Printing with Natural and PostScript

Ron Babuka
Karen Durfee
Cornell University, CIT/Information Resources

presented at Colonial Caucus '95 (May 95)

Abstract/Introduction

The Department of Campus Life at Cornell University had used pre-printed forms in conjunction with impact printers to generate student contractual forms. These forms included: room contracts; room key cards; mailing cards; and confirmation letters. These three-part forms were expensive to purchase, ranging anywhere from $65 to $143 per 1000 and required a significant administrative effort to inventory at proper levels. The Department of Campus Life printed upwards of 50,000 forms every academic year, thus consuming a valuable portion of a limited budget. Additionally, a separate dedicated impact printer had to be purchased and maintained at the customer's facility to facilitate on-site printing.

Recently, the Information Resources division of Cornell Information Technologies proposed the substitution of pre-printed impact forms with PostScript printing technology. This resulted in the timely and efficient production of quality forms with a marked decrease in printing expense. For example, due to the markedly lower cost of standard weight three-part perforated paper, customers saved substantially. Furthermore, the system utilizes a standard PostScript laser printer.

This approach merges two technologies: mainframe Natural/ADABAS applications and PostScript laser printing. While each of these technologies may not be separately considered "leading edge," their merging does represent an innovative use of current technologies. The merging of these technologies produces quality forms, provides our customers with printing location flexibility, and substantially reduces printing costs. This paper and presentation will highlight these benefits and discuss the technical aspects of this solution.

About the Project

Why change from pre-printed to PostScript laser forms?

Until recently, the Campus Life Information System at Cornell University utilized pre-printed forms and impact printers to generate student room contracts and other agreements. While this method was appropriate in the 1980's, it became apparent over time that a more flexible, less expensive system, and more visually impressive output was needed.

The first issue leading to the development of a new printing method was the high cost of pre-printed forms. The cost of pre-printed forms varied from $70 per thousand {for a box and text format on heavy stock paper perforated to 8 1/2" X 5"} to $143 per thousand {for a pre-printed two-part housing contract on 8 1/2" X 11"}. Historically, the Campus Life office required up to 50,000 pre-printed forms per year which resulted in a relatively high average yearly cost of $5,200.

The second issue leading to the development of a new printing method was the obsolescence of impact printers. For example, room contracts were generated by using a three-part carbonless form on an impact printer. More than 50% of these contracts were generated on a high-speed impact printer at Cornell Information Technologies, the University's centrally-located computer operation. Campus Life, located at a different campus location, needed the ability to generate contracts on-site when students signed up for living spaces at the main office. This required Campus Life to own, maintain, and operate a dedicated impact printer for only approximately 2,000 pages of three-part carbonless pre-printed forms per year. This system worked until Cornell Information Technologies decided to phase out the centrally-located high-speed impact printer. When this decision was made, Campus Life was concerned that if its impact printer was ever 'down' for an extended period of time, its inability to print contracts on-site would severely hamper its business function.

The third issue leading to the development of a new printing method was the appearance of the forms. The impact printed fonts

on pre-printed forms did not adequately convey the image of Cornell University as a leader in education, research, and technology. Basically, Campus Life was using an "early 60's" printing technology to process the students of the "90s."

While the technology described was becoming obsolete, a new and better technology was emerging using PostScript. PostScript offered a number of advantages over the use of high impact printers. We had in place at CIT, a high speed Xerox Docutech printer that could process PostScript documents. Additionally, the Apple LaserWriters and Hewlett-Packard LaserJet printers in Campus Life's inventory could also process PostScript documents. For us this meant that the large printing transactions, upwards of 6,000 contracts/forms, could be easily processed by CIT. and the everyday small printing transactions could be processed by Campus Life with no need for new printing equipment. It also meant that Campus Life could be more flexible in its printing locations due to the relatively basic law of physics that it is easier to move an Apple LaserWriter NTX than it is a 200 pound IBM. impact printer.

Standardize forms to use stock paper

We also felt that this was a good time to standardize the weight, color, size, and specialty perforations of the paper that Campus Life would need to purchase and store. This may not sound significant, but it bears repeating that Campus Life historically consumed 50,000 sheets of form paper alone. The goal was to standardize the forms to one type and design the PostScript forms around the paper.

After close examination we found that two of the forms were of standard width paper and that the length was approximately 1/3 that of standard length paper. We realized it would be easy to place three forms on one sheet of paper if the paper was perforated in thirds.

To integrate the housing contract into a standard form required a bit more thought. The previous contracts were approximately six inches long and were designed as a two-part carbonless form. The original concept consisted of a two-page document. Students signed

the form and retained the second page, which had a copy of their signature for their own records. With the advent of PostScript, a design emerged with a one-page contract that worked around the three-perforated sections per page. The top section constituted the contract, which the student signed and mailed back to the Campus Life department. The second section remained an almost exact copy of the first section. The third and last sections would contain the mailing addresses of all roommates and the mailing address for the students themselves.

After these changes were approved by the Cornell University Counsel's Office, there only remained one last issue to be solved. The staff at Campus Life requested heavy stock paper that could withstand the abuse of mailings, multiple filings, and the students themselves. A quick check on reference data identified 32 pound paper as the heaviest weight that could be repeatedly processed by Apple and Hewlett-Packard laser printers and also fit well within the range of weight limits of the Docutech printer. We now had a standard form paper that could be used and purchased in bulk by Campus Life. This paper cost $19 per thousand as compared to $65 - $143 per thousand for pre-printed forms.

Possible solutions merging PostScript with Natural

The next step involved designing the manner in which to get the PostScript to the Docutech printer or to the local printers. We investigated three options.

There had been several smaller projects in which the actual PostScript code had been imbedded within the Natural application itself. As the Natural code was processed, PostScript was sent to the printer. While this method worked and used a minimum of system cycles, it did not meet our goal of fast and flexible access to the PostScript code or form by either a programmer or the customer. In order to make a small change, the program needed to be checked out, modified, tested, validated, and checked in, thus the concern for efficiency.

Another method has successfully stored PostScript on external drives connected to printers. This solution was more flexible but we

found the actual access and modification of forms in this manner to be complex and time consuming. This solution also limited which printer could be used for printing because only printers with external hard drives could store the PostScript form. This solution was also dependent on external drives properly working all the time with little or no support, and our past history has shown that disk failures or system crashes often required more time to correct than available.

The third option we investigated was to store the PostScript code in tables on a table file. This solution could meet the need for fast and simple access for the programmers and customers. We could easily build a procedure to read the table entries and send the PostScript code to the printers.

Advantages of storing PostScript in a table

Once we felt comfortable with this plan, the advantages of this method become apparent:

1. Any PostScript code stored on our tables files would be backed up by a dedicated support staff at no additional cost to the customers.
2. The storage "returns" fields in our table file were 253 characters long, thus able to store long PostScript strings, if the developers or customers preferred that method.
3. CIT. had designed and built many callable routines that sequentially read the table entries.
4. A sequential reading of a PostScript file could be loaded into memory in an array. This array could then be used for processing, eliminating as many file reads as possible. This would increase the speed and lower the processing cost.

Non-Form use

Currently there are four previously pre-printed forms that are generated by this method of PostScript retrieval. There are another five forms that are generated that involve little or no complex PostScript. These five forms are letters that are sent to students that involve some use of shading and small check boxes. It has been

found that this method allows for the production of high quality letters with a variety of fonts, shades, and graphics common to desktop publishing at a volume that an average office-place workstation and printer would not be able to efficiently match.

Cost Analysis

The design for this method of printing required approximately 8 hours, starting from the initial concept to the final implementation. The effort required to design and build the four PostScript forms required roughly 40 hours. The modification to the Natural code required approximately 35 hours to modify, test, and analyze. When these hours are multiplied by the hourly charge to a customer, a total development/replacement cost of $3,800 was accumulated.

There was a small increase in system utilization charges due to the periodic reading of an external tables file. Docutech printing is more expensive than impact printing. Combined, these charges averaged $10 per 1000 forms generated. Over a one-year period, these charges amounted to roughly an additional $500.

The first year implementation ($3,800) and operation expense ($500) accumulated to $4,300. Historically, the Campus Life department consumed $5,200 per year in pre-printed forms. The same volume of PostScripted forms on standard paper calculates to $800 per year. That is a savings in cost for paper of $4,200. When the implementation and yearly operation cost ($4,300) are compared to the savings ($4,200), the first year cost to Campus Life was $100!!

Since the implementation of the original PostScript forms, our staff has created several new forms, and has found that a realistic estimate of 7 - 9 hours, depending on the complexity of the form, is required to design, test, and load the PostScript form. The modification of pre-existing Natural Code takes a bit longer than producing the Natural Code from scratch, but our experience has shown that 5 - 10 hours is required, depending on the complexity of the existing code.

It is important to include the intangible benefits of this project. These benefits include the newfound flexibility of the PostScript forms, the increased reliability of the Docutech and local laser

printers, and the clean and professional appearance of the forms. These benefits are impossible to factor into the cost/benefit equation.

Technical Implementation

This section explains such things as PostScript basics, Corbel's Natural computing environment, how to print the form with merged data, how to store the PostScript form in a Natural/ADABAS table, and the Natural source code.

Things to know about PostScript

- PostScript is a page description language. It is a device independent standard for representing a printed page. It uses x and y coordinates for page location coordinates.
- Each PostScript file has two sections: the prologue and the script. The prologue contains the header information, font information, and definable "subroutines." The script is the executable code.
- The 'showpage' PostScript command allows one page of a PostScript file to be printed.
- PostScript lays down images in an opaque manner. In other words, if a PostScript "item' is laid down over another "item," the top-most item is seen and the bottom-most item cannot be seen. Therefore, order is important when producing PostScript code.

Things to know about Cornell's Natural computing environment

- Cornell University has Natural/ADABAS in a CMS mainframe environment.
- The PostScript form is stored in the table system. Actual data is stored in variables. The variables and associated page location coordinates, along with the 'showpage' command, are embedded within the Natural code. The variables data values and associated page location coordinates are stored in a workfile, then sent to the printer (after the code for the form is sent to the printer).
- The Natural application allows views and updates to the table. The table has its own internal security to verify access.

To print form with data

The printfile must contain the following items, in this order, for the PostScript interpreter to convert the text (PostScript code) to a PostScript image. The following items would produce a form merged with actual data.

1. prologue:
   %!PS (must be first line of every PostScript job)
   font definitions
   "subroutines" definitions

2. script (executable):
   perform "subroutines" definitions
   variables (data) with associated coordinates
   'showpage'

To achieve a printfile with these elements, the following should occur in your source code:

Procedure 1:

1. data record retrieval and manipulation
2. identification of which form will be used
3. (optional) "batching" information - how many forms to "group" into a packet to be send to the printer
5. a write statement of data to a workfile (where each record = 1 line)
6. fetch or call to procedure 2, passing variables (form name and workfile)

Procedure 2 (generic procedure that merges data with form and sends printfile to printer):

1. sets tag & spool
2. sends PS-Header to printer - %!PS
3. reads prologue from the table, stores it in an array (optional, if "batching") , and sends it once to the printer per each batch
4. reads script, stores it in an array, and sends to printer for each record

5. reads workfile and sends data with associated page location coordinates to the printer for each record
6. issues showpage for each record (form with data)
7. quits when end of workfile reached

As procedure 2 reads through the table, each prologue line is loaded into an array, and each script line is loaded into a separate array until a new form is reached sequentially in the table. When it is time to print, procedure 2 will read though the PostScript arrays and send each line to the printer at the appropriate time. There is a minimal amount of PostScript embedded within the Natural program. This is traditionally information associated with the data such as variables. The idea is to keep the generic code for the PostScript form separate from any data.

A typical printfile with 3 records would contain the following elements (in order):

```
%!PS
font definitions
subroutines
perform subroutines and other executable code
data for record 1
showpage
perform subroutines and other executable code
data for record 2
showpage
perform subroutines and other executable code
data for record 3
showpage
```

Storing the PostScript form into a Natural/ADABAS table

Step 1:

In a PostScript programmer's application, such as LaserTalk, write and test PostScript code to generate the form. The PostScript programmer's application should support downloading a PostScript file to the printer for the PostScript interpreter to process.

An PostScript programmer's application, such as LaserTalk, can be used for building the forms. This tool allows any individual trained in PostScript to create PostScript code and download the code to any PostScript laser printer. An individual can use something like LaserTalk to enter the PostScript code and then download it to a printer to test the new code, thus alleviating the need to generate PostScript within a Natural application or to manipulate an ADABAS source code table to test a new form.

Example of PostScript code used to generate Campus Life form:

```
 %
(I am graduating:) 280 583 MS
(January) 350 583 MS (May) 400 583 MS
(Forwarding Address:) 280 574 MS
(Signature) 280 554 MS
 %
375 583 moveto 20PointLine %january
414 583 moveto 20PointLine %may
350 574 moveto 180PointLine %forwarding address1
350 565 moveto 180PointLine %forwarding address2
310 554 moveto 220PointLine %signature
} bind def
 %
/StudentInfoSection
{.7 setlinewidth
(Student Name) 115 735 F2 SMS
(Social Security No.) 235 735 MS
(College) 320 735 MS
(CL) 355 735 MS
(Bldg Rm) 395 735 MS
(Telephone No.) 450 735 MS
(Mailbox No.) 510 735 MS
 %
(Home Address) 90 723 MS
(Home Phone) 275 723 MS
(Birth Date) 395 723 MS
```

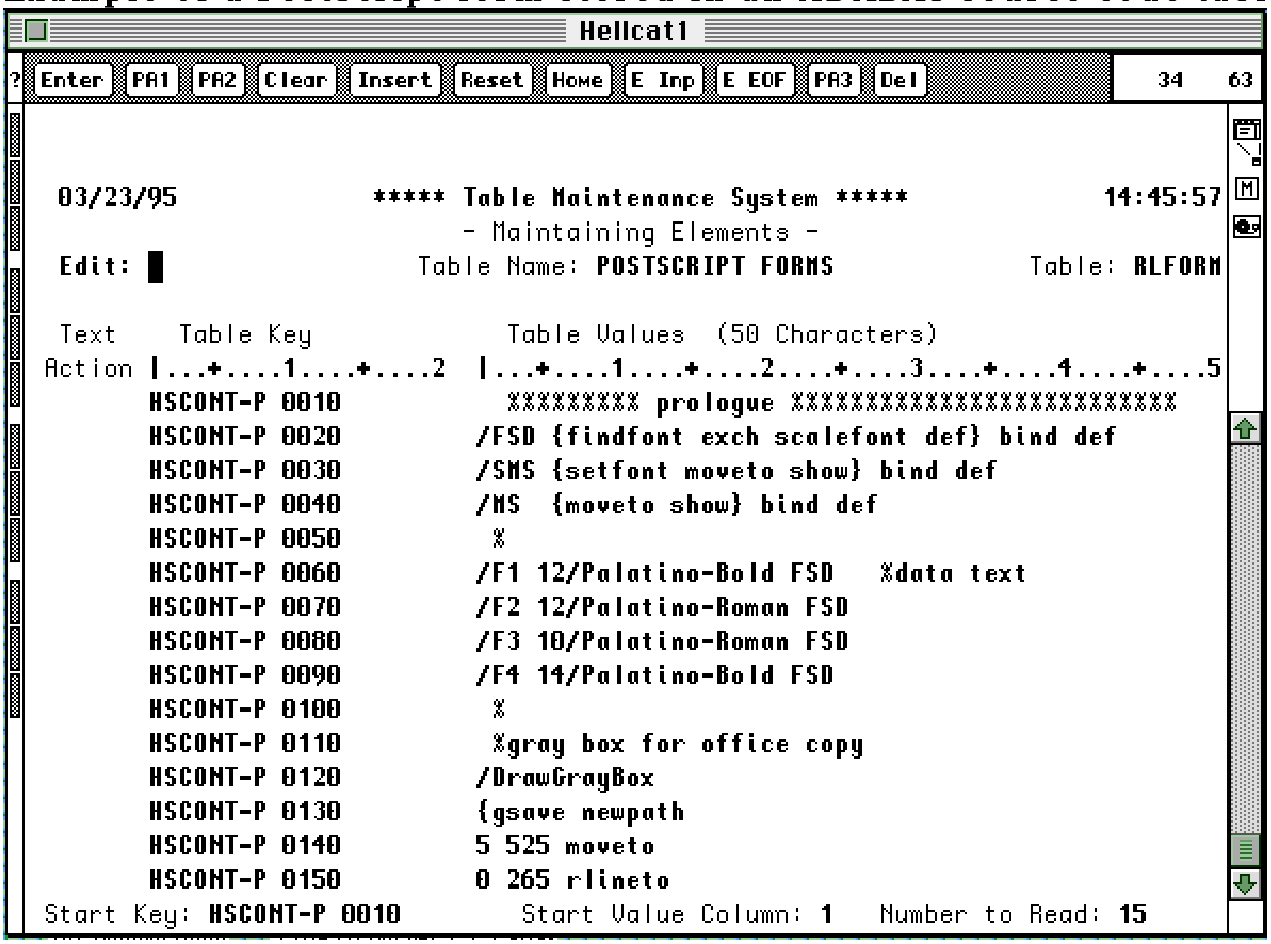
Our ultimate goal is to train Campus Life staff members in PostScript and have them generate a PostScript form in LaserTalk so that our staff can load the PostScript code into an ADABAS source table. This method should save the customers expensive programmer time and charges.

Step 2:

Upload PostScript file from workstation to CMS on mainframe.

Step 3:
Prepare the new CMS file for uploading into Natural table. Keep in mind the concept of what PostScript code will look like in the Natural table system:

Example of a PostScript form stored in an ADABAS source code table:



In the above table, named RLFORM, the Table Key is divided into form name and line number. The form name is HSCONT (-P for prologue and -S for script). The line numbers used for sequence start at 0010 and go until the end of the prologue section. The Table Key used in the script section would be similar, starting at 0010, and with a -S instead of -P for the form name. The PostScript code is stored in the "Table Values" area on the right side of the screen.

<u>Natural code</u>:

The Natural PostScripting code is started after the database information is manipulated for each student processed. A call is made to a main controlling subroutine.

As shown in the figure below, this portion will utilize code that sends the prologue to the printer only with the first record in every batch. The other calls are to subroutines which print the script and data.

```
Enter  PA1  PA2  Clear  Insert  Reset  Home  E Inp  E EOF  PA3  Del          63   1
 RLB11P   NATURAL2 A1  F 96  Trunc=96 Size=1184 Line=1083 Col=1 Alt=1235

      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== 5375 DEFINE SUBROUTINE POSTSCRIPTING
===== 5380   ADD 1 TO #FORMS-IN-BATCH-COUNT
===== 5385*
===== 5390* 'if' needed so that only one psheader and prologue gets send to
===== 5395* printer with each 'batch' of 500 (instead of sending psheader and
===== 5400* prologue with each form)
===== 5405   IF #FORMS-IN-BATCH-COUNT = 1 THEN
===== 5410     #NEED-2-SEND-PSHEADER-&-PROLOGUE    = TRUE
===== 5415   ELSE
===== 5420     #NEED-2-SEND-PSHEADER-&-PROLOGUE    = FALSE
===== 5425   IF #NEED-2-SEND-PSHEADER-&-PROLOGUE THEN DO
===== 5430     PERFORM POSTSCRIPT-HEADER
===== 5435     PERFORM PRINT-PROLOGUE
===== 5440     PERFORM GET-SCRIPT
===== 5445     RESET #NEED-2-SEND-PSHEADER-&-PROLOGUE
===== 5450   DOEND
===== 5455*
===== 5460   PERFORM PRINT-SCRIPT
===== 5465   PERFORM PRINT-DATA
====>
                                                        X E D I T  1 File
```

As shown in the figure below, this subroutine will read the table RLFORM (through a CIT external subroutine TEDIT29) for the PostScript prologue (HSCONT-P) and then send the prologue to the printer.

```
Enter  PA1  PA2  Clear  Insert  Reset  Home  E Inp  E EOF  PA3  Del        77  2
 RLB11P   NATURAL2 A1  F 96  Trunc=96 Size=1182 Line=925 Col=1 Alt=1509

      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== 4585 DEFINE SUBROUTINE PRINT-PROLOGUE
===== 4590    #FORM-NAME = 'HSCONT-P'
===== 4595    #LINE-NUM = '0010'
===== 4600    #TAB-NO = 'RLFORM'
===== 4605    REPEAT    /* read ps form from table
===== 4610      CALLNAT 'TEDIT29' #CODE #ERR-CODE #ERR-MSG #TAB-NO #RETURNS
===== 4615      IF #ERR-MSG = ' ' THEN DO
===== 4620        IF #FORM-NAME EQ 'HSCONT-P' THEN DO
===== 4625          #RETURNS2(A110) = #RETURNS
===== 4630          WRITE (1) NOTITLE NOHDR #RETURNS2
===== 4635        DOEND /* (5360) #form-name = 'hscont-p'
===== 4640        ELSE DO
===== 4645          IF #FORM-NAME NE 'HSCONT-P' THEN
===== 4650            ESCAPE BOTTOM
===== 4655        DOEND  /* else do
===== 4660      DOEND /* if #err-msg = blank
===== 4665    CLOSE LOOP  /* repeat to read form from table
===== 4670 RETURN  /* subroutine print-prologue
===== 4675*
====>
                                                          X E D I T  1 File
```

As shown in the figure below, the next subroutine will load the script of the PostScript code into an array. This array can be written sequentially to a printer to produce PostScript output (as long as the prologue was sent ahead of it in the same printfile). The array can be written as many times as there are forms needed.

```
Enter  PA1  PA2  Clear  Insert  Reset  Home  E Inp  E EOF  PA3  Del     81  2:
 RLB11P   NATURAL2 A1  F 96  Trunc=96 Size=1180 Line=944 Col=1 Alt=1511

      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== 4680 DEFINE SUBROUTINE GET-SCRIPT
===== 4695    RESET #SCRIPT-CODE(A110/20) /* 20 lines of script code in table
===== 4700    #FORM-NAME = 'HSCONT-S'
===== 4705    #LINE-NUM = '0010'
===== 4710    #TAB-NO = 'RLFORM'
===== 4715    REPEAT    /* read ps form from table
===== 4720      CALLNAT 'TEDIT29' #CODE #ERR-CODE #ERR-MSG #TAB-NO #RETURNS
===== 4725      IF #ERR-MSG = ' ' THEN DO
===== 4730        IF #FORM-NAME EQ 'HSCONT-S' THEN DO
===== 4735          ADD 1 TO #SCRIPT-LINE-COUNT(N3)
===== 4740          #SCRIPT-CODE(#SCRIPT-LINE-COUNT) = #RETURNS
===== 4745        DOEND /* (4730) #form-name = 'hscont-s'
===== 4750        ELSE DO
===== 4755          IF #FORM-NAME NE 'HSCONT-S' THEN
===== 4760            #TOTAL-LINES-IN-SCRIPT(N3) = #SCRIPT-LINE-COUNT
===== 4765            RESET #SCRIPT-LINE-COUNT  /* needed for batching
===== 4770            ESCAPE BOTTOM
===== 4775        DOEND /* (4750)
===== 4780      DOEND /* if err-msg = blank
====>
                                                             X E D I T  1 File
```

In the above example, CALLNAT 'TEDIT29', is a call to an external table unload procedure to unload the appropriate form's script (HSCONT-S) into an array.

As shown below, the script array can then be printed when a call to procedure PRINT-SCRIPT is made within our main processing subroutine.

```
Enter  PA1  PA2  Clear  Insert  Reset  Home  E Inp  E EOF  PA3  Del        87  2
 RLB11P   NATURAL2 A1  F 96  Trunc=96 Size=1180 Line=966 Col=1 Alt=1766

      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== 4790 DEFINE SUBROUTINE PRINT-SCRIPT
===== 4795* sends script (which was stored in an array) of postscript
===== 4800* form to printer
===== 4805   FOR #INDEX(N3) = 1 TO #TOTAL-LINES-IN-SCRIPT
===== 4810     WRITE (1) NOTITLE NOHDR
===== 4815       #SCRIPT-CODE(#INDEX)
===== 4820   LOOP
===== 4825 RETURN  /* subroutine "PRINT-SCRIPT"
===== 4830*
===== 4835 DEFINE SUBROUTINE PRINT-DATA
===== 4840* writes all data on form
===== 4845   WRITE (1) NOTITLE NOHDR
===== 4850     / ' %%%%%% print data %%%%%%% '
===== 4855* office copy
===== 4860     / '(' *DATU ') 40 750 F1 SMS'
===== 4865     / '(' #RL-YEAR-PLUS-ONE ') 360 736 MS'
===== 4870     / '(' #STUD-NAME ') 60 708 MS'
===== 4875*    / '(' #CONTRACT-YEAR ') 400 694 MS'
===== 4880     / '(' #LINE-DISP ') 70 680 MS'
====>
                                                          X E D I T  1 File
```

In the above figure, the second subroutine, PRINT-DATA, will print the Natural variables that have processed data represented within. This information is printed last so that it overwrites any form shading, boxes, text, or lines. (Remember, PostScript lays down images in an opaque fashion.) As shown, the proper page location coordinates must be passed along with the variables.

Finally, a showpage subroutine is called to send a required 'showpage' in order to print the form with merged data. In the example below, the SHOWPAGE subroutine also includes batching "flags" to monitor and update the flags that control the batch size of 500 forms per output group. This "batching" source code is optional.

```
Enter  PA1  PA2  Clear  Insert  Reset  Home  E Inp  E EOF  PA3  Del        96  2
 RLB11P   NATURAL2 A1   F 96  Trunc=96 Size=1180 Line=1029 Col=1 Alt=1772

      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== 5105     / '(' #MESSAGE-2 ') 40 36 MS'
===== 5110 RETURN  /* subroutine "PRINT-DATA"
===== 5115*
===== 5120 DEFINE SUBROUTINE SHOWPAGE
===== 5125* issues a showpage at the end of each form; sends last 'batch',
===== 5130* if less than 500 to printer with 'define printer'
===== 5135   WRITE (1) NOTITLE NOHDR
===== 5140     / 'showpage'
===== 5145* for batching of 500
===== 5150   IF #FORMS-IN-BATCH-COUNT GE 500 THEN DO
===== 5155     DEFINE PRINTER (1)
===== 5160     RESET #FORMS-IN-BATCH-COUNT
===== 5165   DOEND
===== 5170 RETURN  /* subroutine "SHOWPAGE"
===== 5175*
===== 5180 DEFINE SUBROUTINE RESET-DATA-VARIABLES
===== 5185   RESET
===== 5190     #STUD-NAME
===== 5195*    #CONTRACT-YEAR
====>
                                                          X E D I T  1 File
```

Summary

The implementation of PostScript printing has resulted in many benefits to the Campus Life and Cornell University. It has eliminated Campus Life's reliance on obsolete hardware, provided more flexibility in appearance and location of output, represented Campus Life and Cornell University in a more professional manner, and most importantly, reduced operational costs.

Impact printers are problematic and expensive to maintain. PostScript laser printing has virtually eliminated the need for Campus Life to own and maintain obsolete hardware.

PostScript forms (and letters) look better and thereby present Campus Life, CIT and Cornell University more professionally. Campus Life staff report one unexpected result: the "more official forms" apparently have impressed students enough that they are less likely to question and argue against system-generated information and assignments.

Additionally, the use of PostScript allows Campus Life more flexibility in meeting the ever-changing needs of students promptly and efficiently by allowing quick and efficient modification of text. Under the previous system, the university ordered pre-printed forms. Thereafter, any changes to forms could not take place without consideration being given to the cost, or until existing supplies of pre-printed forms were used.

Overall, use of the PostScript system has resulted in a sizable decrease in costs of administering the campus life program. One savings is reflected by the fact that Campus Life no longer needs to own a dedicated impact printer or pay for the cost of maintenance or hardware breakdowns. Since Campus Life staff will be able to design its own forms and letters, when properly trained in PostScript language, it will not be necessary to pay an external department like CIT for this time-intensive, but relatively simple task. Campus Life also benefits from the use of standard blank paper which it can purchase in large volumes at a large discount. The lack of pre-printing allows Campus Life to receive more bids for sale from a larger number of vendors, which should also lower the price.

An average of costs over the past year illustrates this savings. Prior to the of implementation of PostScript, the average cost of pre-printed forms was approximately $5,750 per year. Today, the current average annual cost to Campus Life from the use of standardized paper with PostScript is approximately $800 per year. If one were to extend these savings over a five year period, the savings would be more than $22,000!!

Current state and local budgetary pressures have resulted in internal pressure to make cost effective cuts. Cornell University is no exception in this need to reduce its operational costs. While the dollar savings mentioned may not seem large when compared with the Cornell University's overall operational budget, cost cutting plans such as PostScript printing is sizable over time. More importantly, the end product and service provided are far superior to the previous, more expensive paradigm.

To reach us:

Ron Babuka wrb1@cornell.edu

Karen Durfee kld3@cornell.edu