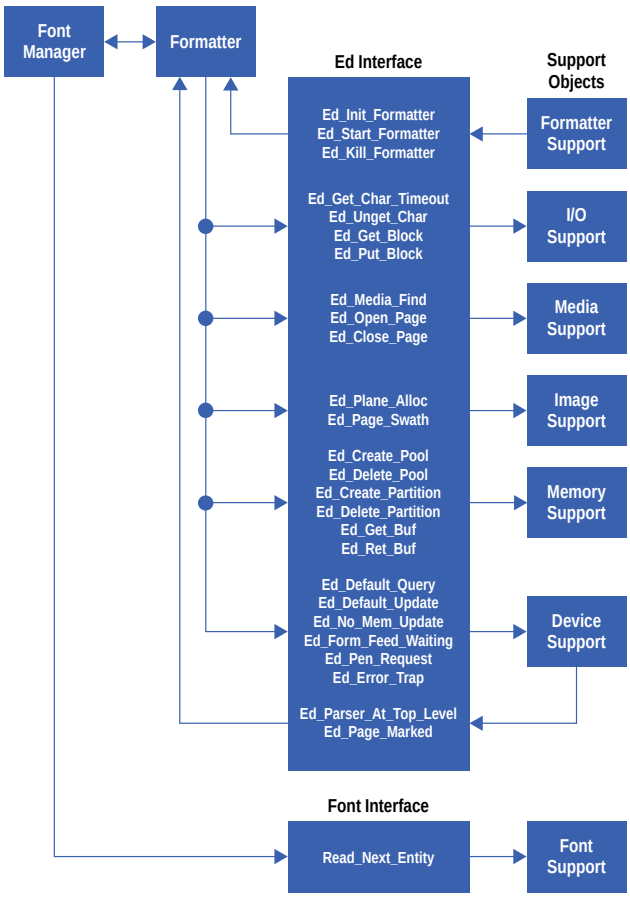

Fig. 15. *PCL emulation is provided in the HP DeskJet 820C printer by mapping the existing Ed Interface calls to DeskJet 820C support objects.*

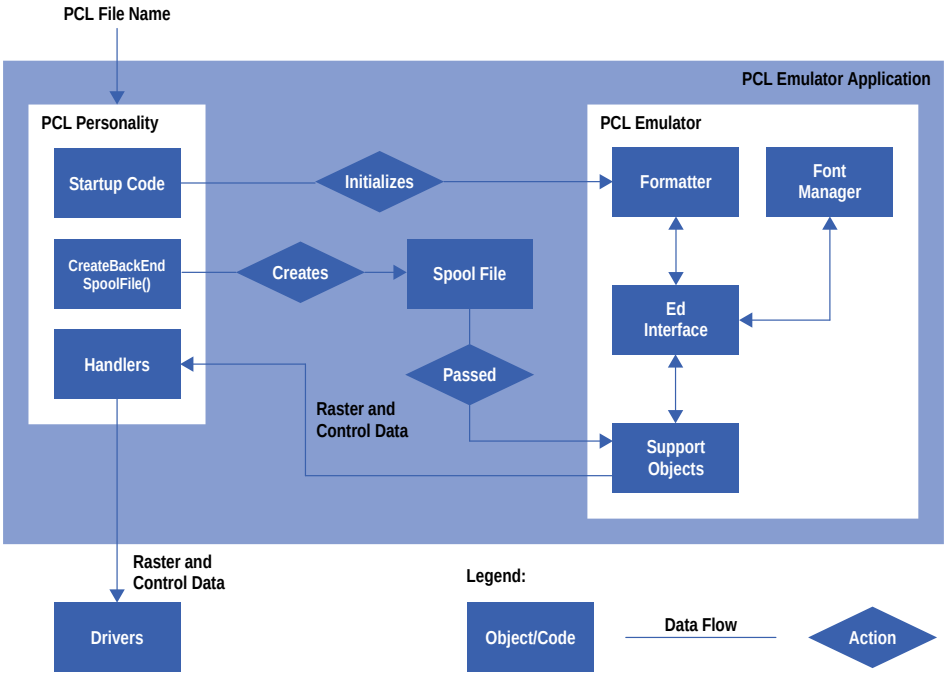

Fig. 16. *The PCL emulator application provides C++ objects that encapsulate the functionality expected by the Ed Interface.*

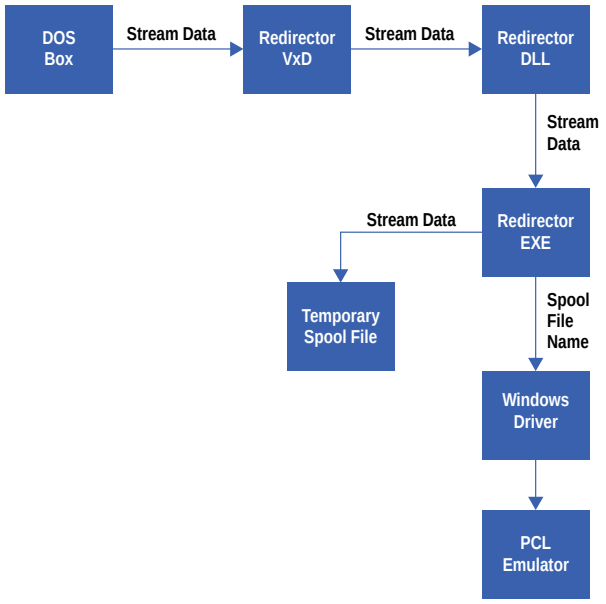

Fig. 17. *DOS box redirection for Windows 3.1.*

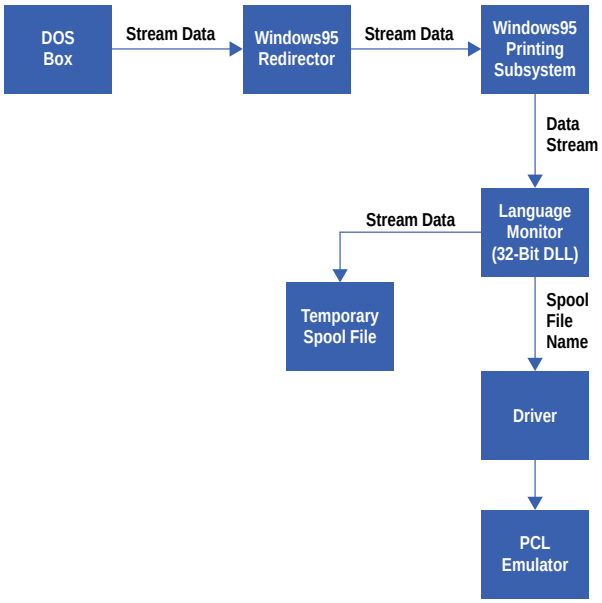

Fig. 18. *DOS box redirection for Windows 95.*

Porting the Firmware

The process of porting the C-language code from the HP DeskJet 660C presented several challenges. The original firmware was developed for a Motorola 68000 processor, while the printer driver runs on the Intel 80x86 processor in Windows 16-bit mode.

These two hardware platforms have conflicting ways of addressing memory for data types larger than a byte—the former is big endian (the most significant byte comes first) and the latter is little endian. As long as a data element is consistently accessed with the same data type, there is no problem. However, there are places in which a data type is written as several single bytes, then read as 2-byte or 4-byte quantities. We needed to identify and change the code in these places.

The original font data that described the glyph (shape) information for the text engine was a single block of 250K bytes of read-only data. This block was mapped to five blocks of resource data, since each block had to be less than 64K bytes for Windows 16-bit mode. These blocks are discardable, meaning that the operating system can load them when it needs to read some data, but to load other code or resource blocks when Windows has run out of memory, they can be replaced by other blocks.

The original firmware's text engine depended on a special hardware component that rotated font glyph data from horizontal to vertical orientation, could double the size of the data, and smoothed the edges of a glyph using several rules for HP Resolution Enhancement technology (REt). Since this hardware was not available to the printer driver, we were able to simulate the first and second of these functions in software. We determined that the print quality would still be better than the HP DeskJet 550C even if we did not simulate the REt rules. The resulting software simulation executes more slowly, but the orginal firmware design included a font cache, which minimizes the the number of times that we need to execute this function.

Some further syntax modifications were necessary. The printer driver is capable of supporting more than one of the same printer, for example, a printer on the LPT1 port and another on the LPT2 port, and these printers can be printing at the same time. For Windows to be able to execute multiple instances of the PCL emulator, the code must be compiled in the Windows *medium-memory model*. This required that many C-language pointer variables be designated *far pointers* rather than the more efficient *near pointers*. Also, some subtle syntax correction was necessary because an integer data type is 32 bits for the 68000, but 16 bits for the 80x86.

The PCL emulation implementation was accomplished in a staged development process. Two months before the first printer driver components to support the HP DeskJet 820C became available, we were able to build a DOS application that was totally decoupled from a printer driver. It would accept a test input stream of PCL data and map the input to an output file of raster data, which could be printed on the HP DeskJet 850C, which was mechanically identical to the target HP DeskJet 820C. Using our test center's extensive suite of input test files, we were able to stabilize the porting implementation, within the limits of the DOS application. For example, we noticed that the DOS memory allocation algorithm would fragment memory that was being continually allocated and freed, so that eventually a memory allocation request would fail. However, when we moved on to a subsequent stage in which we depended on the Windows memory manager, we found that this memory fragmentation no longer occurred. Once the DOS port was stabilized, we integrated the PCL personality into the printer driver, using the HP DeskJet 850C output target path, while still providing an input file of PCL. Next we introduced

and stabilized the DOS redirector input path. When the HP DeskJet 820C output target path finally became available, we were able to switch to it cleanly, and the PCL emulator became an effective tool to help stabilize the new output target path. Finally, we completed the target functionality, always building upon a stable base.

To summarize, by reusing original firmware code we were able to provide identical PCL functionality for PPA printers. Providing support for the Ed Interface API allowed the firmware code to be reused with little design modification.

▶ Go to Next Article

▶ Go to Journal Home Page