LEVEL 68

MULTICS PAGE PROCESSING SYSTEM UTILITY MANUAL

SUBJECT

Description of Software for the Support of the Honeywell Page Processing System on Multics

SPECIAL INSTRUCTIONS

The software documented in this manual supports both the Honeywell Page Printing Sytem (PPS) and the Honeywell Page Processing System (PPS II).

Some sections of this manual are written for users already familiar with the procedures for making input/output tape attachments. The Multics I/O system is described in the MPM Reference Guide (Order No. AG91), and I/O module descriptions can be found in MPM Subroutines (Order No. AG93) and MPM Peripheral Input/Output (Order No. AX49).

SOFTWARE SUPPORTED

Multics Software Release MR8.0
NIPOLOS IBM Support Release 3.21

ORDER NUMBER

CJ97-00

May 1980

Honeywell

PREFACE

This document describes the current state of the Multics Page Processing System support. The software described here is compatible with NIPOLOS IBM support release 3.21. Changes in PPS software may have an effect on the Multics interface described here.

The software described in this manual was previously installed in the Multics experimental library. This is its first official release.

When the term Multics is used as a noun in this document, it is meant to refer to the Multics operating system. Similarly, when the acronym PPS is used without parentheses, it is meant to refer to both the Page Printing System (PPS) and the Page Processing System (PPS II).

File No.: 1L13

CJ97-00

CONTENTS

ILLUSTRATIONS

SECTION 1

INTRODUCTION

This document describes the software provided in the Multics system to support the Honeywell Page Processing System.

THE PAGE PROCESSING SYSTEM

The Page Processing System (PPS) is an off-line, non-impact printing system capable of printing up to 210 pages per minute (equivalent to 18,000 lpm). The PPS consists of one or more read-only tape drives, a system controller, a print unit, and one or more stacker units.

The read-only tape units are capable of reading tapes written at 556, 800, or 1600 bpi.

The system controller is a Honeywell 716 or Level-6 processor.

The print unit employs a fixed electrographic print mechanism that moves the paper at a constant rate of 30 inches per second (20 ips on slower models) past a format drum containing the image of a "preprinted form" for fixed data such as horizontal lines or company logo. This format drum is mounted by the operator.

After passing this drum the paper passes a print station capable of printing 132 character lines read from the input tape. This unit is capable of printing with 4, 6, 8, or 10 lines per inch vertically and 10 or 12 characters per inch horizontally. The print unit is also capable of printing up to 255 copies of a report while reading the input tape only once. Also included in the print mechanism is a paper cutter and hole punch which allow a variety of page sizes and punched hole configurations.

Each stacker unit provides 8 trays, each capable of holding up to 500 sheets. Stacker algorithms supported include stacking one report per tray, one copy per tray (for multiple copy reports), and simple overflow from one tray to the next with or without separator sheets (a separator sheet is a blank page slightly longer than a printed page).

MULTICS INTERFACES

Two user interfaces to the PPS are available on Multics. They are both described in this document. They are: the pps_ I/O module and the make_pps_tape command.

The first, the pps_ I/O module, requires the user to take an active part in the preparation of the PPS tape. The user (or user's program) must attach, open, write to, close and detach the I/O switch being used. This method also requires access to use a tape drive on the Multics system. Examples demonstrating the use of the pps_ I/O module are given in Section 2, and a complete description of the I/O module can be found in Section 5. A complete description of the make_pps_tape command can be found in Section 4.

The second interface, the make_pps_tape command, requires no extra effort on the part of the user or the system administrator. The user need only have access to write the PPS tape on the Multics system.

The I/O module approach is usually better for producing large reports because it offers the user maximum control over the PPS functions employed in the production of the output and does not require that an intermediate copy of the report be online. The make_pps_tape command allows individual users (or groups of users) with printable files to produce tapes that can be processed on a PPS system.

SECTION 2

USE OF THE pps_ I/O MODULE

The pps_ I/O module is designed to give the user maximum control over the PPS reports produced. It provides for the generation of multiple report tapes while allowing full control over page format (i.e., page labels, indentation, channel stops, etc.).

USING THE I/O MODULE

The pps_ I/O module can be used with the I/O features of the Multics programming languages or by calling the iox_ subroutine directly to produce tapes that can be printed offline on the Page Processing System. There are no special restrictions on the number of reports per volume or the number of volumes per report.

Using the I/O Module with Language I/O

The pps_ I/O module should be used to attach and then open an I/O switch for each separate file being copied. The only opening mode supported is stream_output. Each time the I/O switch is attached and opened a new report is started on the output tape. Once the report has been written on the pps tape, the I/O switch is closed and detached and is then ready for another report. Subsequent reports simply require that the user repeat this attach, open, write, close, detach sequence as often as necessary.

For multiple reports, each attachment except the last one should be made with the -retain all option. The last attachment should be made with the -retain none option. These attachments can also be made with the retain_all and retain_none control orders rather than the -retain control argument.

Using the I/O Module with iox_

When the iox_ subroutine is called directly, multiple report generation can be simplified greatly by using the new_report control order. For details, see the example in figure 2-3 and the description of the new_report control order in Section 5.

PROGRAMMING EXAMPLES

The examples that follow in this section show how a user can produce PPS output reports using the supported languages of Multics. See Section 5 for a complete description of the pps_ I/O module.

The PL/I example in figure 2-1 is designed to transfer a single report to an output tape. It queries the user for two arguments: the name of the file to be printed and the tape on which the output is to be placed. The program opens the input and output files, reads and writes the text, and finally closes the two files.

The example of figure 2-2 shows the steps that must be taken to produce multiple reports on the same tape. The extra steps involved are required by the operation of PL/I I/O.

Figure 2-3 shows yet another method of producing multiple reports on the same tape, but it does not use the -retain all control argument. Instead it uses the control order new_report, which is preferable because the program does not open and close the output file for each report and thus avoids the risk of leaving the output tape mounted but unattached.

```
example1: proc ();

dcl  buffer char (1000) varying;          /* text buffer */
dcl  com_err_ entry options (variable);
dcl  endfile condition;                   /* the end of file condition */
dcl  file_name char (168);                /* file to be displayed */
dcl  input file;                          /* the input file */
dcl  output file;                         /* the output file */
dcl  pl1_io_$error_code entry (file) returns (fixed bin (35));
dcl  sysin input file;                    /* user_input */
dcl  sysprint print file;                 /* user_output */
dcl  tape_name char (32);                 /* reel id of the output tape */
dcl  undefinedfile condition;             /* in case of error */


          put skip list ("File name ?");
          get list (file_name);
          on undefinedfile (input) begin;
               call com_err_ (pl1_io_$error_code (input), "example1",
                    "Cannot open input file.  File name = ^a.", file_name);
               goto exit;
          end;
          open file (input) title ("vfile_ " || file_name) stream input;
          revert undefinedfile (input);

          put skip list ("Tape name ?");
          get list (tape_name);
          on undefinedfile (output) begin;
               call com_err_ (pl1_io_$error_code (output), "example1",
                    "Cannot open output file.  Volume = ^a", tape_name);
               close file (input);
               goto exit;
          end;
          open file (output) title ("pps_ -volume " || tape_name) print stream output ;
          revert undefinedfile (output);

          on endfile (input) goto all_done;  /* watch for end of file */
          do while ("1"b);
               read file (input) into (buffer);   /* get a line */
               write file (output) from (buffer); /* and write it */
          end;
all_done:                                   /* we have read all data */
          close file (input), file (output); /* close all files */
exit:  return;
     end example1;
```

Figure 2-1. A simple PL/I example.

```
example2: proc ();

dcl  buffer char (1000) varying;        /* text buffer */
dcl  code fixed bin (35);               /* error code from iox */
dcl  com_err_ entry options (variable);
dcl  endfile condition;                 /* the end of file condition */
dcl  file_name char (168);              /* file to be displayed */
dcl  input file;                        /* the input file */
dcl  output file;                       /* the output file */
dcl  output_opened bit (1) init ("0"b); /* ON => tape was opened once */
dcl  pl1_io_$error_code entry (file) returns (fixed bin (35));
dcl  sysin input file;                  /* user_input */
dcl  sysprint print file;               /* user_output */
dcl  tape_name char (32);               /* volume name for output tape */
dcl  undefinedfile condition;           /* in case of error */


          put skip list ("Tape name ?");
          get list (tape_name);

          do while ("1"b);
TRY_AGAIN:
               put skip list ("File name ?");
               get list (file_name);
               if file_name = "*" then goto EXIT;
               on undefinedfile (input) begin;
                    call com_err_ (pl1_io_$error_code (input), "example2",
                         "Cannot open input file.  File name = ^a.",
                         file_name);
                    goto TRY_AGAIN;
               end;
               open file (input) title ("vfile_ " || file_name) stream input;
               revert undefinedfile (input);

               on undefinedfile (output) begin;
                    call com_err_ (pl1_io_$error_code (output), "example2",
                         "Cannot open output file.  Volume = ^a.", tape_name);
                    close file (input);
                    goto EXIT;
               end;
               open file (output) title ("pps_ -volume " || tape_name || " -ret all")
                    print stream output ;
               revert undefinedfile (output);
               output_opened = "1"b;

               on endfile (input) goto ALL_DONE;  /* watch for end of file */
               do while ("1"b);
                    read file (input) into (buffer);   /* get a line */
                    write file (output) from (buffer); /* and write it */
               end;
ALL_DONE:                               /* we have read all data */
               close file (input), file (output); /* close all files */
          end;

EXIT:
          if output_opened then do;     /* tape ever attached? */
               open file (output) title ("pps_ -volume " || tape_name || " -ret none")
                    print stream output;
               close file (output);
          end;
          return;

     end example2;
```

Figure 2-2. An example of writing multiple reports.

```
example3: proc ();

dcl buffer char (1000) varying;          /* text buffer */
dcl com_err_ entry options (variable);
dcl endfile condition;                   /* end of file condition */
dcl file_name char (168);                /* file to be displayed */
dcl input file;                          /* the input file */
dcl output file;                         /* the output file */
dcl iocbp ptr;                           /* output IOCB ptr */
dcl iox_$control entry (ptr, char (*), ptr, fixed bin (35));
dcl code fixed bin (35);                 /* error code */
dcl pl1_io_$error_code entry (file) returns (fixed bin (35));
dcl pl1_io_$get_iocb_ptr entry (file) returns (ptr);
dcl sysin input file;                    /* user_input */
dcl sysprint print file;                 /* user_output */
dcl tape_name char (32);                 /* reel_id of output tape */
dcl undefinedfile condition;             /* in case of error */


      put skip list ("Tape name ?");
      get list (tape_name);
      on undefinedfile (output) begin;
            call com_err_ (pl1_io_$error_code (output), "example3",
               "Cannot open output file.  Volume = ^a", tape_name);
            goto EXIT;
         end;
      open file (output) title ("pps_ -volume " || tape_name)
         print stream output;
      revert undefinedfile (output);

      do while ("1"b);
         put skip list ("File name ?");
         get list (file_name);
         if file_name = "*" then goto ALL_DONE;
         on undefinedfile (input) begin;
               call com_err_ (pl1_io_$error_code (input), "example3",
                  "Cannot open input file.  File name = ^a.",
                  file_name);
               goto NEXT;
            end;
         open file (input) title ("vfile_ " || file_name) stream input;
         revert undefinedfile (input);
         iocbp = pl1_io_$get_iocb_ptr (output);
         call iox_$control (iocbp, "new_report", null (), code);
         if code ^= 0 then do;
            call com_err_ (code, "example3",
               "Unable to begin new report.");
            goto EXIT;
         end;
         on endfile (input) begin;
               close file (input);       /* close this input file */
               goto NEXT;
            end;
         do while ("1"b);
            read file (input) into (buffer); /* get a line */
            write file (output) from (buffer); /* and write it */
         end;
NEXT: end;

ALL_DONE: close file (output);           /* close all files */

EXIT: return;

   end example3;
```

Figure 2-3. Another example of writing multiple reports.

The FORTRAN example of figure 2-4 is simply a program to print the first 25 powers of 2. Before executing this program, the user must attach I/O switch file03 using the pps_ I/O module with an io command, such as follows:

```
io attach file03 pps_ w-100
```

where w-100 is the name of a tape volume. Another version of this program with the file attachment specified in an open statement is shown in figure 2-5. Note that this sample program always asks for tape volume w-100.

```
      do 10 i=1,25
10    write (03,11)i,2**i
11    format(1x,i5,i12)
      stop
      end
```

Figure 2-4. A simple FORTRAN example.

```
      open (03,form="formatted",attach="pps_ w-100",mode="out",
     &access="sequential")
      do 10 i=1,25
10    write (03,11)i,2**i
11    format(1x,i5,i12)
      stop
      end
```

Figure 2-5. An example of use of the open statement.

The COBOL example program in figure 2-6 reads the file attached to the insw I/O switch and writes the content of this file on the PPS tape. Before executing this program, the user must attach both insw and outsw I/O switches using an io command, such as follows:

```
io attach insw vfile_ xyz
io attach outsw pps_ w-100
```

where xyz specifies the file to be printed and w-100 is the name of a tape volume. Note that the COBOL program treats the PPS attachment as a printer and that the resultant report will be double spaced.

```
identification division.
program-id. example6.
environment division.
configuration section.
source-computer. Multics.
object-computer. Multics.
input-output section.
file-control.
    select external pps assign to outsw-printer.
    select external qaz assign to insw;
    organization is stream.
data division.
file section.
fd pps data record is pps-rec,
    label records are omitted.
01 pps-rec picture x(120).
fd qaz data record is qaz-rec,
    label records are omitted.
01 qaz-rec picture x(120).
working-storage section.
procedure division.
open-files.
    open input qaz.
    open output pps.
loop.
    read qaz record; at end go to close-files.
    move qaz-rec to pps-rec.
    write pps-rec after advancing 2 lines.
    go to loop.
close-files.
    close qaz, pps.
    stop run.
```

Figure 2-6. A COBOL program to list a file on the PPS.

The BASIC example shown in figure 2-7 is the equivalent of the FORTRAN program in figure 2-5. Note that this sample program always asks for tape volume w-100.

```
0010 file #1: ":foo pps_ -volume w-100"
0020 for i = 1 to 25
0030 print #1: i; 2^i
0040 next i
0050 end
```

Figure 2-7. A sample BASIC program.

SECTION 3

LIMITATIONS OF PPS SUPPORT ON MULTICS

There are limitations placed on the Multics user by the PPS interface described in this document. Some are avoidable by extra user coding, some are due to limitations of the Multics PPS support software, others are due to the PPS system itself.

PAGE FORMAT

The page format limitations are due to physical limitations of the PPS hardware. The PPS line length is limited to 132 print positions. That is, no print line can represent more than 132 columns of printed output. The page length is limited to a maximum of 93 lines per printed page. These values may be restricted further, of course, by the physical page dimensions. Tables 3-1 and 3-2 show the line length and page length limits for all of the allowable page sizes.

CHARACTER SETS

In all cases the Multics PPS support software attempts to produce PPS output that is visually similar to output produced by Multics on a terminal or line printer. With the PPF6025 hardware option on the destination PPS system almost all underscored text and many other overstruck character combinations are possible. If the user specifies the "-char_table ppf6025" control argument in the pps_ attach description, any overstruck character combination that can be reproduced on the PPS will be accommodated. Any combination not representable on the PPS will be displayed as a special "black box" character. This "black box" character is just as its name suggests - a one character black rectangular box. With the standard PPS font (known as NIP optimized), any underscored text will have the underscores removed and any other overstruck character combinations that cannot be reproduced on the PPS will be displayed as a "black box".

It is possible, using the cv_ppscf command, to define Multics ASCII strings (either single characters or sequences of overstruck characters) that represent any of the 8 bit characters of the PPS. See Section 4 for a description of cv_ppscf and Section 5 for a description of the -char_table control argument to the pps_ I/O module.

paper width (inches)	characters per line (physical line length)	
	small characters (pitch = 12.8)	large characters (pitch = 10)
5.0	58	46
5.5	64	51
7.5	89	71
8.0	96	76
8.5	102	81
11.0	132	106

Figure 3-1. Character positions per line.

sheet length (inches)	usable lines per page (physical page length)			
	lpi = 4	lpi = 6	lpi = 8	lpi = 10
3.0	8	14	20	26
3.5	10	17	24	31
4.0	12	21	28	36
4.5	14	23	32	41
5.0	16	26	36	46
5.5	18	29	40	51
6.0	20	32	44	56
7.0	24	38	52	66
8.0	28	44	60	76
8.5	30	47	64	81
10.0	36	56	76	93
11.0	40	62	84	93
12.0	44	68	93	93
14.0	52	80	93	93

Figure 3-2. Lines per page.

SECTION 4

MULTICS COMMANDS USED WITH PPS

The commands described in this section allow the user to create tables to control character conversions done by pps_ and to create listings tapes for printing on the PPS. The discussion below briefly describes the context of the various divisions of the command descriptions.

Name

The "Name" heading for each command lists the full command name and any abbreviated form. The name is usually followed by a discussion of the purpose and function of the command and the expected results from the invocation.

Usage

This part of the command description first shows a single line that demonstrates the proper format to use when invoking the command and then explains each element in the line. The following conventions apply in the usage line.

1. Optional arguments are enclosed in braces (e.g., {path}, {User_ids}). All other arguments are required.

2. Control arguments are identified in the usage line with a leading hyphen (e.g., {-control_args}) simply as a reminder that all control arguments must be preceded by a hyphen in the actual invocation of the command.

3. To indicate that a command accepts more than one of a specific argument, an "s" is added to the argument name (e.g., paths, {paths}, {-control_args}).

NOTE: Keep in mind the difference between a plural argument name that is enclosed in braces (i.e., optional) and one that is not (i.e., required). If the plural argument is enclosed in braces, clearly no argument of that type need be given. However, if there are no braces, at least one argument of that type must be given. Thus "paths" in a usage line could also be written as:

path1 {path2 ... pathn}

The convention of using "paths" rather than the above is merely a method of saving space.

Notes

Comments or clarifications that relate to the command as a whole are given under the "Notes" heading. Also, where applicable, the required access modes, the default condition (invoking the command without any arguments), and any special case information are included.

Examples

The examples show different valid invocations of the command. An exclamation mark (!) is printed at the beginning of each user-typed line. This is done only to distinguish user-typed lines from system-typed lines. The results of each example command line are either shown or explained.

cv_ppscf

Name: cv_ppscf

The cv_ppscf command converts a data file known as a PPS character file into a source segment that is then assembled to create a PPS character table. This character table can then be used to control the translation of Multics ASCII characters and overstruck character sequences to PPS characters (a modified EBCDIC character set).

Usage

cv_ppscf path {-control_args}

where:

1. path
   is the pathname of the PPS character file. If the suffix ppscf is not supplied it is assumed.

2. control_args
   can be chosen from the following:

   -list, -ls
   causes a listing file to be produced. The name of this file is the source path with the ppscf suffix replaced by the ppsctl suffix.

   -long, -lg
   causes a message reporting the usage of the available positions in the character matrix to be printed.

Notes

Execution of this command results in the creation of a segment called path.alm. The output segment must then be assembled using the Multics ALM assembler. The alm command is described in the MPM Subsystem Writers' Guide, Order No. AK92.

The PPS character file consists of lines of character definitions. The first line of the file is used to specify the default character and space character as two hexadecimal values separated by white space. The default character is used to represent any character or sequence that is not defined in the remainder of the character file. The space character defines the PPS equivalent for the ASCII space character. The remaining lines define Multics ASCII equivalents for the PPS characters (one definition per line).

A PPS character is defined by specifying the hexadecimal value for the character followed by the ASCII equivalent. This ASCII equivalent can be a series of ASCII characters, in which case the overstruck combination of these characters is defined. The sequence of ASCII characters can only contain printable characters.

A PL/I like comment can be specified at the end of any line in the PPS character file.

cv_ppscf

The delimiters used in parsing these definition lines are white space. This includes the characters space and horizontal tab.

The one restriction on any overstruck sequence that is being defined is that all subsets of the overstruck sequence must also be defined in the character file. This means that if a user is defining "A" overstruck with "_" (i.e. "A") both "A" and "_" must also be defined in the character file.

Examples

Listed below is the input file for the cv_ppscf command which defines three PPS characters (plus, minus, and underscore) and all of their combinations as overstruck characters. The hexadecimal representation is that of the ppf6025 character table shown in figure A-2 in Appendix A.

```
ff 40      /* define default and space characters */
04 +-_     /* now define PPS hex values for ASCII */
6d _       /* sequences listed */
4e +
60 -
31 -_
24 +_
```

make_pps_tape

Name: make_pps_tape

The make_pps_tape command is a convenient means by which a user can create a magnetic tape which can be printed on the PPS.

Usage

make_pps_tape target_spec paths

where:

1. target_spec
   can be one of the following:

   -volume volume_name, -vol volume_name
   specifies the name of the tape volume to be used. (see Notes below).

   -target_description attach_desc, -tds attach_desc
   specifies the attach description to be used (See Notes below).

2. path
   is the pathname of a file to be processed.

Notes

If the -volume control argument is specified the make_pps_tape command attaches a uniquely named I/O switch using an attach description of the form "pps_ -volume volume_name". If the -target_description control argument is specified the make_pps_tape command will attach a uniquely named I/O switch using the attach description specified. Use of the -volume control argument is recommended rather than -target_description because it is simpler.

Examples

In the following example all of the segments in the user's working directory with the suffix list will be written to tape 12763:

```
! make_pps_tape -vol 12763 [segs **.list]
```

The next example shows the use of -target_description:

```
! make_pps_tape -tds "pps_ 12763 -ct ppf6025" [segs **.list]
```

SECTION 5

PPS I/O MODULE

This section describes the PPS I/O module. The description is similar to the I/O module descriptions found in MPM Subroutines (Order No. AG93) and MPM Peripheral Input/Output (Order No. AX49).

For a general description of the I/O system see "Multics Input/Output System" in section V of MPM Reference Guide (Order No. AG91).

Name: pps_

This I/O module attaches a uniquely named target I/O switch using the tape_ibm_ I/O module, such that a tape suitable for processing on the PPS will be produced.

Entry points in this module are not called directly by users; rather the module is accessed through the I/O system.

Attach Description

The attach description has the following form:

```
pps_ {volids} {-control_args}
```

where:

1. volid
   is the name of a tape volume to be used for output.

2. control_args
   can be chosen from the following:

   -bottom_label XX, -blbl XX
   specifies that the string XX is to be used as a label at the bottom of every page of output until it is modified by the page_labels control order.

   -char_table XX, -ct XX
   specifies the pathname of the PPS character table to be used. The default character table is ppf6023 (see Appendix A). This character table is created using the cv_ppscf command.

   -density n, -den n
   specifies the density of the tape to be produced, where n can be either 1600 or 800. If this control argument is not given, the tape will be produced at 1600 bpi.

   -label XX, -lbl XX
   specifies the string XX as a label at the top and bottom of every page until it is modified by the page_labels control order.

   -modes XX, -mds XX
   specifies the initial mode string XX to be used. (See "Modes Operation" below.)

   -number n, -nb n
   specifies the file number at which this report is to begin. This control argument can be used to add new reports to an existing tape or overwrite specific reports on a tape (see tape_ibm_ in MPM Peripheral Input/Output, Order No. AX49, for details). Since a single attachment can result in multiple output files being placed on the tape, care must be taken when using this control argument.

-retain XX, -ret XX
specifies the disposition of the PPS tape on detachment of this switch. Valid values for XX are "all" and "none". The default is "none".

-top_label XX, -tlbl XX
specifies the string XX as a label to be used at the top of every page of output until it is modified by the page_labels control order.

-volume XX, -vol XX
specifies the tape volume XX as an output tape volume for the target attachment. This control argument must be used when a volume name begins with a hyphen (-).

Open Operation

The only opening mode supported is stream_output. Opening the I/O switch results in the attachment and opening of the target switch.

Close Operation

Closing the I/O switch results in the closing and detachment of the target switch, thus terminating the current PPS output report.

Detach Operation

Detaching the I/O switch releases any temporary segments created during the attachment.

put_chars Operation

The put_chars operation formats the data into a form acceptable to the PPS and writes it on the output tape.

Control Operation

The I/O module supports the following control operations.

channel_stops
end_of_page
get_count
get_error_count
get_position
inside_page
new_report
outside_page
page_labels
paper_info
pps_paper_info
reset
retain_all
retain_none
runout
set_position

In the descriptions below, info_ptr is the information pointer specified in the iox_$control call. Unless specified otherwise, the I/O switch can be either open or closed when processing control orders.

channel_stops
sets the software channel stops. The info_ptr must point to the following structure in which the channel stops are specified (prt_order_info.incl.pl1):

```
dcl channel_stops(256) bit(16) based(info_ptr) unaligned;
```

where:

channel_stops
defines which of the 16 possible channel stops have been set in each of the 256 possible lines on a physical page

end_of_page
positions the output to the end of the current page and writes the bottom page label if specified. The I/O switch must be open.

get_count
returns information about the current page format and print position. The info_ptr must point to the following structure (prt_order_info.incl.pl1):

```
dcl 1 counts          based(info_ptr) aligned,
      2 line          fixed bin,
      2 page_length   fixed bin,
      2 lmarg         fixed bin,
      2 rmarg         fixed bin
      2 line_count    fixed bin,
      2 page_count    fixed bin;
```

where:

line
is the current line number.

page_length
is the current page length.

lmarg
is the current left margin column.

rmarg
is the current right margin column.

line_count
is the number of lines printed since the last reset.

page_count
is the number of pages printed since the last reset.

get_error_count
is simply provided for compatibility with the printer software. The info_ptr must point to the following return value (prt_order_info.incl.pl1):

```
dcl ret_error_count fixed bin based(info_ptr);
```

where ret_error_count is always returned as zero.

get_position
returns the current print position and certain statistics. The info_ptr must point to the following structure (prt_order_info.incl.pl1):

```
dcl 1 position_data          based(info_ptr) aligned,
      2 line_number          fixed bin(35),
      2 page_number          fixed bin(35),
      2 total_lines          fixed bin(35),
      2 total_chars          fixed bin(35),
      2 pad                  (4) fixed bin;
```

where:

line_number
is the current line on the page.

page_number
is the current page number.

total_lines
is the number of lines printed since the last reset order.

total_chars
is the number of characters processed since the last reset order.

pad
is reserved for future use.

inside_page
moves the current print position to the top of the next page. The I/O switch must be open.

new_report
starts a new report on the current PPS tape. This implicitly does a reset control order. The I/O switch must be open.

outside_page
moves the current print position to the top of the next page. The I/O switch must be open.

page_labels
sets the current page labels. If the info_ptr is null the page labels are reset and will not appear on output produced. If the ^endpage mode has been specified an error will result. To set page labels the info_ptr must point to the following structure (prt_order_info.incl.pl1):

```
dcl 1 page_labels          based(info_ptr) aligned,
      2 top_label          char(136),
      2 bottom_label       char(136);
```

where:

top_label
is the label placed at the top of every page.

bottom_label
is the label placed at the bottom of every page.

paper_info
sets the current page size. If an invalid paper size is specified an error will occur (see figure 3-1). The pitch is always set to 12.5 characters per inch by this control order. This order is supported for compatibility with the printer software. The info_ptr must point to the following structure (prt_order_info.incl.pl1):

```
dcl 1 paper_info           based(info_ptr) aligned,
      2 phys_page_length   fixed bin,
      2 phys_line_length   fixed bin,
      2 lines_per_inch     fixed bin;
```

where:

phys_page_length
is the number of lines that would be printed on the page if all lines were printed (as when in ^endpage mode).

phys_line_length
is the number of print positions per line on the page.

lines_per_inch
is the number of printed lines per inch of paper.

pps_paper_info
sets the page size being processed. This control order should be used rather than paper_info whenever possible. The info_ptr must point to the following structure (pps_paper_info.incl.pl1):

```
dcl 1 pps_paper_info          aligned based(info_ptr),
      2 sheet_width           fixed dec(5,1),
      2 sheet_length          fixed dec(5,1),
      2 lines_per_inch        fixed dec(5,1),
      2 chars_per_inch        fixed ded(5,1);
```

where:

sheet_width
is the width in inches of the paper to be used. Valid values are: 5.0, 5.5, 7.5, 8.0, 8.5, and 11.0.

sheet_length
is the length in inches of the paper to be used. Valid values are: 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 7.0, 8.0, 8.5, 10.0, 11.0, 12.0, and 14.0.

lines_per_inch
is the number of lines per inch. Valid values are: 4.0, 6.0, 8.0, and 10.0.

chars_per_inch
is the number of characters per inch (pitch). Valid values are: 10.0 and 12.5.

reset
sets the default mode, and resets the number of lines printed and the number of characters processed. This control order is included to make programs written for printer software compatible with PPS software.

retain_all
causes the retain_all order to be passed to the target I/O switch. This causes the output tape to be retained when the I/O switch is detached. This control order should be used as described in Section 2.

retain_none
resets retain_all. This control order should be used as described in Section 2.

runout
causes all data buffered internally by the pps_ I/O module to be written to the tape.

set_position
sets the line, page and character counts kept by the I/O module. The info_ptr must point to the same structure as for the get_position control order. The line_number value supplied is ignored. This control order is included to make programs written for printer software compatible with PPS software.

Modes Operation

The modes operation is supported whenever the switch is attached. The recognized modes are listed below. These modes are also accepted following the -modes control argument in the pps_ attach description.

default
resets all modes to their default values. This is equivalent to setting the modes endpage, vertsp, edited, fold, ^esc, print, ll<LL>, pl<PL>, in0, stop0 where <LL> is the current physical line length and <PL> is the current physical page length minus the number of lines per inch.

debug, ^debug
causes the I/O module to enter debug mode. This mode is not recommended for users. (Default is ^debug.)

edited, ^edited
specifies that ASCII control characters that do not affect carriage or paper motion are to be escaped (e.g., \177 for DEL). Otherwise, these control characters are ignored. (Default is ^edited.)

endpage, ^endpage
specifies that when the normal printed area of a page is overflowed, printing is continued on the next page. Otherwise, text is printed on every line of the physical page. (Default is endpage.)

esc, ^esc
specifies that the special processing of the ASCII ESC character is to be enabled. (Default is ^esc.)

fold, ^fold
specifies that lines that are longer than the line length are folded to the next line. Otherwise, such lines are truncated to n print positions (where n is the current line length). (Default is fold.)

in*n*
specifies that each line of output is to preceded by n spaces. (Default is in 0.)

ll*n*
specifies the length in print positions of the output line. When an attempt is made to use more than n print positions on a line the remaining text is moved to the next line or discarded depending on the setting of the fold mode. (Default is ll 132.)

pl*n*
specifies the length in lines of the printed page. When an attempt is made to print the n+1th line on the page a form feed character is inserted causing the output to proceed on a new page unless the endpage mode is off. (Default is pl 60.)

pll*n*
specifies the physical line length in characters. This value cannot be smaller than the line length. (See figure 3-1). (Default is pll 132.)

ppl*n*
specifies the physical page length in lines. This value must be chosen to correspond to a valid PPS paper size (see figure 3-2). (Default is ppl 64.)

vertsp, ^vertsp
performs the vertical tab and form feed functions. Otherwise, these characters are simply mapped into newline characters. (Default is vertsp.)

In order to provide some compatibility with the printer software, some additional modes are supported:

1pg
has no effect on the pps_ I/O module.

non_edited, ^non_edited
is the complement of edited (i.e., non_edited = ^edited).

noskip
is the complement of endpage (i.e., noskip = ^endpage).

print, ^print
has no effect on the pps_ I/O module.

single, ^single
is the complement of vertsp (i.e., single = ^vertsp).

stopn
has no effect on the pps_ I/O module.

truncate, ^truncate
is the complement of fold (i.e., truncate = ^fold).

Notes

Because the target I/O switch uses the tape_ibm_ I/O module, the user's process may get queries from tape_ibm_. See Peripheral Input/Output (Order No. AX49) for a description of tape_ibm_.

The volume names specified in the attach description are used in the order they appear in the attach description.

There are two character tables supplied as part of the released PPS software. They are ppf6023 and ppf6025. The ppf6023 character table contains 92 printable ASCII characters with all of the underscored characters mapped to remove the underscores. The ppf6025 character table contains these same 92 printable characters as well as these characters overstruck with the underscore character. For a full description of these character tables see Appendix A.

The I/O module may hold data in buffers between operations. For this reason no operations should be attempted on the target I/O switch while it is being used with the pps_ I/O module.

APPENDIX A

MULTICS PPS CHARACTER SETS

There are two PPS character sets supported by the Multics PPS support package. The character set in figure A-1 is the ASCII character set without underscored characters (ppf6023). The second character set allows a number of overstruck character sequences in addition to the characters in figure A-1. These additional characters are shown in figure A-2.

The figures that follow show the correspondence between the PPS character set and the Multics ASCII character set. The PPS characters are shown in hexadecimal and the ASCII characters or character sequences are shown without backspaces. Figure A-1 for example shows that the ASCII string A is the same as the PPS character represented by C1(16).

HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII
40		f1	1	c2	B	e6	W	92	k
5a	!	f1	1_	c3	C	e6	W_	92	k_
5a	!_	f2	2	c3	C_	e7	X	93	l
7f	"	f2	2_	c4	D	e7	X_	93	l_
7f	"_	f3	3	c4	D_	e8	Y	94	m
7b	#	f3	3_	c5	E	e8	Y_	94	m_
7b	#_	f4	4	c5	E_	e9	Z	95	n
5b	$	f4	4_	c6	F	e9	Z_	95	n_
5b	$_	f5	5	c6	F_	ad	[	96	o
6c	%	f5	5_	c7	G	ad	[_	96	o_
6c	%_	f6	6	c7	G_	e0	\	97	p
50	&	f6	6_	c8	H	e0	_	97	p_
50	&_	f7	7	c8	H_	bd	]	98	q
7d	'	f7	7_	c9	I	bd	]_	98	q_
7d	'_	f8	8	c9	I_	5f	^	99	r
4d	(	f8	8_	d1	J	5f	^_	99	r_
4d	(_	f9	9	d1	J_	3d	^¦	a2	s
5d	)	f9	9_	d2	K	3d	^¦_	a2	s_
5d	)_	7a	:	d2	K_	6d	_	a3	t
5c	*	7a	:_	d3	L	81	a	a3	t_
5c	*_	5e	;	d3	L_	81	a_	a4	u
4e	+	5e	;_	d4	M	82	b	a4	u_
11	+-	4c	<	d4	M_	82	b_	a5	v
11	+-_	8c	<=	d5	N	83	c	a5	v_
4e	+_	8c	<=_	d5	N_	83	c_	a6	w
6b	,	4c	<_	d6	O	4a	c¦	a6	w_
6b	,_	7e	=	d6	O_	4a	c¦_	a7	x
60	-	ae	=>	d7	P	84	d	a7	x_
32	-<	ae	=>_	d7	P_	84	d_	a8	y
32	-<_	7e	=_	d8	Q	85	e	a8	y_
07	-=	be	=¦	d8	Q_	85	e_	a9	z
07	-=_	be	=¦_	d9	R	86	f	a9	z_
60	-_	6e	>	d9	R_	86	f_	8b	{
8f	-¦	6e	>_	e2	S	87	g	8b	{_
8f	-¦_	6f	?	e2	S_	87	g_	4f	¦
4b	.	6f	?_	e3	T	88	h	4f	¦_
4b	._	7c	@	e3	T_	88	h_	9b	}
61	/	7c	@_	e4	U	89	i	9b	}_
61	/_	c1	A	e4	U_	89	i_		
f0	0	c1	A_	e5	V	91	j		
f0	0_	c2	B	e5	V_	91	j_		

Figure A-1. The ppf6023 character table.

HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII
40		f1	1	49	B_	e6	W	92	k
5a	!	36	1_	c3	C	71	W_	cc	k_
25	!_	f2	2	51	C_	e7	X	93	l
7f	"	37	2_	c4	D	72	X_	cd	l_
26	"_	f3	3	52	D_	e8	Y	94	m
7b	#	38	3_	c5	E	73	Y_	cf	m_
27	#_	f4	4	53	E_	e9	Z	95	n
5b	$	39	4_	c6	F	74	Z_	d0	n_
28	$_	f5	5	54	F_	ad	[	96	o
6c	%	3a	5_	c7	G	db	[_	da	o_
29	%_	f6	6	55	G_	e0	\	97	p
50	&	3b	6_	c8	H	e0	_	dc	p_
2a	&_	f7	7	56	H_	bd	]	98	q
7d	'	3c	7_	c9	I	75	]_	dd	q_
2b	'_	f8	8	57	I_	5f	^_	99	r
4d	(	3e	8_	d1	J	22	^	de	r_
2c	(_	f9	9	58	J_	3d	^¦	a2	s
5d	)	3f	9_	d2	K	3d	^¦_	df	s_
2d	)_	7a	:	cb	K_	6d	_	a3	t
5c	*	41	:_	d3	L	81	a	ea	t_
2e	*_	5e	;	59	L_	e1	a_	a4	u
4e	+	42	;_	d4	M	82	b	eb	u_
11	+-	4c	<	62	M_	78	b_	a5	v
04	+-_	8c	<=	d5	N	83	c	ed	v_
24	+_	23	<=_	63	N_	79	c_	a6	w
6b	,	43	<_	d6	O	4a	c¦	ee	w_
33	,_	7e	=	64	O_	17	c¦_	a7	x
60	-	ae	=>	d7	P	84	d	ef	x_
32	-<	16	=>_	65	P_	80	d_	a8	y
07	-=	44	=_	d8	Q	85	e	fa	y_
31	-_	be	=¦	66	Q_	8a	e_	a9	z
8f	-¦	18	=¦_	d9	R	86	f	fb	z_
4b	.	6e	>	67	R_	90	f_	8b	{
30	._	45	>_	e2	S	87	g	fc	{_
61	/	6f	?	68	S_	9a	g_	4f	¦
02	/0	46	?_	e3	T	88	h	fd	¦_
be	/=	7c	@	69	T_	aa	h_	9b	}
18	/=_	47	@_	e4	U	89	i	fe	}_
34	/_	c1	A	6a	U_	ba	i_		
f0	0	48	A_	e5	V	91	j		
35	0_	c2	B	70	V_	ca	j_		

Figure A-2. The ppf6025 character table.

APPENDIX B

MULTICS PPS TAPE FORMAT

The tapes produced by the Multics PPS support package are described in detail in "OS and DOS Support for the Page Processing System" (Order No. AR86). This format in terms of a Multics tape_ibm_ attach description is:

```
tape_ibm_ -vol volid -format fb -rec 133 -block 1596 -mode ascii ...
```

The tape is written using the ASCII mode due to the fact that character translation to the PPS character set is accomplished using the PPS character table.

See MPM Peripheral Input/Output (Order No. AX49) for a complete description of tape_ibm_.

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: LEVEL 68
MULTICS PAGE PROCESSING SYSTEM
UTILITY MANUAL

ORDER NO. CJ97-00

DATED MAY 1980

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

CUT ALONG
FOLD ALONG LINE
FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In the U.K.: Great West Road, Brentford, Middlesex TW8 9DH
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

27531, 5C580, Printed in U.S.A.

CJ97-00