PRIME Computer

SOFTWARE RELEASE DOCUMENT REV.18.2 MRU4304-005

MAN 3251-001

Software Release Document MRU4304-005

Revision 0

This document details the enhancements and changes to Prime software between Rev 18.1 and Rev 18.2

PRIME Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

ACKNOWLEDGEMENTS

We wish to thank the members of the documentation team and also the non-team members who contributed to and reviewed this book.

First Printing July 1981

All correspondence on suggested changes to this document should be directed to:

Rosemary Simpson
Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701

CONTENTS

SECTION 1

INTRODUCTION

NEW BOOKS

DBMS/QUERY Reference Guide (IDR 4607)

This book is a programmer's guide to DBMS/QUERY, which is the Data Base Management System query language and report writer. The query language lets a user retrieve information from a data base without having to write application programs. The report writer lets you format the retrieved information in very special ways. This manual describes every user-level command. Also described are possible additions that you might have to make to a subschema so that complex DBMS data structures can be accessed by QUERY. As this book is a reference guide, it assumes you already know how to use QUERY.

DBMS/QUERY User's Guide (IDR4608)

The DBMS/QUERY User's Guide is a tutorial explantion of the DBMS query language and report writer. After reading this book, you will know how to use all of the fundamental and most of the advanced QUERY commands. The book contains many examples that illustrate how to retrieve information and how this retrieved information can be displayed. This book does not assume that the reader is a programmer.

FED User's Guide (IDR4940)

The Forms Editor, FED, is an interactive program for designing forms that is usable by non-programmers. This book describes those areas of forms management about which a forms designer may wish to confer with a Systems Administrator and/or applications programmer; describes the actual procedures for using FED; and provides a practical example in designing a form and includes screen pictures that illustrate using FED.

SECTION 2

SYSTEM ADMINISTRATOR

THE SYSTEM ADMINISTRATOR'S GUIDE - PDR3109

General Note

Be sure to check the INFO UFD on the master disk for detailed installation information.

Change to the Table of Shared Segments on Page 3-3

The table currently reads:

2100-2177 Reserved for Prime

The table should read:

2100-2167 Reserved for Prime
2170-2177 Reserved for customers

New Option -LOWEND

Add the following to page 7-6:

A new option, -LOWEND, has been added to COPY_DISK to provide faster performance on models other than the 750 and 850. An example of a COPY_DISK request using this option would be:

COPY_DISK -NOVERIFY -LOWEND

Changes and Additions to Printer Support

Add the following to page 8-10:

SPOOL now uses the Electronic Vertical Format Unit (EVFU) in the 300 lpm printer/plotter to define the form length. The forms-length switch is no longer necessary in this device.

The EVFU is enabled by issuing the PROP subcommand, "EVFU ON", when creating or modifying a printer environment. A subsequent command, "PROP name - START" loads the EVFU into the printer as it starts it. If the power to the printer goes off, the EVFU must be reloaded by

stopping and restarting the printer using PRIO.

SPOOL now supports the new "band" printer. This printer uses an EVFU to determine the form length, so its environments must always have EVFU turned on. The DEVICE parameter may be pr0, pr1, pr2, or pr3.

Since the format of the EVFU is different from the one in the 300 lpm printer/plotter, these two devices must be distinguished. A new environment parameter called TYPE has been defined for this purpose. The band printer is TYPE 1 and the printer/plotter is TYPE 0.

850 Halt Handling

Add to Handling Halts Under PRIMOS on page 10-2:

Step 1: On the 850

As the 850 contains two instruction streams, it is first necessary to determine which stream caused the halt. This is done via VCP on the supervisor terminal by typing:

```
A 4/176106 <cr>
```

If the number displayed is '41004, stream #1 is halted and the correct halt address is given in the "Halted at xxxxyyyy" message.

If the number displayed is '120010, stream #2 is halted and its registers must be examined. This is done via VCP on the supervisor terminal by typing:

```
A 4/176400 <cr>
A 7 <cr>
```

The correct halt address is given in the "Slave Halted at xxxxyyyy" message.

F77 Optimization Defaults

Add to Defaults Set by Driver Programs on page 14-1:

Note

Any level of optimization may be set to be the site default by using the distributed program F77DF in the UFD F77>TOOLS. These values are stored in the driver file called F77DATA in the system UFD SYSOVL. This file also stores the error messages returned when compilation errors occur.

FAM II ADDISK Command

Add the following to page 16-2:

The ADDISK command for FAM II systems is different from the ADDISK command used on FAM I systems. The FAM II command has the format:

ADdisk -ON nodename packnamel packname2....packname9

nodename The network node name for a valid FAM II system. The node must be FAM II enabled (see NETCFG).

packname-n The name of the remote partition. FAM II does not use device numbers.

Unlike FAM I, FAM II does not require that the remote system is up, or that the remote disk is started. The FAM II ADDISK command adds the disk and system name to the search list displayed by the STAT DISK command. The status of the disk is checked whenever a user attempts to access the remote disk.

Caution on Use of LWORD

Add the following note to Reverse Channel Protocol on Page 16-6:

The reverse channel protocol is intended as an alternative to the XON/XOFF protocol. Since both protocols use the same stop bit, i.e. bit 4 of the LWORD, they must not be used together in the same LWORD.

Increase in Number of AMLC Boards

Change page 16-7 to reflect the following:

The number of AMLC boards that a single configuration can support has been changed from four to eight.

Changing the Size of the DMQ Buffer

Add the following note to DMQ-Size Option of AMLBUF Command on page 17-3:

To change the size of the DMQ buffer on an assignable line, use the actual line number.

Configuration Directive AMLIBL

Add the following to page 17-4:

Configuration directive AMLIBL defines the size of the DMC input tumble tables at cold start. AMLIBL explicitly sets the size of the input buffers or automatically allocates the maximum size allowed by the available buffer space. The syntax is:

AMLIBL buffer-size

buffer-size is an octal number which represents the number of words allocated to each input buffer. There are two buffers for each AMLC controller and all buffers are made the same size. Except for the special value of zero described below, the number must be greater than '20. The upper bound is variable depending on the number of controllers configured and the amount of space available in the system for buffers. If buffer-size is zero or omitted, the size of the buffers is automatically calculated as the maximum possible. If the AMLIBL directive is not specified, the default buffer size is 60 octal.

If buffer-size is too small, the error message:

BAD AMLIBL PARAMETER (CINIT)

will ge generated during cold start initialization. If buffer-size is too large, the error message:

INPUT BUFFERS TOO LARGE (AMINIT)

is generated at cold start initialization. The user should modify the parameter to be a value within the permissible range as described above.

Default Value Changes

In Filunt Direction on page 17-7 the default values should be 18('22), 127('177), and 2048('4000).

NAMLC Configuration Directive Change

On page 17-9, add to the NAMLC configuration directive:

NAMLC + NTUSR must be less than or equal to octal '177.

Configuration Directive NSLUSR

Add the following to page 17-10:

Each user accessing files on your system from remote systems will require a slave process for the duration of the access. These slave

processes come out of the PRIMOS 128 process pool. Configuration directive NSLUSR defines slave processes. NSLUSR sets the number of slaves configured for a system. The syntax is:

NSLUSR number

number is the number of simultaneous remote file accesses your system wishes to support. If this pool is exhausted when a remote user makes an attach request, the E$NSLA (no NPX slaves available) error code is returned.

Note

NTUSR+NPUSR+NRUSR+NSLUSR must be less than 129.

Change to NTUSR configuration directive

Change page 17-11 to show that the number of terminal users specified can now be a positive octal integer between 2 and 200 instead of 2 and 100.

Assignment of Paging Device Records

Add the following to page 20-6:

At Rev 18.2, paging device records are assigned in blocks of 8 pages at a time, rather than in blocks of 64 pages (or one segment).

Since user segments are often only 25% to 50 % full, a Rev. 18.2 system may be able to handle twice as many segments as a Rev. 17 or Rev. 18.1 system on the same amount of paging space. For this reason, the formula given on page 20-5 for calculating the number of records needed on the paging device (pagdev) now represents the maximum number of records needed. A given system may function well with anywhere between 50% to 100% of this number of records. Therefore, systems on which disk space is tight may want to try reducing the number of records being used for paging.

If the number of paging device records becomes too low, users attempting to access a new page will receive the message "No free paging device records".

Correction to PSD COPY Command

On page A-9, the PSD COPY command used to relocate PRIMOS II during a magtape boot is incorrectly shown with a relocation value of '57541. The correct value is '57477. The command given under point 17 should read:

$C 10000 57477 130000

MAGSAV Error Messages

Add the following to page K-3:

MAGSAV has three new error messages:

MAGSAV UNABLE TO CONTINUE

MAGSAV cannot find a UFD in a pathname. MAGSAV prints the PRIMOS error message plus this message and exits.

RUN OUT OF UNITS
ALTER NUMBER OF UNITS
treename
TYPE 'S' TO CONTINUE

MAGSAV ran out of units to open UFDs on. The file concerned will be lost.

'TOO MANY LEVELS' treename

MAGSAV can only save up to 18 levels on PRIMOS, 13 on PRIMOS II. If more than these levels are attempted, MAGSAV will print this message, ignore that file, return to the previous level, save the files at that level and continue back up the tree in that manner.

Note

A PRIMOS system configured for 16 file units/user can save only 13 levels.

SECTION 3

LANGUAGES

COBOL REFERENCE GUIDE - FDR3056

Correction to UNCOMPRESSED Option

Replace the first paragraph on page 7-4 with:

1. The UNCOMPRESSED clause is optional. When used, it enables a READ base on record length (PRWF$$), rather than compression control characters (I$AD07).

(The reference to subroutines PRWFIL and RDASC is obsolete).

THE FORTRAN 77 REFERENCE GUIDE - IDR4029

Performance Improvements

The overall compilation rate for F77 source programs with a "reasonable" number of comments should exceed 1500 lines per minute in most cases. We have observed compilation rates in excess of 2500 lines per minute on some real benchmarks. Programs that consist mostly of DATA statements will compile much slower than 1500 LPM figure, particularly if large arrays are being initialized.

The most notable case of improved object code for F77 involves references to arrays, where considerably shorter code is generated for references such as

```
X(I+5,J-4)
Z(I+4)
Y(I,1)
```

Major improvements were made in the code generated for many string operations, but these changes will probably not be important for mose F77 programs.

The single precision math routines (SQRT, SIN, COS, etc.) were rewritten for greater accuracy. In most cases, the new versions are slightly faster than the old versions.

Clarification of STOP Statement

On page 3-22, delete the sentence recommending CALL EXIT as a substitute for STOP. CALL EXIT is not equivalent to STOP because STOP closes all file units opened by the program, while CALL EXIT leaves such file units open.

Clarification of Sequential and Direct File Access

In Section 4, files created using the sequential access method are termed SAM files, and files created under the direct access method are termed DAM files. This terminology is incorrect.

The terms "SAM" and "DAM" refer to the two basic file organization PRIMOS uses to implement files. These organizations are not specific to any one language. The terms "sequential access method" and "direct access method" refer to the two types of file offered by the FORTRAN 77 language. Either type of FORTRAN file could be implemented using either PRIMOS file organization. The implementation used is transparent at the programming level, and is subject to change.

For detailed information on PRIMOS SAM and DAM files, see The PRIMOS Subroutines Reference Guide - (PDR3621).

Internal Files

Replace the third paragraph on page 4-4 with the following:

When an internal file is an array, each array element acts as a separate record. If the file is a variable, array element, or substring, the file consists of a single record. After each read or write of an internal file, the file pointer returns to the beginning of the file. To access records other than the first in an array internal file, use the slash (/) edit descriptor.

Optimization Options

In addition to the options listed in Table 7-2, there are three levels of optimization: -OPT1, -OPT2, and -OPT3, where the default is normally -OPT2. The old option -OPTIMIZE is retained and is synonymous with -OPT2. The chosen level is noted in the option header line of the compiler's listing output file. Optimization is turned off, as before, by specifying -NOOPT.

The effect of specifying -OPT2 is elimination of the optimizer logic that moves invarient code out of loops. This is a costly process that was found to consume up to 15 per cent of total compile time on programs with many nested DO loops. It is still available using -OPT3 and is useful in compiling fully debugged programs to be used in frequent production situations. The default optimization (-OPT2)

performs both code pattern replacement and redundancy elimination. The lowest level (-OPT1) does only pattern replacement.

Correction Concerning Statement Functions

Delete the last paragraph on page 8-4 and its header. In F77, statement functions and function subprograms execute with equal efficiency.

FTN - F77 Interface Restriction

To the list of restrictions at the top of page A-3, add:

- An F77 program unit cannot pass a subprogram as an argument to an FTN program unit, nor can an FTN unit pass a subprogram to an F77 unit.

THE PASCAL REFERENCE GUIDE - IDR4303

New Appendix

APPENDIX D

INTERFACING PASCAL TO PLIG AND F77

This appendix provides three examples for interfacing Pascal to incompatible data types of PLI and FORTRAN. The first two examples deal with argument passing of character data types from Pascal to PLI and F77. The last example describes the correct procedure for testing FORTRAN LOGICAL's from Pascal, (i.e. APPLIB).

EXAMPLE ONE

```
{ <char-star.pascal> Passing character( * ) and character*( * ) parameters
                     to PL1G and F77 respectively from Pascal.

  This example program passes any length string from a Pascal program
  to either PL1G or F77. Note, the strings being received in the PL1G
  and F77 programs have star extents for their dimensions. PL1G and
  F77 pass these extents as hidden arguments at the end of the users'
  argument list. Pascal users must pass these extents as actual
  arguments since star extents are not supported in Primes' Pascal.
  The following program does just that.

}
program chrNonVr;
type
  longInteger =
    -32769 .. 32769;

  string5 =
    array[1 .. 5] of char;

  string10 =
    array[1 .. 10] of char;

var
  p1,p2 : string5;
  pR    : string10;

  f1,f2 : string5;
  fR    : string10;

procedure PGconcat(var s1,s2,s3 : char;          { First element of string }
```

```
                    l1,l2,l3 : longInteger   { Extents of the string   }
                ); extern;

procedure F7concat(var s1,s2,s3 : char;        { First element of string }
                    l1,l2,l3 : longInteger   { Extents of the string   }
                ); extern;
begin
  p1 := '12345';
  p2 := '6789.';
  f1 := p1;
  f2 := p2;
  fR := pR;

  PGconcat(p1[1],p2[1], pR[1], 5, 5, 10);

  F7concat(f1[1], f2[1], fR[1], 5, 5, 10);

  if (fR = pR) and
     (fR = '123456789.')
    then
      writeln('PASS...strings(non-varying) in Pascal, PL1G, and F77')
  else
    writeln('FAIL...strings(non-varying) in Pascal, PL1G, and F77');

end.
```

```
/* <PGconcat.pl1g> PL1G procedure to concatenate 2 strings to form
   a third. Accepts as input 2 character nonvarying strings
   a returns a nonvarying string as third argument                    */

PGconcat: procedure(s1,s2,s3);

declare (s1, s2, s3) character(*);

  s3 = s1||s2;  /* concatenate strings and return */
end PGconcat;
```

```
C <F7concat.f77> F77 procedure to concatenate 2 character*(*) strings
C to form a third. Accepts as input 2 character*(*) strings and returns
C a character*(*) string as the third argument.

      subroutine F7concat(s1,s2,s3)
      character*(*)      s1,s2,s3

      s3 = s1//s2                    /* concat the strings */
      return
      end
```

```
/* <char-star.pllg> Using CHARACTER( * ) parameters in pllg.        */
/* The following is the equivalent PL1G program to perform the same
   task as the Pascal program used before for passing star extent
   character strings.                                               */

main: procedure options(main);

declare (f1,f2, p1,p2) character(5);
declare (fR, pR)  character(10);

/* Note; only 3 arguments are required in PL1. The PL1G compiler
   supplies the hidden arguments defining the dimensions of the
   character strings passed.
*/
declare (F7concat,PGconcat) entry(  character(*),
                                    character(*),
                                    character(*) ) external;

  f1 = '12345';
  f2 = '6789.';
  p1 = f1;
  p2 = f2;

  call PGconcat(p1,p2, pR);
  call F7concat(f1,f2, fR);
  if pR = fR
    then
      put skip list('Pass...string(non-varying) PL1G calling');
  else
    put skip list('Fail...string(non-varying) PL1G calling');
end main;
```

EXAMPLE TWO

```
{ <readtty.pascal> Reads text from the user terminal using the external
                   PRIMOS routine - cl$get

  This program provides an example on how define a suitable Pascal
  structure for implementing the character varying datatype found in PL1.
  Since standard Pascal prohibits reading string data from files
  without subscripts, this example will provide an alternate
  solution for reading strings from the user terminal, without
  explicit subscripting.

  The simple object of the program is to read 3 strings from the terminal
  and display them in complete reverse order.
}
program readTTY;
type
  longInteger =
    -32769 .. 32769;
```

```
  char80varying =
    record
      l : integer;
      s : array[1 .. 80] of char;
   end;

var
  cmdline : char80varying;
  table   : array[1 .. 3] of char80varying;
  i,j     : integer;
  status  : integer;

{ This procedure is documented in the PRIMOS SUBROUTINES REFERENCE GUIDE -
  PDR3621, page 5-2.                                                      }
procedure cl$get(var cmdline : char80varying;  { Command line input buffer }
                     lenBytes: integer;        { Length of cmdline in bytes}
                 var status  : integer);       { Return error code status  }
          extern;                              { External PRIMOS procedure }

begin
  { Loop to input the text entered from the user terminal using the
     PRIMOS routine defined above (cl$get).
  }
  for i := 1 to 3 do
  begin
    write(i:1,'> ');
    cl$get(cmdline, 80, status);
    if status <> 0
      then
        writeln('Bad status code returned, status =',status);
    table[i] := cmdline;  { save the command line }
  end;

  { Display the lines just typed in reverse order. }
  writeln;

  for i := 3 downto 1 do
  begin
    write(i:1,'< ');
    for j := table[i].l downto 1 do
      write(table[i].s[j]);
    writeln;
  end;
```

EXAMPLE THREE

```
 { <applib.pascal> Example of how to interface a Pascal program to the
                   standard APPLIB routines.

  FORTRAN logical's ARE incompatible with Pascal boolean data types.
```

Therefore, interfacing to the applications library from Pascal can be a problem. The following "program" shows the easiest way to to determine True and False when calling FORTRAN subroutines with logical's.

Note: This program assumes that the type of logical returned is a LOGICAL*2, and only occupies 2-bytes of memory. If the FORTRAN subroutine called was compiled with default options using F77, the type of result returned by the FORTRAN subroutine would have to be a Long Integer.

```
}
program main;
const


type
  longInteger =
    -32769 .. 32769;
  string8 =
    array[1 .. 8] of char;

  string16 =
    array[1 .. 16] of char;

var
  msg : string16;
  date: string16;
  time: string8;

procedure date$a(var s : string16); extern;
procedure time$a(var s : string8); extern;

function ysno$a(var s : char;      {Pass by ref, first loc of the msg  }
                    l : integer;   {Pass by value, length of msg       }
                    k : integer    {Pass by value, default keys        }
              ):integer; extern;  {Returns fortran logical as integer }

begin
  date$a(date);  {Read today's date    }
  time$a(time);  {get the current time}
  writeln;
  writeln('Today`s date: ',date);
  writeln('Time:         ',time);
  writeln;

  msg := 'Yes | No        ';
  if ord( True ) = ysno$a(msg[1],8, a$ndef)
    then
      writeln('Ok!')
  else
    writeln('Absolutely NO!');
end.
```

The PASCAL compiler supports the following option:

-UPCASE

-UPCASE maps lowercase characters into uppercase characters in identifiers.

THE PL/I SUBSET G REFERENCE GUIDE - IDR4031

New Options

Add the following options to page 14-7:

-ERRTTY - List errors on the terminal
-NOERRTTY - Do not list errors on the terminal
-ERRLIST - Produce an errors-only listing file
-FRN - Round the floating accumulator before storing a float bin (23)

Maximum Sizes Add the following information to the MAXIMUM SIZES list on page 11-1 of The PL/I Subset G Reference Guide:

- The maximum number of %INCLUDE nesting level positions is 32.
- The maximum number of items in a structure is 1024.

Change in SKIP format

A value of 0 can now be given for n in the SKIP format. In the SKIP format discussion on page 9-16, the third paragraph that begins "If n is omitted," should now read:

"If n is omitted, a value of one is supplied. The value of n must not be negative. If 0 is given the current line will be overwritten."

SECTION 4

UTILITIES

EDITOR

The FNAME Command

Insert the following information after the discussion on the WHERE command in Section 3 of The New User's Guide to EDITOR and RUNOFF - (FDR3104).

The FNAME command prints out the name of the file you are working on during an editing session. For example, if you are editing a file called junk, the FNAME command will print the filename junk wherever you issue the command in the file:

```
Print 2
The name of this file is called junk.
.sk2
fname
JUNK
```

If you're working on a new file and haven't yet specified a filename, the FNAME command will elicit this message:

```
File name not specified
```

MODE NOSEMI and MODE SEMI

The following information affects Section 9 of The New User's Guide to Editor and Runoff - (FDR3104) and Section 4 of The Prime User's Guide - (PDR4130).

NOSEMI mode eliminates the need to use the CHANGE command to insert semicolons in a file, using either input mode or edit mode.

In input mode, MODE NOSEMI causes the EDITOR to treat a semicolon as a regular print character instead of a line terminator.

In edit mode MODE NOSEMI causes the APPEND, INSERT, and RETYPE commands to treat semicolons as literals. For example:

```
OK, ed sheep
EDIT
mode nosemi
next
BA, BA BLACK SHEEP
append; HAVE YOU ANY WOOL
BA, BA BLACK SHEEP; HAVE YOU ANY WOOL
```

The semicolon still terminates (or separates) EDITOR commands that do not insert new text into the file. For example:

```
OK, ed sheep
EDIT
mode nosemi
t;n
BA, BA BLACK SHEEP
```

MODE SEMI causes the EDITOR to recognize the semicolon as the line terminator when in input mode. The default semicolon mode is MODE SEMI.

ASSIGN

Add the following to page 2-6 of The Prime User's Guide:

Mounting Tapes: When you assign a tape drive, you may also specify a particular tape to be placed on the drive by using the -TPID control argument. Sometimes it is necessary to have one tape removed from the tape drive and another tape placed or mounted on the tape drive. This is done by using the -MOUNT control argument. To specify -MOUNT, the tape drive must already be assigned. For example, suppose ADLEY, user number six, assigns logical drive seven:

```
OK, ASSIGN MTO -ALIAS MT7 -800 BPI -TPID GRADES
OK,
```

Now, suppose ADLEY reads or writes the tape with id GRADES, and then wants another tape mounted. ADLEY types:

```
OK, ASSIGN -ALIAS MT7 -MOUNT -TPID EXAMS
OK,
```

The operator receives a message at the system terminal indicating that user ADLEY wants tape EXAMS mounted. The operator responds to ADLEY's request by using the REPLY command. ADLEY then receives a message indicating whether or not the mount operation was successful. The mount operation might be unsuccessful, for example, if the operator could not find the requested tape.

LOAD (CORRECTION)

Delete the phrase "(to which LOAD will add the SAVE suffix)" from page 6-8 of The Prime User's Guide.

LOAD appends the .SAVE suffix only to default filenames which it creates itself. It does not append the .SAVE suffix to user specified filenames.

PHANTOM LOGOUT (CORRECTION)

Delete the information on phantom logout notification and its associated subroutines, LO$CN and LO$R on page 9-11 of The Prime User's Guide and on pages 78-6 through 78-10 of PIU78 - Subroutines. This functionality is not provided at Rev 18.

CRMPC (CORRECTION)

Replace the section on page 12-2 of The Prime User's Guide beginning "Source deck header control cards..." and ending "...by the command START at the terminal." with the following:

The CRMPC command translates the card images into an ASCII file. Cards are expected on 029 representation. Control cards may be inserted into the card deck to instruct the card reader as follows:

Columns 1 and 2 of deck control card	Instruction
$6	Placed before a deck of cards in 026 format. Instructs the card reader to interpret 026 cards as if they were in 029 format.
$9	Instructs the card reader to resume reading in 029 format.
$E	Placed last in the deck and signals the end of the deck. Control returns to PRIMOS and the file is closed.

If the card deck is exhausted but contains no $E card at the end (or if the reader is halted by the user), control returns to PRIMOS but the file is not closed. If more cards are to be read into the file, the reader should be reloaded; reading is resumed by the command START at the terminal.

SUBROUTINES

Change the type declaraction of LOGICAL on page 11-48 of The PRIMOS Subroutines Reference Guide (PDR3621) to read LOGICAL*2.

This change makes A$KEYS, the $INSERT file used with APPLIB and VAPPLB to define parameter keys, compatible with F77.

Formerly, A$KEYS declared its logical variables as LOGICAL. This created an incompatibility with F77 because LOGICAL defaults to LOGICAL*4 in F77, while APPLIB and VAPPLB expect LOGICAL*2 arguments.

The logical variables in A$KEYS are now explicitly declared as LOGICAL*2, eliminating the incompatibility with F77.

Replace T$AMLC description on page 20-16:

ASYNCHRONOUS CONTROLLERS

The following describes the raw data mover for assigned AMLC lines. Refer to the System Administrator's Guide for the AMLC command, and how to assign AMLC lines.

T$AMLC

T$AMLC is a direct entrance call. It performs raw data movement, provides status information about assigned amlc lines, and transfers characters to and from the caller's buffer to a desired assigned line's buffer. The caller must own the desired line, i.e., the corresponding LBT entry must contain the caller's user number.

CALL T$AMLC (line, user_buf_addr, char_count, key, stat_vec,char_pos_arg, errcode)

line Desired amlc line number.

user_buf_addr Address (pointer) to the caller's buffer.

char_count Desired number of characters to move. No maximum limit is enforced.

key

1 input char_count characters.

2 input char_count characters or until .NL.
stat_vec(1) = actual number of characters read.

3 output char_count characters.
Maximum = char_count. This key assures the caller that char_count characters will be output. For example, an error is not returned if the line's input or output buffer is smaller than char_count. T$AMLC will output blocks of data from the caller's buffer into the available room in the line's output buffer until char_count is exhausted. A one second wait is issued between output chunks to allow time for the line's output buffer to clear. In most cases, the entire char_count should be output at once.

4 stat_vec(1) = number of characters in input buffer.
stat_vec(2) = state of carrier.

5 return status of output buffer.
stat_vec(1) = 1 if room for char_count in output buffer.
stat_vec(1) = 0 if not enough room for char_count.
stat_vec(2) = state of carrier.

6 input all available characters in the input buffer.
Maximum = char_count. This key will place all the available characters in the line's input buffer into the caller's buffer.
stat_vec(1) = number of characters actually input.

7 return additional output buffer status (refer to key 5).
stat_vec(1) = amount of character space remaining in the
output buffer.

8 flush input buffer.

9 flush output buffer.

10 flush both output and input buffers.

11 output characters to available room in output.
This key will output as many characters as possible into the line's output buffer. A wait will not be done to exhaust char_count.
stat_vec(1) = number of chars that were not successfully output, i.e. stat_vec(1) = 0: means all characters were output.

stat_vec Two word status vector used by certain keys.

char_pos_arg The caller may wish to indicate a starting position within the buffer addressed by user_buf_addr. Char_pos_arg applies for both input and output keys. This is an optional argument. If omitted, the default is to start with the first character. Note: if

char_pos_arg is used, the first character position should be indicated by one (there is no character at position zero). Also, char_pos_arg is NOT updated within T$AMLC.

errcode Optional argument to return error status. If errcode is present, error messages will not be printed at the caller's physical terminal.

SOURCE LEVEL DEBUGGER REFERENCE MANUAL - IDR4033

PASCAL Additions

Change page 1-3 language support description to include PASCAL.

Add to page 3-7:

PASCAL: A PASCAL program block is a main program, procedure, or function, and is identified by the name given in the PROGRAM, PROCEDURE, or FUNCTION statement. Nested procedures are qualified as they are in PL/1.

Change page 3-11 statement label description to include PASCAL - statement-label may be a PASCAL statement number preceded by a dollar sign.

Change page 5-3 : command description to include PASCAL as a valid language-name value.

Replace builtin function list on page 4-13 by the following listing which includes PASCAL functions.

ABS	COMPLEX	HBOUND	NINT
ACOS	CONJG	HIGH	NOT
ADD	COPY	IABS	NULL
ADDR	COS	IDIM	ODD
ADDREL	COSD	IDINT	OFFSET
AFTER	COSH	IDNINT	ONCODE
AIMAG	CSIN	IFIX	OR
AINT	CSQRT	IMAG	ORD
ALOG	DABS	INDEX	POINTER
ALOG10	DATAN	INT	PRED
AMAX0	DATAN2	INTL	PTR
AMAX1	DATE	INTS	RANK
AMIN0	DBLE	IRND	REAL
AMIN1	DCOS	ISIGN	REVERSE
AMOD	DDIM	LBOUND	RND
AND	DEC	LEN	RS
ANINT	DECAT	LENGTH	RT
ASIN	DECIMAL	LGE	SEARCH
ATAN	DEXP	LGT	SHFT
ATAN2	DIM	LLE	SIGN
ATAND	DIMENSION	LLT	SIN
ATANH	DINT	LOC	SIND
BEFORE	DIVIDE	LOG	SINH
BIN	DLOG	LOG10	SNGL
BINARY	DLOG10	LOG2	SQRT
BIT	DMAX1	LOW	SUBSTR
BOOL	DMIN1	LS	SUBTRACT
BYTE	DMOD	LT	SUCC
CABS	DNINT	MAX	TAN
CCOS	DPROD	MAX0	TAND
CEXP	DSIGN	MAX1	TANH
CHR	DSIN	MIN	TIME
CLOG	DSQRT	MIN0	TRANSLATE
CMPLX	EXP	MIN1	VERIFY
CMPX	FIXED	MOD	XOR
COLLATE	FLOAT	MULTIPLY	

Change page 5-17 LANGUAGE command description to include PASCAL.

UNWATCH Command Changes

On page 5-38 add the option -ALL to the UNWATCH command list. If UNWATCH -ALL is specified, all variables are removed from the watch list.

VTRACE Command Changes

On page 5-39 replace the command line and first paragraph of the VTRACE command with the following:

The VTRACE command allows the user to enable full or entry/exit value tracing, or to disable value tracing at any time, while retaining the variables in the watch list.

The format of the command (abbreviated VT) is:

VTRACE { FULL | ENTRY_EXIT | OFF }

The OFF argument disables value tracing without disturbing the contents of the watch list. The FULL and ENTRY_EXIT arguments control the frequency of comparisons of saved values to current values. If "full" value tracing is enabled, comparisons occur following execution of each statement. If "entry_exit" tracing is enabled, values are compared only at entry to and exit from each routine in debug mode.

HELP Command Changes

On page 68-3 of PTU68 - DBG - replace the description of HELP with the following:

The HELP command may be used to find the name of the most recent and up-to-date DBG documentation. It may also be used to display a list of all debugger commands, the syntax of any DBG command, a list of all syntax symbols used in DBG command syntax descriptions, or the definition of a command syntax symbol.

The format of the HELP command is:

HELP [-LIST | -SYM_LIST | command-name | syntax-symbol]

Where:

command-name is the name or abbreviation of a DBG command.

syntax-symbol is a symbol enclosed in angle brackets used in a command syntax description, for example, "<breakpoint-identifier>".

If the HELP command is issued with no arguments, the syntax of the HELP command and the name of the most recent DBG documentation are printed.

```
> HELP
For help, refer to IDR4033 Source Level Debugger Reference Manual.

HELP -LIST              prints a list of all DBG commands
HELP -SYM_LIST          prints a list of all syntax symbols
HELP <command-name>     prints the syntax of <command-name>
HELP <syntax-symbol>    prints the definition of <syntax symbol>
```

If -LIST is specified, a list of DBG commands is printed. Command abbreviations are indicated by capital letters.

```
> HELP -LIST
        !                   *                   :
        ActionList          ARGumentS           BReaKpoint
        CALL                CLeaR               CLeaRAll
        CmdLine             Continue            ENVironment
        EnvList             ETrace              GOTO
        HELP                IF                  IN
        INFO                LANGuage            LET
        LIST                LiSTAll             LoadState
        MACro               MacroList           MAIN
        OUT                 PAuse               PMode
        PSYMbol             Quit                ReSTart
        ReSUbmit            SaveState           SEGmentS
        SouRCe              STATUS              Step
        StepIn              STrace              SYMbol
        TraceBack           TRAcepoint          TYPE
        UnWatch             UNWIND              vPSD
        VTrace              WAtch               WatchList
        WHere
```

If the HELP command is followed by -SYM_LIST, a list of all DBG syntax symbols used in DBG command syntax descriptions is printed.

```
> HELP -SYM_LIST
ACTION-LIST             ACTIVATION-NUMBER       ALTERNATE-ENTRY-NAME
ALTERNATE-ENTRY-ID      ARGUMENT                ARGUMENT-LIST
BOUND-PAIR              BREAKPOINT-IDENTIFIER   BREAKPOINT-TYPE
CHARACTER-VALUE         COMMAND-LIST            COMMAND-NAME
EXPRESSION              FILE-NAME               INSERT-LINE
LANGUAGE-NAME           LINE-OFFSET             LOWER-BOUND
MACRO-NAME              NEW-MACRO-NAME          OLD-MACRO-NAME
PRIMOS-COMMAND-LINE     PRINT-MODE              PROCEDURE-NAME
PROGRAM-BLOCK-NAME      SEGMENT-NUMBER          SOURCE-COMMAND
SOURCE-LINE             STATEMENT-IDENTIFIER    STATEMENT-LABEL
STATEMENT-OFFSET        STEP-COMMAND            SYMBOL-NAME
SYNTAX-SYMBOL           UPPER-BOUND             VALUE
VARIABLE                VARIABLE-LIST           VARIABLE-NAME
WORD-NUMBER
```

If HELP is followed by a <command-name>, the syntax of that command is displayed.

```
> HELP INFO
INFO  {<program-block-name> \  |
       <alternate-entry-id>  |
       <statement-identifier>}
```

If a <syntax-symbol> follows HELP, the definition of that symbol is printed.

```
> HELP <STATEMENT-IDENTIFIER>
<STATEMENT-IDENTIFIER>:
<source-line> |
<source-line>+<statement-offset> |
<source-line> (<insert-line>) |
<source-line> (<insert-line>+<statement-offset>) |
<statement-label> |
<statement-label>+<line-offset> |
<statement-label>+<line-offset>+<statement-offset>
```

MACRO Command Changes

On page 68-5 of PTU68 - DBG - replace the description of MACRO with the following:

Using the MACRO command, the user can define a macro name which may be used in place of one or more debugger commands.

The format of the command (abbreviated MAC) is:

```
MACRO  macro-name  {command-list | -DELETE | -EDIT}  |
       -CHANGE_NAME old-macro-name new-macro-name  |
       -ON  |  -OFF
```

Where:

macro-name and new-macro-name are user-supplied names.

command-list is one or more debugger commands enclosed in square brackets ("[]").

old-macro-name is the name of an existing user macro.

A macro is defined with the MACRO command by entering a macro-name followed by a command-list. The macro is then entered into a debugger table known as the "macro list". Thereafter, whenever macro-name is entered at DBG command level, the debugger commands in command-list will be executed, with supplied parameters, if any.

A macro-name is removed from the macro list by supplying the argument

-DELETE.

The -EDIT argument indicates that the command-list associated with this macro-name is to be edited using the DBG command line edit facility.

The -CHANGE_NAME argument may be used to change the name of a user macro from old-macro-name to new-macro-name.

The MACRO -OFF command inhibits expansion of user macros without disturbing the macro list.

The MACRO -ON command reenables expansion of user macros.

SAVESTATE Command Changes

On page 68-5 of PTU68 - DBG - replace the description of SAVESTATE with the following:

The SAVESTATE command may be used to save certain debugging information for restoration at a subsequent invocation of DBG.

The format of the SAVESTATE command (abbreviated SS) is:

SAVESTATE file-name [-MACROS] [-BREAKPOINTS] [-TRACEPOINTS]

Where:

file-name is a tree name describing a file.

When this command is issued with no control arguments, the following are saved: all tracepoints and breakpoints with associated action lists and attributes, and all user-defined macros. DBG commands to restore this information are written to file-name in user-readable form.

If any of the arguments -MACROS, -BREAKPOINTS, or -TRACEPOINTS are specified, then only those items requested will be saved.

SECTION 5

DATA MANAGEMENT SYSTEMS

GENERAL DATA BASE INFORMATION

General Note to DBMS Users

The way in which DBMS alters set currencies for optional member record occurrences now conforms with CODASYL specification. If a record is defined as optional manual, mandatory manual, or optional automatic, and does not participate in any set occurrence of the set, set currency will not be affected by that record. Previously, when a record became current, all set currencies that the record occurrence could belong to were updated. This was done whether or not the record actually participated as a member of the set.

With revision 17.6 the only currencies updated are those in which the record occurrence actually participates.

Application programs conforming to present documentation are not affected. However, if a record does not currently participate as a member in a set (set X), a FIND OWNER in X of current of set X, will result in a "stale" owner record — the owner of the most recently found member record that does participate as a member in set X.

Converting Pre-Rev. 18 Data Bases

Since the size of the before-image header has been increased, pre-Rev. 18 schemas will NOT work correctly unless the following actions are taken:

1. BEFORE installing Rev. 18, run before-image recovery on all schemas and do SAVEs for each schema.

2. Install Rev. 18 DBMS.

3. In DBUTL, make the schema the current one (e.g., SC MYSCHEMA) and then type the DBUTL REV18 command.

4. Rev. 18 will then do this one-time per schema conversion.

Any SAVEs of pre-18 schemas will not work correctly on a Rev. 18 system. Consequently, if you restore a pre-Rev. 18 save, invoke DBUTL and type the REV18 command to convert the restored data base. It is recommended that you do a new save of the schema so that you have a replacement for the pre-18 schema.

If you are not sure if a schema has been converted to Rev. 18 format, invoke DBUTL, DUMP BEFORE and examine GENBIT. If GENBIT is 8 words long, the schema has been converted; if it is 4 words long, it has not.

If you attempt to run a pre-18 schema, you will get the message:

```
THIS SCHEMA NEEDS DBUTL REV18 CONVERSION
CONTACT DBA FOR INSTRUCTIONS
```

WARNING

There is no mechanism for preventing inappropriate conversion (i.e., conversion of schemas either created with an 18 or higher version of DBMS or schemas that have already been converted). The DBA must exercise caution in this area.

THE DBMS ADMINISTRATOR'S GUIDE - PDR3276

DBACP RECOVER SCHEMA Command

If you type the DBACP RECOVER SCHEMA command and run-units are accessing the data base, DBACP asks the question:

```
DO YOU WISH TO ABORT THE RUN-UNITS?
```

If you answer YES, DBACP waits for you to abort all run-units accessing the data base. This is done by logging these run-units out from another terminal. After you log these run-units out, run CLUP so that the data base is in a correct state. When you run CLUP, do not forget to use the:

```
CLUP -U user-number
```

form of the CLUP command.

Note

Do not CLUP a run-unit while it is running or a loss of data integrity will result.

After DBACP determines that no one is accessing the data base, it performs the RECOVER operation.

If you type NO, DBACP asks:

```
DO YOU WISH TO WAIT FOR THE RUN-UNITS TO FINISH?
```

If you answer YES, DBACP will wait until it determines that no one is accessing the data base. It then performs the recovery operation.

If you again type NO, DBACP returns to its command level.

SCHED

When adding new files (e.g., areas, sets, etc) or modifying existing files, SCHED now creates DAM rather than SAM files. The following procedure identifies and corrects existing data bases:

1. Attach to all PDBMS UFD's that contain production data bases.

2. Use the file utility (FUTIL) to find if any DBMS files are SAM files by first using FROM SDnnnn, where nnnn is the SCHEMA number, zero filled. Then use L T.

3. If there are any SAM data base files:

   - Use DBACP to SAVE the data base.

   - Use DBACP to RESTORE the data base.

All segment directory entries for the data base will now be DAM files.

PTU73 - REV 18 DBMS

DBUTL

Two commands, VERIFY and DBK, have been added.

VERIFY: For each b-tree in the current set, VERIFY establishes that every leaf node data base key is in the data base. The format of this command is:

```
VERIFY [ integer ]
```

If the optional integer argument is used, a checkpoint message will be displayed after the specified number of owner directories (i.e., set occurrences) have been processed. This numerical argument is a decimal number.

The messages that can be printed are:

1 COULD NOT POSITION TO ROOT OF B-TREE

2 COULD NOT POSITION TO LEFT-MOST LEAF NODE

3 INFINITE LOOP IN LEAF NODE 'RIGHT' POINTERS

4 COULD NOT POSITION TO THE NEXT (RIGHT) LEAF NODE

6 DBK NOT IN DATABASE

7 DBK MARKED AS DELETED IN DATABASE

This command also serves as a compliment to the new DMLCP command which is described below.

DBK: The format of this DBUTL command is:

```
      { n1 n2 n3 }
DBK   {          }
      { dbk      }
```

If you speify a data base key (dbk) in its logical format (which is area-id, record-id, occurrence-number), DBUTL displays the internal 48 bit representation of the dbk as three decimal numbers. If you type the internal representation of the dbk (the three decimal number option), DBUTIL displays the dbk in its logical (unpacked) format. This comand is used to determine the logical format of a dbk given the packed format displayed in DMLCP traces (e.g., FTRACE).

DBMS FORTRAN REFERENCE GUIDE - PDR3045 AND DBMS COBOL REFERENCE GUIDE - PDR3046

DMLCP

The command line option -VERIFY has been added to DMLCP. This option lets an application program check record integrity in data base areas as well as check the consistency of CALC and set files.

When using this option, the application program should contain a series of FIND (or FETCH) NEXT RECORD OF AREA statements to check each key of interest. DMLCP locates the record using all keys (i.e., CALC, sort, and search keys) defined for the record type and verifies that the record is contained in the non-sorted set occurrences for which it is currently a member. In addition, for each set occurrence that the record owns, DMLCP checks that the record's dbk is contained in the set file.

The one inconsistency that DMLCP -VERIFY does not detect is the case in which a dbk is contained in a set list, but it is either not contained in the area file specified by the dbk or is marked as deleted in the area file. For this case, use when the DBUTL VERIFY command to detect inconsistencies of this type. (The DMLCP and DBUTL verifies are complimentary and do not overlap in functionality.)

When an inconsistency is detected, DMLCP writes a binary description of the error to a file opened on unit 45. Consequently, a file must be opened on this unit number before the run-unit is invoked. For example:

```
OPEN BUG.FILE 45 3
SEG #PROGRAM -VERIFY
CLOSE 45
```

The program DBMSLB>VFYPRT.SAVE displays the contents of BUG.FILE in a formatted fashion. VFYPRT will print a menu of options that can be used to display or analyze the bug file. For example, VFYPRT can be used to print the definition of a particular bug number discovered via the -VERIFY.

Creation of a DML Application Program

Once a schema has been written and compiled and a subschema has been written and compiled, and the data base files have been allocated with DBACP, the user can write application programs for the data base in either COBOL or FORTRAN. The sequence used to transform the source code into executable code is as follows:

1. Preprocess the source code with the host language preprocessor (CDML OR FDML).

2. Compile the output of the preprocessor (D_xxxxx) with the host language compiler (COBOL or FTN).

3. Link the binary output of the compiler to the DML command processor with the segmented loader SEG.

Sample job streams to do these operations with either a COBOL or FTN program may be found in UFD DBMSLB called C_CDML, C_CLOAD, C_FDML, and C_FLOAD. These template jobs are used in conjunction with EXEC.

DML ERROR MESSAGE CLARIFICATIONS

The following two minor exception codes have been added.

39 Implicit access of a set not included in the subschema

50 Retrieving next of set for set wherein current member removed

DML SYNTAX

In some cases, there has been some ambiguity concerning COBOL and FORTRAN DML syntax. The following pages describe all syntax.

The DML command descriptions use the following conventions:

UPPERCASE	Represent keywords that you must type as shown.
UPPERCASE	Represent words that increase readability. You may include or omit these words at your discretion.
lowercase	Represent variables for which you must supply a value.
{ }	Enclose a group of options from which you must choose only one.
[ ]	Enclose optional choices; you can include the item or items in brackets, or you can omit it, as you choose.
|| ||	Enclose a group of options from which you can choose as many as you like; however, you must choose at least one. In all cases, when you select more than one, options may be separated by commas if you so choose.
...	Indicates that repetition is allowed. The portion of the format that can be repeated is delimited by the [or { that logically matches the] or } to the immediate left of the ... .
-list	Means you can repeat the last entity. For example, the generic term "item-list" means: item-1 [,item-2] ... "-list" is only used with lowercase variables (that is, ones for which you supply a value). In all cases, a list could consist of only one entity.

In many cases, the singulars and plurals of keywords are parsed identically; that is, if you see a plural, you can type the singular or vice versa. In particular, AREA and AREAS, RECORD and RECORDS, SET and SETS, and ERROR and ERRORS have identical meaning. In most cases, only one form is indicated.

COBOL Syntax

ABORT TRANSACTION

```
ABORT TRANSACTION identifier .
```

CLEAR ERROR

```
CLEAR ERROR .
```

CLEAR SUPPRESS

```
CLEAR [ SUPPRESS { ALL                          } ].
                 { RECORD                       }
                 { AREA                         }
                 { SETS set-list                }
```

CLOSE AREAS

```
CLOSE { ALL AREA[S]      } .
      { AREAS area-list  }
```

DELETE

```
DELETE [ MANDATORY ] .
       [ SELECTIVE ]
       [ ALL       ]
```

END TRANSACTION

```
END TRANSACTION identifier .
```

EXIT DBMS

```
{ EXIT  } DBMS .
{ ABORT }
```

FIND/FETCH

Format 1

```
{FETCH}
{FIND }  USING db-key .
```

Format 2

```
{FETCH}                                                  {RECORD record}
{FIND }  [ {OWNER  }  IN set-1 OF ]  CURRENT OF  {SET set-2   }  .
           {MEMBER }                             {AREA area   }
                                                 {RUN-UNIT    }
```

Format 3

```
{FETCH}  {NEXT   }
{FIND }  {PRIOR  }
         {FIRST  }  RECORD [ record ] OF  {SET set  }  .
         {LAST   }                        {AREA area}
         {integer}
         {identifier}
```

Format 4

```
{FETCH}
{FIND }  [ NEXT DUPLICATE WITHIN ] RECORD record .
```

Format 5

```
{FETCH}
{FIND }  record VIA [ CURRENT OF ] SET set [ USING item-list ] .
```

Format 6

```
{FETCH}
{FIND }  NEXT DUPLICATE WITHIN SET set USING item-list .
```

GET

```
GET [ item-list ] .
```

IF

Format 1

```
IF set SET [ NOT ] EMPTY { cobol-procedure-1 }
                         { NEXT              }
        [ [;] ELSE cobol-procedure-2 ] .
```

Format 2

```
IF RECORD [ NOT ] { MEMBER } OF { set SET } { cobol-procedure-1 }
                  { OWNER  }    { ANY SET }  { NEXT           }
        [ [;] ELSE cobol-procedure-2 ] .
```

INSERT

```
INSERT INTO { SETS set-list } .
            { ALL SET[S]   }
```

INVOKE DBMS

```
INVOKE DBMS .
```

MODIFY

```
MODIFY [ item-list ] .
```

MOVE

Format 1

```
MOVE CURRENCY STATUS FOR { RUN-UNIT    } TO identifier .
                         { RECORD record }
                         { AREA area     }
                         { SET set       }
```

Format 2

```
MOVE { RECORD-NAME } FOR { RUN-UNIT          } TO identifier-2 .
     { AREA-NAME   }     { RECORD record     }
                         { AREA area         }
                         { SET set           }
                         { identifier-1      }
```

ON ERROR

Format 1

```
[;] ON ALL ERRORS GO TO cobol-procedure .
```

Format 2

```
[;] { ON ERROR integer-list GO TO cobol-procedure-1 } ...
    [ ON OTHER ERRORS GO TO cobol-procedure-2 ] .
```

OPEN AREAS

Format 1

```
OPEN ALL AREAS
    [ USAGE[-]MODE IS [ EXCLUSIVE ] { RETRIEVAL } ] .
                      [ PROTECTED ] { UPDATE    }
```

Format 2

```
OPEN AREAS area-list-1
    [ USAGE[-]MODE IS [ EXCLUSIVE ] { RETRIEVAL } ]
                      [ PROTECTED ] { UPDATE    }
    [ ; AREAS area-list-2
        [ USAGE[-]MODE IS [ EXCLUSIVE ] { RETRIEVAL } ] ] ... .
                          [ PROTECTED ] { UPDATE    }
```

PRIVACY KEY

Format 1

```
PRIVACY KEY FOR [ || [ EXCLUSIVE ] { RETRIEVAL } || ]
                    [ PROTECTED ] { UPDATE    }

           OF { AREAS area-list } IS { identifer } .
              { ALL AREAS       }    { literal   }
```

Format 2

```
PRIVACY KEY FOR [ { REST                       } ]
                [ { || STORE                || } ]
                [ { || GET                  || } ]
                [ { || MODIFY               || } ]
                [ { || INSERT               || } ]
                [ { || REMOVE               || } ]
                [ { || DELETE MANDATORY     || } ]
                [ { || DELETE SELECTIVE     || } ]
                [ { || DELETE ALL           || } ]
                [ { || FIND                 || } ]
           OF { RECORDS record-list } IS { identifer } .
              { ALL RECORDS         }    { literal   }
```

Format 3

```
PRIVACY KEY FOR [ { REST          } ]
                [ { || STORE   || } ]
                [ { || GET     || } ]
                [ { || MODIFY  || } ]

           OF DATA-ITEMS item-list IS { literal    } .
                                      { identifier }
```

Format 4

```
PRIVACY KEY FOR [ { REST | INSERT | REMOVE | FIND } ]
                OF { SETS set-list | ALL SETS } IS { identifer | literal } .
```

REMOVE

```
REMOVE FROM { SETS set-list | ALL SETS } .
```

START TRANSACTION

```
START TRANSACTION identifier [ , UPDATE | , RETRIEVAL ] .
```

STORE RECORD

```
STORE record .
```

SUBSCHEMA

```
SUBSCHEMA subschema OF SCHEMA schema .
```

SUPPRESS

```
SUPPRESS { ALL | RECORD | AREAS | SETS set-list } .
```

FORTRAN DML Syntax

ABORT TRANSACTION

```
ABORT TRANSACTION identifier .
```

CLEAR ERROR

```
CLEAR ERROR .
```

CLEAR SUPPRESS

```
CLEAR [ SUPPRESS { [ ALL          ] } ] .
                 { [ RECORD       ] }
                 { [ AREAS        ] }
                 { [ SETS set-list ] }
```

CLOSE AREAS

```
CLOSE { ALL AREA[S]      } .
      { AREAS area-list  }
```

DBMS SUBPROGRAM

```
DBMS SUBPROGRAM .
```

DELETE

```
DELETE [ MANDATORY ] .
       [ SELECTIVE ]
       [ ALL       ]
```

END TRANSACTION

```
END TRANSACTION identifier .
```

EXIT DBMS

```
{EXIT }
{ABORT} DBMS .
```

FIND/FETCH

Format 1

```
{FETCH} USING db-key .
{FIND }
```

Format 2

```
{FETCH}  [ {OWNER  } IN set-1 OF ] CURRENT OF {RECORD record}
{FIND }    {MEMBER }                           {SET set-2    }
                                               {AREA area    }
                                               {RUN-UNIT     } .
```

Format 3

```
{FETCH}  {NEXT      }
{FIND }  {PRIOR     }
         {FIRST     }
         {LAST      } RECORD [ record ] OF {SET set }
         {integer   }                    {AREA area}
         {identifier} .
```

Format 4

```
{FETCH}
{FIND } [ NEXT DUPLICATE WITHIN ] RECORD record .
```

Format 5

```
{FETCH}
{FIND } record VIA [ CURRENT OF ] SET set [ USING item-list ] .
```

Format 6

```
{FETCH}
{FIND }  NEXT DUPLICATE WITHIN SET set USING item-list .
```

GET

```
GET [ item-list ] .
```

IF

Format 1

```
IF set SET [ NOT ] EMPTY {fortran-label-1}
                         {NEXT           }
      [ [;] ELSE fortran-label-2 ] .
```

Format 2

```
IF RECORD [ NOT ] {MEMBER} OF {set SET} {fortran-label-1}
                  {OWNER }    {ANY SET} {NEXT          }
      [ [;] ELSE fortran-label-2 ] .
```

INSERT

```
INSERT INTO {SETS set-list}
            {ALL SET[S]   } .
```

INVOKE DBMS

```
INVOKE DBMS .
```

MODIFY

```
MODIFY [ item-list ] .
```

MOVE

Format 1

```
MOVE CURRENCY STATUS FOR { RUN-UNIT       } TO identifier .
                         { RECORD record  }
                         { AREA area      }
                         { SET set        }
```

Format 2

```
MOVE { RECORD-NAME } FOR { RUN-UNIT       } TO identifier-2 .
     { AREA-NAME   }     { RECORD record  }
                         { AREA area      }
                         { SET set        }
                         { identifier-1   }
```

ON ERROR

Format 1

```
[;] ON ALL ERRORS GO TO cobol-procedure .
```

Format 2

```
[;] { ON ERROR integer-list GO TO cobol-procedure-1 } ...
    [ ON OTHER ERRORS GO TO cobol-procedure-2 ] .
```

OPEN AREAS

Format 1

```
OPEN ALL AREAS
    [ USAGE[-]MODE IS [ EXCLUSIVE ] { RETRIEVAL } ] .
                      [ PROTECTED ] { UPDATE    }
```

Format 2

```
OPEN AREAS area-list-1
    [ USAGE[-]MODE IS [ EXCLUSIVE ] { RETRIEVAL } ]
                      [ PROTECTED ] { UPDATE    }

    [ ; AREAS area-list-2
        [ USAGE[-]MODE IS [ EXCLUSIVE ] { RETRIEVAL } ] ] ... .
                          [ PROTECTED ] { UPDATE    }
```

PRIVACY KEY

Format 1

```
PRIVACY KEY FOR [ || [ EXCLUSIVE ] { RETRIEVAL } || ]
                    [ PROTECTED ] { UPDATE    }
                OF { AREAS area-list } IS { identifer } .
                   { ALL AREAS      }     { literal   }
```

Format 2

```
PRIVACY KEY FOR [ { REST                  } ]
                [ {  STORE                } ]
                [ {  GET                  } ]
                [ {  MODIFY               } ]
                [ {  INSERT               } ]
                [ {  REMOVE               } ]
                [ {  DELETE MANDATORY     } ]
                [ {  DELETE SELECTIVE     } ]
                [ {  DELETE ALL           } ]
                [ {  FIND                 } ]
            OF { RECORDS record-list } IS { identifer } .
               { ALL RECORDS        }     { literal   }
```

Format 3

```
PRIVACY KEY FOR [ { REST | STORE | GET | MODIFY } ]
    OF DATA-ITEMS item-list IS { literal | identifier } .
```

Format 4

```
PRIVACY KEY FOR [ { REST | INSERT | REMOVE | FIND } ]
    OF { SETS set-list | ALL SETS } IS { identifer | literal } .
```

REMOVE

```
REMOVE FROM { SETS set-list | ALL SETS } .
```

START TRANSACTION

```
START TRANSACTION identifier [ , UPDATE | , RETRIEVAL ] .
```

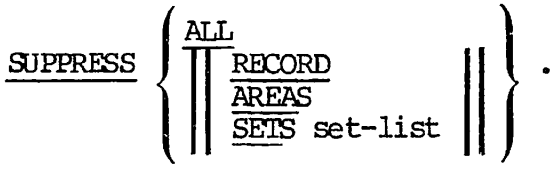
STORE RECORD

```
STORE record .
```

SUBSCHEMA

```
SUBSCHEMA subschema OF SCHEMA schema .
```

SUPPRESS



THE MIDAS USER'S GUIDE - IDR4558

The following is a compilation of error corrections and addenda. The changes are listed on a section by section basis.

Section 2 Changes

On page 2-3, the first sentence should read:

- Secondary key types and sizes — these are optional and should be used when you want more than one search key for the file. Secondary search keys do not have to be part of the data record except in COBOL.

Page 2-10 states that a read-write lock setting for n readers and n writers is "equivalent to the PRIMOS RWLOCK setting of 3". This is incorrect. Instead, what was meant was that you would use the FUTIL "3" setting to set the file read-write lock to n readers and n writers.

On page 2-11, the word "protected" should read "projected."

Section 3 Changes

There are several new messages output by KBUILD during the process of building a file. Most of them should not concern the user as they are simply informative and do not indicate difficulties. The diagnostic messages make more sense if you understand how KBUILD goes about building a MIDAS file. Briefly, KBUILD builds a MIDAS file in one or more stages, called "passes." On each pass, one or more index subfiles are built or are deferred for building during a subsequent pass. The KBUILD message:

FIRST BUILD/DEFER PASS COMPLETE

simply indicates that KBUILD has finished building the data subfile, (if the primary index needs to be built) and has built one or more indexes while possibly deferring the building of others. KBUILD defers the building of an index only if the index to be built is empty and the user-provided input is unsorted. During the first pass KBUILD puts the unsorted input entries for each such index into a temporary "defer" file. After the first pass is complete, KBUILD sorts the individual defer files and builds the index subfiles from the now sorted data.

Each time an index is sorted, KBUILD prints out a message indicating which index it is going to sort. After the sort is complete, the message:

SORT COMPLETE

is printed. Similarly, KBUILD announces the building of each index. For example:

BUILDING INDEX 0

After this index is built, the message:

INDEX 0 BUILT

is displayed. When KBUILD has finished building the data subfile and all the index subfiles that needed to be built, it displays the message:

KBUILD COMPLETE.

and control returns to PRIMOS (unless you are running KBUILD out of a command file).

On page 3-32, the word "under" should be eliminated from the first sentence in paragraph one.

Section 5 Changes

On page 5-3, the word OUTPUT on the second line from the bottom should be INPUT instead.

Section 6 Changes

On page 6-18, the description of the buffer argument incorrectly reads, "The size of the data record buffer" when it should be simply "The data record buffer."

Also on page 6-18, it should be noted that the bufsiz argument is always supplied in words.

On pages 6-21 and 6-31, the argument file-no was mistakenly omitted from the argument explanation list. Please insert the following in between the index and bufsiz arguments on these pages:

file-no Set this to 0: obsolete.

On page 6-32, the description for FL$FST incorrectly reads, "Tells FIND$ to the position to first entry..." when it should read, "Tells FIND$ to position to the first entry...". The sentence should now read correctly.

On page 6-49, the argument listed as key value should be simply key.

On pages 6-52 and 6-58, the key argument was inadvertently omitted from the calling sequences, although it does appear in the argument explanation lists on pages 6-53 and 6-59. The key argument should be inserted between the buffer and array arguments in these calling sequences.

Section 7 Changes

On page 7-18, delete the section START and Locked Records.

On page 7-19, the phrase KEY IS should be inserted after PHONE-FILE in the example at the top of the page.

Section 9 Changes

On page 9-15, the second and third comment lines from the top of the page should be omitted. This program contains no on-units for the KEY condition (using an invalid key in a file access operation). This is why the error messages shown in the output occur. To help clarify this, the following sentence should be added before Reminders, on page 9-15:

> "The error conditions are raised because there is no on-unit to trap KEY errors: see ERROR HANDLING below."

Section 12 Changes

The information contained in the Note on page 12-11 is very important. We're mentioning it here to make sure that the note is not overlooked.

Section 13 Changes

On page 13-5, there should be a step 4 indicating that the read/write file locks on all MIDAS files should be changed to 2 (n readers and 1 writer) when disabling concurrent process handling.

Section 14 Changes

On page 14-2, the argument namlen does not have to be declared as INT*2, as implied in the book. This is the default however.

On page 14-11, the statement under Why Use Offline Routines? claims that offline routines are faster because they are not shared. This is only part of the actual story. Offline routines are not meant to be shared and therefore are not concerned with multi-user access to a file. Therefore they don't write out index blocks after each index entry is added to a file, as ADD1$ does. (Online routines must always write index blocks out to the file after each operation on the block so

that the file will not be damaged by concurrent access and so each user will have a consistent view of the file while accessing it.) By not writing out the index blocks each time, a considerable amount of I/O overhead is saved, making offline routines faster than their online counterparts. In addition, the offline routines bypass the concurrent process handling method which normally single-threads MIDAS use for online routines. Thus only one person can have access to a MIDAS file at a time when that file is being processed by an offline routine.

On page 14-34 under ERROPN: the funit argument should always be specified as a variable and not a constant, because it returns a value of 0 if the call to ERROPN is unsuccessful.

Section 15 Changes

Page 15-8 discusses the EXTEND option of CREATK as a method of making the index subfile longer. This method is preferable to the double-length index method which can also be used to lengthen an index subfile. Users are urged to use the EXTEND option whenever they need to enlarge an index subfile. Please note this carefully, as it was not explicitly stated anywhere in the book.

Index Changes

On page X-9, some of the listings for KX$RFC and KX$TIM incorrectly refer to Section 4 instead of Section 14.

General Changes

The file-no argument, which appears in all of the FORTRAN subroutine calling sequences, can be set to any value the user desires. Currently the documentation infers that file-no should be set to 0. This argument is ignored completely by MIDAS and is being preserved only for compatibility.

Special Characters

Replace page 1-12 by the following:

Several characters reserved for special use by POWER cannot be included in descriptor names or filenames. These special characters are:

. period
, comma
(left parenthesis
) right parenthesis
/ slash
* asterisk
+ plus
- minus
= equal
< greater than
> less than
' single quote
# pound (number) sign
$ dollar sign
! exclamation mark
% percent sign
ESCape key (ESC on most terminals)

Characters that do NOT appear in the above list, like colon (:) and semicolon (;) are assumed to be legal characters in both filenames and descriptor names.

Exceptions: A few of the special characters listed above CAN be used in POWER filenames. These characters are:

period (.)
minus(-)
pound sign (#)
dollar sign ($)

For example, the names $TOTAL and ACCT# are legal POWER filenames, but they would not be legal descriptor names.

CREATE Options

Add the following to page 4-22:

A MIDAS search descriptor may not be added or have its data length or type changed with the CREATE options. If you wish to make this kind of change (or wish to change a display descriptor to a search descriptor), you should perform the following steps:

1. Dump all data to a file.
2. Destroy the original file.

3. Exit POWER and tredel the data file.
4. Enter POWER and recreate the file as desired.
5. Batch add the data.

If any descriptor names are being changed for the new file, these name changes should be made on the old file (using the CHANGE DESCRIPTOR option). Otherwise, date in those descriptors will not be added.

All structural changes to a file of any type that contain data should be made in the same manner.

Redefining a MIDAS Index

Replace the description on page 4-24 with:

The above example introduces a prompt displayed only when a MIDAS file is changed. A "NO" response should be given to the prompt "DOES DESCRIPTOR REDEFINE A MIDAS INDEX?", unless:

- a search descriptor is deleted
- a search descriptor's record position is changed.

Change to LIST Command Syntax

On page 6-5 the LIST command syntax is given as:

LIST (file filename)

It should read:

LIST (formname -specification)

Default Display Formats

Change page 7-10 to read:

Unless a user has created a heading for a file, all descriptors will be displayed using POWER's default formats, as listed below:

Data Type	Default Display
NUM1 (R*8)	-ZZZZZZZ.##
NUM2 (R*4)	-ZZZZZZZ.##
NUM3 (I*2)	-ZZZZZ
NUM4 (I*4)	-ZZZZZZZZZZ
NUM5 (Decimal)	-ZZZZZZZ.##
NUM6 (COMP-3)	-ZZZZZZZ.##

If these default displays are now what is wanted, you should create a heading (using HEADING CREATE).

PT65

Add the following to page 7-25:

POWER now functions with the Prime PT65 terminal. Users must initiate the downloader (by QA TERM) before entering POWER. As the PT65 operates in low intensity mode, the low intensity visual attribute in a POWER screen will not function. Entering this characteristic will have no effect on the screen. Also, fields must not begin before column 3 in a PT65 screen.

HEADING CREATE

There is no reference in the manual to the query for desired numeric format if the descriptor is numeric. You should know that unless you want the default display format for that data type, you must enter a format. A carriage-return response will result in the default displays.

MIDAS CHANGES FOR 18.2

The major changes to MIDAS for 18.2 involve organization of the product on the master disk and small naming modifications to comply with new Prime software standards. These changes impact Section 13 of The MIDAS User's Guide under INITIALIZING MIDAS. Most of this information impacts the Administrator only, and should not affect most users. Any changes involved in installation of MIDAS can also be found in the INFO file on the master disk.

Changes to MIDAS Organization

Pages 13-5 and 13-6 of the MIDAS manual list the names of all the sub-UFD's and command files that appeared in the MIDAS UFD up until Rev 18.2. As of Rev 18.2, the following changes have been made:

- The MIDAS ufd no longer contains any source files or utility-building command files. These files, listed on page 13-6, were formerly contained in MIDAS>SOURCE. SOURCE now exists in a new ufd called MIDASSRC. The MIDAS ufd now contains insert files, share and install command files, plus the run files that exist in the sub-ufd's CMDNC0, LIB, SYSCOM, and SYSTEM.

- The command files to install and share MIDAS remain the same, but build files have been rewritten in CPL and are now stored in MIDASSRC. The master file to build all of MIDAS is called MIDAS.BUILD.CPL and calls all the individual CPL files to build the MIDAS utilities. These CPL files are named CREATK.BUILD.CPL, KBUILD.BUILD.CPL, and so forth.

- All source files in the the sub-ufd MIDASSRC>SOURCE conform to the new Prime suffix standard. All FORTRAN source files have the .FTN suffix, all FORTRAN insert files have the suffix .INS.FTN, and so on.

- For compatibility, all insert files in MIDAS and MIDAS>SYSCOM exist with and without the proper suffix, for example, PARM.K and PARM.K.INS.FTN. Use the versions with the new suffixes in future applications.

- The MIDAS utilities CREATK, IMIDAS, KBUILD, KIDDEL and MPACK do not accept command line arguments or options (and they never have). They now report the message:

  NO COMMAND LINE ARGUMENTS POSSIBLE

  when a user attempts to supply an argument upon invocation.

- MCLUP now recognizes when a user attempts to use the "-USER ##" option more than once per invocation.

For FORTRAN Users Only

When deleting entries from a MIDAS file through the FORTRAN interface, flags should be used with care when making calls to NEXT$ before and after a call to DELET$. Some users have encountered concurrency errors (error 13) when they first call NEXT$ with FLAGS = FL$USE + FL$RET, then incorrectly call DELET$ with FLAGS = FL$USE, and finally call NEXT$ with FLAGS = FL$USE + FL$RET. The error 13 happens because the array being used on the call to NEXT$ points to a deleted entry. The correct way to perform this sequence is to:

1. Call NEXT$ with FLAGS = FL$USE + FL$RET

2. Call DELET$ with FLAGS = FL$USE + FL$RET

3. Call NEXT$ with FLAGS = FL$USE + FL$RET

This pattern should be followed except when deleting the first entry in an index. In such a case, FL$RET should NOT be set on the call to DELET$. Instead, FLAGS should be set to FL$FST on the second call to NEXT$ to avoid problems.

SECTION 6

FORMS

THE FORMS PROGRAMMER'S GUIDE - PDR3040

Changes to FORMS

The FDL source line limit has been extended to 90 characters (from 72). This implies that any text that used to follow column 72 must now be commented, i.e. the text must be preceded by /*.

FDL now supports the correct standard suffices, .FORM for source, .FBIN for binary and .LIST for listing.

FAP handles the standard binary suffix of .FBIN (see the FDL compiler).

TCB handling has been extended to allow (if required) machine unique TCB lists, instead of the standard system unique TCB. This can be achieved by copying TCB.BN from FORMS* and placing it in machine unique TCB* ufds. The original under FORMS* would then no longer be needed.

The maximum user number allowed in the TCB list has been extended from 64 to 128.

SECTION 7

PRIMENET

CHANGES TO NETCFG

The changes shown below apply to Section 5, Network Configuration, of The Primenet Guide, IDR3710.

Additional Public Data Network

PRIMENET now supports the TYMNET public data network. The name TYMNET is a correct response to the NETCFG dialog question, "Your national Public Data Network (PDN)?" (Users who do not have PDN support do not see this NETCFG question.)

NETCFG Changes for FAM II

The dialogue for the Network Configuration program (NETCFG) has been modified to allow users to specify whether a node will use FAM I or FAM II remote disk access. A new question, "Enable FAM II?" has been added to the dialogue.

The following is a description of the modified dialogue:

Enable FAM II?

YES — Enable FAM II on this node. The remote node must also enable FAM II to you. NETCFG procedes to ask the Naming Sphere and Ring 0 Password questions below.

NO — FAM II will not be enabled. NETCFG asks the Enable FAM I and Permit Remote FAM questions, below.

Is this node in your naming sphere?

YES — Node is in the naming sphere.

NO — Node is not in the naming sphere.

Note

This question is provided for future expansion. Answer "YES" to this question.

Ring0-Ring0 password?

A 32 character password may be specified for each FAM II node. Passwords may be changed with the NETCFG -PASSWORD option. This password is used to prevent unauthorized nodes

or software from making FAM II requests. Whenever FAM II services are initiated, the source node and password are checked against the configured FAM II nodes and passwords. If the match fails, FAM II service will be denied.

A <cr> means no password.

Enable FAM I?

YES Enable FAM I on the node.

NO Neither FAM I nor FAM II is enabled for this node.

Permit remote FAM to start disks?

YES Allow the remote system administrator to add your disks.

NO Do not allow the remote system administrator to add your disks.

NETCFG -PASSWORD Option

The Network Configuration program (NETCFG) now supports a password modification option. This allows users to create, delete or change passwords for all nodes in a specific network without reconfiguration. The following is an example of the NETCFG -PASSWORD dialogue:

```
OK, netcfg -password
Review old network configuration? yes

Rev    18.2 network configuration file

Ring Net
       Name          Addr          Ring ID   FAM INFO   RLOG
      -------    --------------    -------  ----------  ----
*ME*  QWERTY                            1
      ASDF                              2   II/SAME-NS   Yes
                        Ring0-Ring0 password: GGJJG

Do you want to update node passwords? yes

Node name (CR if done)? asdf
New Ring0-Ring0 password? mnbv

Node name (CR if done)?

Review new network configuration? yes

Rev    18.2 network configuration file
```

```
Ring Net
      Name          Addr          Ring ID   FAM INFO   RLOG
      --------      ------------  -------   ---------  ----
*ME*  QWERTY                          1
      ASDF                            2     II/SAME-NS  Yes
                    Ring0-Ring0 password: MNBV

OK,
```

SECTION 8

DPTX

INTRODUCTION

Significant changes have been made to the DPTX product at Rev 18.2. The following is a listing of these changes and the sections of The DPTX Guide, IDR4035 that they impact:

- Documentation correction to OWLDSC command — Section 2
- Changes to TCF command — Section 2
- Changes to TSF command — Section 2
- Changes to the Block Device Interface (BDI) — Section 3
- New DPTCFG, DPTX Startup, Warm Start, Multiline TM — Section 4

CORRECTION TO OWLDSC COMMAND

On page 2-9, no stars should appear arround the "USAGE:" message. The message itself should read:

USAGE: OWLDSC [FAST] [NOLOCK] [REPORT]

CHANGES TO DPTX/TCF

The following changes have been made to DPTX/TCF (pass-through) for Rev 18.2:

- Changes in the TCF command line
- Modifications for the use of the TEST REQUEST key
- New connect time error message
- New post-invocation error message
- Multiple lines and general polling

Use of PA2 and TEST REQUEST Keys

Program Access key 2 (PA2) is now available for the QUIT function. In the command line:

TCF HOST hname -TERMINAL tname [-QUIT 'q-string'] [-PA m] [-PF n] [-TR]

the value of m may be 1, 2, or 3.

The use of the TEST REQUEST key may be inhibited (not recognized) for some terminals. This key is disabled in the DPTX configuration when general polling is enabled for the control unit to which the terminal is attached. It is enabled if general polling is not used.

New Connect Time Error Message

The following message is new for Rev 18.2:

```
Correct syntax is:
TCF -HOST <host_name> -TERMINAL <term_name> [-PF <pf_key#>]
                      [-Q[UIT] 'q_string']
    <term_name> can be '*' (use this terminal).
    PF (program function key #'s 1-12 can be changed to
       PA (program access key #'s 1 thru 3)
    or TR (test request -- no number needed)
OK,
```

The user failed to follow the correct command format. Reenter the command line. TCF returns the invoking terminal to PRIMOS command level after sending this message.

New Post-Invocation Messages

The following message is new to DPTX/TCF as of Rev 18.2:

```
HOST not responding, returning to PRIMOS.
OK,
```

The following message is sent to the invoking terminal when the block mode virtual link to the terminal is broken and all attempts at recovery fail. The invoking terminal is returned to PRIMOS command level:

```
Unable to retain terminal link, data possibly lost,
      returning to PRIMOS.
OK,
```

Multiple Lines and General Polling

The traffic manager (TM) of DPTX/TCF now supports multiple synchronous lines and general polling, as described below for DPTX/TSF.

CHANGES TO DPTX/TFS

The following changes have been made to DPTX/TSF for Rev 18.2:

- Multiple Synchronous Lines
- General Polling

Multiple Synchronous Lines

DPTX/TSF can now be configured for more than one synchronous line. Prior to Rev 18.2, only one line was supported. As before, a given synchronous line may have more than one control unit address configured, that is, DPTX/TSF supports multidropping of IBM control units.

General Polling

DPTX/TSF has been enhanced to do general polling of attached IBM control units, in addition to specific polling. This has been implemented as a new option under SP3270. See the information on DPTCFG on how to configure general polling.

General polling is usually preferred over specific polling (which is the default configuration), since general polling queries an entire control unit which responds if any of its devices has text or certain kinds of status to send. In contrast, specific polling queries only a specific device. Use of general polling reduces the number of nonproductive "nothing to send" polls.

CHANGES TO BLOCK DEVICE INTERFACE (BDI) CALLS

The following changes have been made to the Block Device Interface for Rev 18.2:

- New BDKEYS insert files
- Changes to BD$ATT, BD$SET, BD$INP and BD$INF
- A new call, BD$LST

BDKEYS Insert Files

Insert files for PL/I and assembler programs using the Block Device Interface (BDI) have been added to the UFD SYSCOM. SYSCOM now contains these insert files:

```
BD$KEYS.INS.FTN     (for FORTRAN programs)
BD$KEYS.INS.PL1     (for PL/I programs)
BD$KEYS.INS.PMA     (for PMA programs)
```

BD$ATT: Attaching a device

BD$ATT has a new code parameter, E$PHNA:

E$PHNA: The Protocol Handler for this device is not active; call ignored.

BD$SET: Set attributes for a Device

The following changes have been made to the BD$SET call: a new set of keys and the inclusion of a new CODE parameter, E$IWST.

Keys: The current status of input/output operations performed by the Protocol Handler on behalf of the BDI user. The current status of these operations, as they relate to these keys, may be determined by a call to BD$INF.

K$RSMI: (Default attribute) Enable input from the device. This causes the Protocol Handler to resume normal input. Device input is automatically enabled by a "BD$ATT" call. (To preserve compatibility with earlier versions of the BDI, this key is functionally equivalent to BD$ENBD, and they are interpreted the same.)

K$SPDI: Disable device input. The Protocol Handler will stop soliciting and/or accepting input for this device. The user may continue to dequeue any remaining device input (BD$INP). (Again, to preserve compatibility, this key is functionally equivalent to K$DSBD in earlier versions of the BDI.)

K$ABTI: Abort device input. The Protocol Handler will cease soliciting and/or accepting input from the device (as for K$SPDI above). Additionally, at a time of its choosing, the Protocol Handler will drain the device input queue to the user. Having completed this, it will reset the abort condition and leave the device input inhibited. While the abort is still in effect, the user may not reenable input by means of the K$RSMI key above, however, once it has been reset by the Protocol Handler, he may do so if he so chooses. The user should use this feature with care, as an abort flushes output statuses (TTOK, TTF: see BD$INP types and subtypes) as well as device inputs. In such an instance, the user might well be unaware of which messages were sent to the device, and which were rejected (if any). Note that an input abort is entirely independent of the output abort condition described below.

K$RSMO: (Default attribute) Enable output to the device. The Protocol Handler will resume sending output from the user output queue to the device. This option is automatically enabled by the BD$ATT call.

K$SPDO: Disable output to device. The Protocol Handler will cease

dequeuing messages from the user output queue to output. The user may continue to queue (BD$OUT) output to the device (until the queue becomes full), but no output will be sent to the device until output is reenabled by a BD$SET call of K$RSMO.

K$ABTO: Abort output. The Protocol Handler will cease dequeuing messages from the user output queue to send to the device (as for K$SPDO, above). At a time of the Protocol Handler's choosing, it will drain the output queue form the user of all messages (no indication of how many messages are drained is given), reset the abort condition, and leave output to the device inhibited. While the abort condition exists, the user may not queue any additional outputs (BD$OUT) nor may he reenable output to the device. When the the abort condition has been cleared, as can be determined by a call to BD$INF, the user may then reenable output and/or queue output as he sees fit. Note that the output abort condition is entirely independent from the input abort condition described above (except as affected by the Protocol Handler).

K$INWT: Enable the "input waiting" condition for BD$OUT. If the user attempts to output (BD$OUT) to a device which has input waiting for him, the attempt is rejected and an error code of E$INWT is returned. This condition may also be enabled at configuration time, in which case the user may not enable or disable it (a return code of E$INWT is given, and the call is ignored).

K$IWOF: (Default attribute) Disable the "input waiting" condition on BD$OUT. The user may queue output to the device Protocol Handler regardless of the presence of any device input awaiting his acceptance. This condition may be enabled at device configuration time, in which case the user may not disable it (an error code of E$IWST is returned).

K$PA2P: Disable DPTX Support Traffic Manager (STM) trapping of the PA2 key. DPTX-TSF uses the PA2 key as a "quit" or "break" key, which, if depressed while "quits" are enabled, causes an immediate return of the user's process to PRIMOS command level. If the PA2 trap is disabled, the PA2 key will not be recognized as a "quit" key by the STM, and will be passed through to the user program while that device is in block mode. K$PA2P is invalid for any other protocol.

K$PA2Q: Enable the aforementioned trap of the PA2 key by the DPTX Support Traffic Manager. K$PA2Q is invalid for any other protocol.

K$RAWD: Send the data in data (and the fields in outarr) to the Protocol Handler for transmission to this device without checking the validity of that data, or performing any translations on it. This key will be rejected (at the BDI) for devices whose Protocol Handlers do not accept such data.

Code: The code description is:

E$IWST: "Input waiting" condition enabled in configuration; call ignored.

BD$INP: Input from a Device.

The following changes have been made to the BD$INP call:

- A new key value, K$WATT
- A new code error value, E$BPRH
- A new argument, period

CALL BD$INP (device, key, data, length, inparr, code, period)

K$WATT: Wait until some input is available and return with it, or until the specified time limit (period — tenths of a second) is exceeded and return with an error code of E$NINP if no input is available.

E$BPRH: Bad protocol handler defined in configuration.

period A time period specified by the user to indicate the length of time to wait for input before returning to the user if no input is at hand when the call is made. This value is expressed in tenths of a second, and is only examined when a key of K$WATT is specified. In this manner, BD$INP remains compatible with earlier versions which did not have this capability or parameter. If the value is less than or equal to 0, it is equivalent to making the same call with a key of K$WAIT. If the time period runs out and no input is received an error code of E$NINP is returned. If input is available, the data is returned to the user, without any indication of how long the wait was.

BD$INF: Information about a device

The following changes have been made to BD$INF:

- A new key value, K$INFS
- Redefinition of the infarr array

K$INFS: BD$INF will return static device data in words one through 4 of infarr. The contents of data are left untouched.

infarr: A ten-word array to be filled with information about the device:

For key = K$INFN or K$INFD:

word 1: Protocol type of this device.

0: undefined

1: DPTX/TSF (3270 Terminal Support)

2: DPTX/DSC (3270 Emulation port)

word 2: Device status and configuration bits.

The configuration bits are set at configuration time (the DPTX command at the system console).

bit 1-10: Reserved.

bit 11: Physical status: 1 = up, 0 = down.

bit 12: "input waiting" condition for BD$OUT enabled by configuration. If true (= 1), the user must retrieve (BD$INP) any input data from the device queues before sending any output (BD$OUT).

bit 13: Input allowed from device. If true, the protocol handler will solicit (or accept, as appropriate for the protocol) input, provided the user has not inhibited it (see below and BD$SET).

bit 14: Output allowed to device. If true, the protocol handler will send user output (queued through BD$OUT) to the device, provided the user has not inhibited it (see below and BD$SET).

bit 15: Block mode allowed. The user may attach this device in block mode. (This is somewhat redundant as the user must be able to attach the device if he is to do the BD$INF call.)

bit 16: Prime standard terminal. This device may be used as a command device (i.e. the user may converse with PRIMOS through it).

word 3: Maximum input and output buffer length defined for this device. This is the maximum value of length, which is acceptable on a BD$OUT call with key of K$XMTD. It is also the largest size in characters, which any input from this device can attain. The user program should provide an input buffer of at least this length on BD$INP calls to ensure enough

room for the maximum sized message.

word 4: Character codes used by the device (bits 1-8), and the user (bits 9-16).

0: ASCII

1: EBCDIC

The user can control the value of the latter (user character code) through the use of the BD$SET call (keys of K$ASCD — ASCII and K$EBCD — EBCDIC).

word 5: User alterable device enable/disable status. The user may indirectly control the input and output data streams from the Protocol Handler to the device and to the user by calls to the BD$SET routine. This word describes the current status of those requests.

bit 1-10: Reserved.

bit 11: PA2 key DPTX-TSF Support Traffic Manager (STM) trap disabled. This indicator is not valid for non-DPTX-TSF command terminal devices. When false (= 0), the STM will trap the PA2 key on input from such a device, signal a "quit" condition, and not pass the key to the user program. When true (= 1), the STM will pass the PA2 key to the user program exactly as it would any other device input.

bit 12: Input waiting condition enabled. If this bit is turned on (= 1) (or bit 12 of word 2 above), the user must retrieve device input, if any, before sending output (BD$OUT) to the device. This setting only has effect if the "input waiting" condition was not enabled in the configuration. This bit is altered by calls to BD$SET with keys of K$IWON (condition enabled) and K$IWOF (condition disabled). It is disabled when the user attaches (BD$ATT) the device.

bit 13: Abort Transmit in progress. If true (= 1), the user has requested (BD$SET) a transmit abort at some earlier time, and it is still in progress. Input from the device is inhibited. At some later time, the Protocol Handler will drain the device input queues and reset the abort condition. The user may not re-enable

(BD$SET) device output until the abort has been completed. Further, the user may not queue (BD$OUT) output for the device while the abort is in progress. However, when the abort condition has been cleared, the user may queue output, even though output is still disabled, at least until the output queue to the Protocol Handler fills up.

bit 14: Receive Abort in progress. If true (= 1) the user previously requested (BD$SET) an input abort for this device, and it is still in progress. At some later time the Protocol Handler will drain the device input queue and reset the abort condition. Device input is disabled, and the user may not re-enable (BD$SET) it until the abort condition has been cleared. However, the user may dequeue (BD$INP) remaining device input from the queues while the abort condition is in progress.

bit 15: Output suspended. If true (= 1), the user has previously requested (BD$SET) either a transmit abort or suspension. While output is suspended, the Protocol Handler will not send any output to the device. However, provided a transmit abort is not in effect, the user may queue output (BD$OUT) for the device until the device output queue fills up. The user may reset (BD$SET) this condition provided a transmit abort (see above and BD$SET) is not in progress.

bit 16: Input suspended. If true (= 1), the Protocol Handler is not accepting input from the device. The user disabled device input by a previous call to BD$SET. The user may retrieve (BD$INP) any remaining input from the device input queue while the input is disabled. The user may re-enable (BD$SET) device input provided an input abort is not in progress.

word 6: (reserved)

words 7-10: Protocol-specific status information, if any. See DPTX/DSC Specific Information and DPTX/TSF Specific Information in Section 3 of The Distributed Processing Terminal Executive Guide, IDR4035, for

details for some Prime-supplied protocols.

For key = K$INFS:

word 1: Device type:

0: terminal

1: printer
This is presently used for 3270 printer emulation (as part of DPTX/DSC) and support (as part of DPTX/TSF) and indicates the type of device rather than the Protocol Handler.

word 2: Device type flags:

bit 12 = 1: "input waiting" condition enabled (in configuration).

bit 13 = 1: input allowed from device.

bit 14 = 1: output to device allowed.

bit 15 = 1: block mode allowed.

bit 16 = 1: this device is a Primos terminal.

word 3: Screen size: Maximum physical buffer size (bytes) of the device.

word 4: Protocol type:

0: undefined

1: DPTX/TSF (3270 Terminal Support)

2: DPTX/DSC (3270 Emulation port)

words 5-6: (reserved)

words 7-10: Protocol-specific status information, if any. See DPTX/DSC Specific Information and DPTX/TSF Specific Information in Section 3 of The Distributed Processing Terminal Executive Guide, IDR4035, for details for some Prime-supplied protocols.

BD$LST: List BDI Configuration Data

CALL BD$LST(key, name, nameln, datbuf, datlen, code)

This is a new call. It fetches information about the current DPTX configuration as maintained by the Block Device Interface.

key Indicates the type of information requested by the user. This argument is referenced but not altered by BD$LST.

K$INFD: Information (as described below) about a specific device whose "name" is specified in name is returned in datbuf.

K$INFN: Information (as described below) about a specific device whose logical station ID is specified by the caller in datbuf(1) is returned in the remainder of datbuf and in name.

K$LTAT: A list of the logical station ID numbers which correspond to devices residing on the logical synchronous line specified in datbuf(1) is returned in the remainder of datbuf.

K$LPAT: A list of the logical Poll Group numbers which correspond to poll groups (CUs) residing on the synchronous line specified in datbuf(1) is returned in the remainder of datbuf.

K$PTAT: A list of the logical station ID numbers corresponding to devices in the poll group (attached to the CU) whose logical poll group number is specified in datbuf(1) by the user is returned in the remainder of datbuf.

K$PATD: A description of the poll group whose logical poll group number is specified in datbuf(1) is returned in the remainder of datbuf.

K$BSYS: Get system data. The total number of poll groups and the total number of devices in the configuration, the identification of the protocol handler and the number of devices and the number of poll groups on each line is returned to the caller in datbuf.

name The name of the device for key = K$INFN and K$INFD. For key = K$INFD, the name must be specified and must exist in the TAT definition tables. For key = K$INFN, as many characters (left justified, blank filled) of the device name as will fit (as specified in nameln) are returned to the user in name. If the device name must be truncated, a code of E$BFTS (buffer too small) is returned. The user is not informed of the actual name length. BDI device names are a maximum of 32 characters long. For K$INFN, if nameln is specified as zero, name is left unaltered. For all other values of key this field is unaltered and not referenced by BD$LST, the user may therefore specify it as he wishes, but it may not be omitted.

nameln The length of name in characters. Specified by the user. For keys of K$INFN and K$INFD, this argument is referenced

but not altered. For other values of key, this argument is not referenced.

datbuf An array of returned data, the contents of which vary depending on the key specified.

K$INFD The device specified by name is

K$INFN described in datbuf. If code = E$BDEV (device not found), the contents of this array are invalid.

word 1 Logical station id of the device.

word 2 Protocol type:

1 = DPTX Terminal Support

2 = DPTX Data Stream Compatibility

word 3 Device type:

0 = terminal

1 = printer

word 4 Station capabilities (bit encoded): This word is exactly the same as INFARR(2) returned from a call to BD$INF with a key of K$INFN or K$INFD.

bits 1-11: (reserved)

bit 12: Read before write required.

bit 13: Input allowed from device.

bit 14: Output to device allowed.

bit 15: Block mode operation permitted.

bit 16: Prime Standard Terminal (DPTX-TSF only).

word 5 Maximum buffer size of the device.

word 6 Maximum message size defined for the device's protocol.

word 7 Physical device character set:

0 = ASCII

1 = EBCDIC

word 8 Station status:

1 = offline

2 = device marked down by protocol handler.

3 = device operational.

word 9 Owner's process number (if zero, the device is not owned).

word 10 Device address (word 1) in that device's physical character code. (For DPTX, only the lower byte is valid, and is the EBCDIC representation.)

word 11 Device address (word 2) in that device's physical character code. (For DPTX, only the lower byte is valid, and is the EBCDIC representation.)

word 12 Physical station identification:

bits 1-8 Logical line number (for synchronous lines, this value is in the range 0-7).

bits 9-16 Logical poll group number.

word 13 Printer information (not valid for devices other than printers):

bits 1-8: Platen length.

bits 9-15: (reserved)

bit 16: Vertical Forms Control on. (DPTX 3270 products only)

word 14 Maximum output queue length.

word 15 Maximum input queue length.

K$LTAT List all the logical station IDs corresponding to devices residing on the synchronous line (0-7) specified in datbuf(1). If datlen is less than two, no processing is performed for this call except identification of the key, and a code of E$BPAR is returned.

datbuf(1) The synchronous line number. Supplied by the caller, this is not altered by BD$LST. If its value is outside the permissible range of synchronous lines, a

code of E$BDEV is returned to the user.

datbuf(2) The number of block devices configured on the specified line.

datbuf(3...) The logical station IDs of the devices on this line. If datlen indicates a buffer too small to hold all the ids, this list is truncated at the last one which can fit in datbuf, and a code of E$BFTS is returned. If datlen exceeds the size required, the remainder of datbuf is valid only up to the number of entries specified in datbuf(2) (i.e. if datbuf(2) equals one, datbuf(4) through datbuf(datlen) are invalid).

K$LPAT Return a list of all the poll groups defined on the synchronous line specified in datbuf(1). If datlen is less than two, no processing is performed for this call other than the identification of the key, and a code of E$BPAR is returned. The returned values are similar to those specified for K$LTAT.

datbuf(1) The synchronous line specified by the user. This entry is not altered by BD$LST. If it lies outside the permissible range of synchronous lines, no further processing is attempted, and a code of E$BDEV is returned.

datbuf(2) A count of the logical poll groups residing on the synchronous line specified in datbuf(1). If 0, this indicates that no block device poll groups are defined on that synchronous line in the current configuration.

datbuf(3...) A list of the logical poll group numbers corresponding to poll groups defined on the synchronous line specified in datbuf(1). If datlen indicates a buffer too small to hold all the IDs, this list is truncated at the last one which can fit in datbuf, and a code of E$BFTS is returned. If datlen exceeds the size required, the remainder of datbuf is valid only up to the number of entries specified in datbuf(2) (i.e. if datbuf(2) equals one, datbuf(4) through datbuf(datlen) are invalid).

K$PTAT Return a list of the logical station IDs corresponding to the devices associated with the poll group whose

logical ID number is specified in datbuf(1). If datlen is less than two, no processing is performed for this call other than the identification of the key, and a code of E$BPAR is returned.

datbuf(1) The logical poll group number. Supplied by the caller, this is not altered by BD$LST. If its value is outside the permissible range of poll group numbers, a code of E$BDEV is returned to the user.

datbuf(2) The number of block devices configured for the specified poll group.

datbuf(3...) The logical station IDs of the devices on this poll group. If datlen indicates a buffer too small to hold all the IDs, this list is truncated at the last one which can fit in datbuf, and a code of E$BFTS is returned. If datlen exceeds the size required, the remainder of datbuf is valid only up to the number of entries specified in datbuf(2) (i.e. if datbuf(2) equals one, datbuf(4) through datbuf(datlen) are invalid).

K$PATD Return a description of the logical poll group. If datlen is less than two, no processing is performed for this call other than the identification of the key, and a code of E$BPAR is returned. datbuf is filled only up to the minimum of datlen or six elements. If datlen is less than six, a error code of E$BFTS is returned, but the array is filled up to and including the last element as described below:

datbuf(1) Logical poll group number. Supplied by the user. If this value is outside the range configured, a code of E$BDEV is returned and no further processing is enacted.

datbuf(2) Poll group type:

1 = DPTX 3270 Terminal Support

2 = DPTX 3270 Terminal Emulation

datbuf(3) The number of the synchronous line upon which this poll group resides.

datbuf(4) The number of logical devices defined to be associated with this poll group.

datbuf(5) The poll group status:

1 = Undefined (offline)

2 = Marked down by protocol handler

3 = State is unclear, potentially up

4 = Operational

datbuf(6) Poll group address. For DPTX, only the lower six bits of this address are valid.

K$BSYS Return system data. This key requests general information about the current configuration. If datlen is less than one, no processing is done beyond the identification of the key, and a code of E$BPAR is returned. A maximum of datlen or four times the number of PRIMOS configured synchronous lines plus two (4 * (# synchronous lines) + 2) parameters are returned, whichever is smaller. Currently, the number of PRIMOS-configured synchronous lines stands at eight. If datlen is the smaller, a code of E$BFTS is returned. If datlen is the larger figure, the remainder of datbuf is untouched.

datbuf(1) The total number of block devices configured.

datbuf(2) The total number of poll groups configured.

datbuf(3, 7, 11...) The protocol type of the protocol handler on this line. If this parameter has the value 0, no protocol handler is presently operating on this line.

1 DPTX 3270 terminal support.

2 DPTX 3270 terminal emulation.

datbuf(4, 8, 12...) The PRIMOS user number of the protocol handler on this line.

datbuf(5, 9, 13...) The number of block devices defined on this line.

datbuf(6, 10, 14...) The number of poll groups defined on this line.

This last set of datbuf parameters is most easily described in terms of a PL1-like data structure:

```
DCL 1 SYSINFO,
      2 TOTAL_DEVICES fixed bin (15),
      2 TOTAL_POLL_GROUPS fixed bin (15),
      2 PER_LINE_DESC (NUMBER_SYNCHRONOUS_LINES),
          3 PH_TYPE fixed bin (15),
          3 PH_USER_NO fixed bin (15),
          3 LINE_DEVICES fixed bin(15),
          3 LINE_POLL_GROUPS fixed bin (15);
```

datlen The size of the array databuf (words).

code The value of this parameter indicates the success or failure of the call, and the reason for the latter if it occurs.

0: Action performed, all specified parameters returned as defined.

E$BFTS: nameln was too small (key of K$INFN only), or datlen was too small (all keys) to contain the data.

E$BPAR: datlen was too small to even begin processing the request.

E$BLEN: nameln was less than 0 for a key of K$INFN, or less than or equal to 0 for a key of K$INFD.

E$DNAV: Device not found in configuration tables (key = K$INFD).

E$BKEY: key not recognized.

E$BNWD: For a key of K$INFN or K$INFD, the value of datlen was specified as less than 0.

E$BDEV: The device number (key = K$INFN), line number (key = K$LTAT or K$LPAT), or poll group number (key = K$PATD or K$PTAT) specified by the user in databuf(1) was outside the range defined in the current configuration.

E$DNC No BDI configuration has yet been done.

The following example may serve to illustrate the use of this entry:

```
      INTEGER*2 DATA(15), CODE, I
$INSERT SYSCOM>KEYS.F
$INSERT SYSCOM>BDKEYS.INS.FTN
      CALL BD$LST(K$INFD, 'MY_DEVICE', 9, DATA, 15, CODE)
      IF(CODE .NE. 0) GO TO 100
      WRITE(1, 50) (DATA(I), I = 1, 15)
50    FORMAT(//'Description of MY_DEVICE:',
     +   /'Logical Station ID:', I2,
```

```
     +   /'Protocol type:', I2,
     +   /'Device type:', I2,
     +   /'Configuration bits:', I2,
     +   /'Device buffer size:', I5,
     +   /'Maximum message size:', I5,
     +   /'Physical device code:', I4,
     +   /'Device status:', I2,
     +   /'Owner user ID:', I2,
     +   /'Device address:', 2I4,
     +   /'Line and CU numbers:', I8,
     +   /'Printer information (?):', I4,
     +   /'Maximum output queue length:', I2,
     +   /'Maximum input queue length:', I2//)
         STOP
 100     CALL ERRPR$(K$NRIN, CODE, 'We blew it!', 11, 'DEMO', 4)
         STOP
         END
```

DPTCFG: THE DPTX CONFIGURATION COMPILER

This section describes the user interface to the DPTX configuration compiler (DPTCFG) as of Rev 18.2. Included are descriptions of the command line, and the source level input to the compiler. A complete catalogue of the error and warning messages follows.

Substantial changes in the functionality of this product have been made for Rev 18.2, and the program itself has been entirely rewritten. This section should be carefully read before using the Rev 18.2 DPTCFG.

Introduction

DPTCFG is an external command used to invoke the DPTX configuration compiler. The principal function of this program is to translate a source input file describing the physical configuration (as seen by DPTX) of IBM 3271 compatible devices and "host" machines into a compact configuration table which is output as a "binary" file. This latter file is then loaded into the system when DPTX is initialized.

Several other options have been provided that yield a better user interface. One may now reverse the compilation process, producing a source file from a compiled binary. This will be of some use should the user wish to determine the contents of a binary file and not have access to the corresponding source file. There is the further advantage that should the format of the binary file change to accommodate future changes in DPTX, one may reverse the compilation of the binary files with the "old" version of DPTCFG and then recompile with the new version.

An option to generate a tabular description of a configuration at the user's terminal and/or into a file has been added. As the binary file format was never intended to be particularly user visible or

interpretable, this will enable the user to immediately confirm the contents of a particular configuration. A tabular format is also considerably easier to read than the typical source file. Coupled with this is an option to suppress binary/source file output.

Lastly, an error listing file generation option has been added. This output file has an appearance similar to the output listing files of the more familiar language translators (FTN, PL/1, etc.), and it lists the input source statements interspersed by applicable error and warning messages (if any). This tends to alleviate some of aggravation caused by the error message handling of the previous product.

Command Line Syntax

The DPTCFG command line is of the form:

```
DPTCFG input_treename options
```

Where input_treename is the treename of the input file, whether source (when -REVERSE is omitted) or binary (when -REVERSE is present) see below.

Valid options are:

-OUTPUTFILE treename -OUTPUT treename	Specify the output treename that is to be used for the configuration file (if -REVERSE is not specified) or source file (if -REVERSE is specified).
-NO_OUTPUT -NOUT	Do not generate an output file (a configuration file if the -REVERSE option is not given, a source file if it is). Obviously, -OUTPUT and -NO_OUTPUT are mutually exclusive.
-ERRLIST treename -ERRL treename	Specify the file to contain an error listing.
-REVERSE	Use the configuration file as input and generate a source file as output (the latter is inhibited if -NO_OUTPUT was also specified).
-LIST treename -L treename	Produce a tabular description of the configuration, and place it in the file specified.
-TTY [screen_length]	Produce the same tabular description as for -LIST (above), except output it to the user terminal. (The program stops outputting data to the terminal after a full screen of data has been output, after a full group or after several groups if they will all fit on the screen at one time, and then prompts the user

for a termination (Q or QUIT) or continuation (anything else).) screen_length defaults to 23 lines, and must not be less than 9.

The options -ERROR and -REVERSE are mutually exclusive. Pathnames follow the file naming standards. The following file suffixes are used by DPTCFG:

.DPTCFG	source file
.CONFIG	binary file
.ERROR	error listing file
.LIST	display listing file

The input treename must be present. If any option requiring a treename (-OUTPUTFILE, -ERRLIST, -TTY) has no treename specified in the command line, that treename is inferred from the input treename

The default actions for DPTCFG are to generate a configuration output file using the same base name as the source input file, and to suppress error and display listing generation.

Should the command line contain an error, DPTCFG will not process the input file, nor write any of the output files. Similarly, if there is an error in the input file, DPTCFG will not write any output file, with the exception of the -ERRLIST file (if specified).

An Example

The following examples are intended to illustrate the more common uses of DPTCFG.

```
DPTCFG MYFILE
```

DPTCFG will search the current UFD for the file MYFILE.DPTCFG, if this search fails, the file MYFILE will be searched for. Should both searches fail, DPTCFG will terminate with an appropriate error message. The file which is found will be used as the source input.

DPTCFG will then open the file MYFILE.CONFIG (creating it if necessary) and use it for binary output. Error messages (if any) will be sent to the standard user output stream. When compilation is complete, a message such as "0002 ERRORS (DPTCFG Rev. 18.2)" will be output.

Another Example

```
DPTCFG MYFILE -OUTPUT FOO -ERRL -TTY -LIST FOOBAR
```

The source input file will be searched for as above. The binary output file FOO will be opened. As the input file is compiled an error

listing file (MYFILE.ERROR) is produced, listing each source statement and its line number, followed by any error messages pertaining to it.

If the source file is fault-free, DPTCFG will output a tabular description into a listing file (FOOBAR) when the compilation is complete. When this has finished, a similar table will be output to the standard user output stream (COMOUTPUT file and/or user terminal) using the interaction previously described.

DPTCFG SOURCE FILES

The two major changes to DPTCFG source file grammar at Rev 18.2 are the addition of the ENABLE GENERAL_POLL statement, and the elimination of the need for device and group numbering. Device and group numbers are no longer required, and their use is discouraged, but DPTCFG will check them, if present, for correctness. See DPTCFG Warning and Error Messages, below.

Source Input File Grammar

Lines in a DPTCFG source file are parsed (for the most part) according to the rules for the operating system routine RDTK$$. One notable exception is that a decimal number specified as the object of a keyword may be entered either as a string of decimal digits or as a colon followed by a string of octal digits. In the latter case, the octal number supplied will be appropriately converted to its decimal equivalent.

Upper and lower case are not differentiated and may be used interchangeably. The source file consists of series of structured blocks, each one defining an individual group, where a "group" is equivalent to an IBM 3271 control unit, whether real or emulated. A group definition contains exactly one "Define group" statement, followed by a variable number of "Define device" statements. Both of these statements are followed by a variable number of keywords which specify attributes (e.g. address C1). A statement is terminated by the next occurrence of a major command, or the end of the file. Blank lines are ignored in statement parsing, and their use is encouraged for clarity. The basic block structure is outlined below:

```
DEFINE GROUP
   keyword_attributes
  DEFINE DEVICE
     keyword_attributes
  DEFINE DEVICE
     keyword_attributes
```

The total number of groups and devices must not exceed the constraints specified by DPTX (currently these limits are 32 groups and 32 devices in any combination, with the obvious restriction that a device must be associated with one -- and only one -- group). DPTX at Rev 18.2 supports no more than 32 devices, total.

Major Statement - DEFINE GROUP

This statement must precede any definitions for devices, and must be immediately followed by the "DEFINE DEVICE" statement for each device in the group. The format of the statement is:

DEFINE GROUP keyword_attributes

The following keyword_attributes are required and will generate an error if not present:

PROTOCOL {EM3270 | SP3270}

Specifies the type of service (Emulate or Support) for which devices in this group are to be used. EM3270 specifies that the devices are virtual 3277 terminals to be emulated, and SP3270 specifies that the devices are 3277 terminals to be supported. Protocols may not be mixed on the same synchronous line.

LINE n

Specifies the logical line number of the synchronous line which this group is on. n is currently restricted to the range 0 thru 7.

ADDRESS nn

Specifies the address of this group. The address must be specified as a two digit hexadecimal number which represents an EBCDIC character. For a list of valid addresses, consult the "IBM 3270 Information Display System Component Description" (IBM publication GA27-2749).

The following keyword attribute is optional:

ENABLE GENERAL_POLL

This keyword specifies that particular operations or modes of operation are enabled for a particular control unit. Currently there is only one option, and it is restricted to SP3270 control units. If not specified, the default is to disable general polling by DPTX to SP3270 control units.

Enabling general polling on SP3270 control units allows DPTX-TSF support to general poll that control unit. This usually results in a vast reduction in non-productive line traffic, which yields a significantly higher throughput for terminal I/O bound applications. The sole disadvantage to using general polling on a control unit is that the Test Request key on any terminal attached to that control unit will be ignored by DPTX-TSF. When this occurs input will be inhibited until the terminal operator resets the keyboard. Under DPTX-TCF, the IBM host may also reset the

keyboard, but this latter is entirely dependent on the host's programming.

Major Statement - DEFINE DEVICE

This statement specifies the attributes associated with individual devices in a group. The format of the statement is:

DEFINE DEVICE keyword_attributes

The following keyword_attributes are required:

NAME 32 character name

Specifies the name of this device for Block Device attaches. The first character must be a letter, and the remainder of the name should follow the PRIMOS standard file name conventions for non wild-card names.

ADDRESS nn

Specifies the device address. nn is interpreted the same as in group addresses.

The following keyword_attributes are optional:

USER n

(Only legal for terminal devices in groups for which the SP3270 protocol has been specified.) Specifies the user number associated with this device when it is used to issue commands to PRIMOS.

Note

There must exist Primos terminal buffers associated with this user number (i.e. the Primos configuration file must specify the number of terminal users such that it includes the users specified for all TSF devices). Further, processing of the characters in these terminal buffers by the Primos AMLC process must be disabled to prevent the loss of characters (enter AMLC TTYNOP line number where line number is the octal representation of the AMLC line number (nn - 2) normally associated with this user process. See The System Administrator's Guide, PDR3109, for further details on the AMLC command.

ENABLE list_of_enable_options

Specifies the operations which may be performed on this device and the modes of access to the device available to the user. This keyword and its associated options are illegal for a printer

device.

READ: The device may be read from
WRITE: The device may be written to
BLOCK: The device may be manipulated by Block Device calls
COMMAND: The device is to be considered a user terminal, for command input to Primos. This option is only valid for terminals in a SP3270 group.

If ENABLE is not supplied by the user, DPTCFG supplies the defaults READ, WRITE and BLOCK for EM3270 terminals and SP3270 terminals for which USER has not been specified. Should the user specify the USER option for an SP3270 device, the ENABLE option defaults to READ, WRITE and COMMAND if not specified. READ and WRITE must be enabled on a SP3270 COMMAND terminal.

PRINTER [VFC | PLATEN nnn]

Specifies that the device is a printer. The options VFC and PLATEN may be specified, but are not necessary.

VFC indicates that the Vertical Forms Control (hardware page eject) is available on the physical printer. If this option is not specified for a DPTX SP3270 printer, page ejects in files directed to a DPTX supported IBM 3270 printer by the standard Primos spool system will be performed by successive line feeds.

PLATEN nnn specifies the platen width in characters. If this option is included, the decimal number nnn in the range from 1 to 255 inclusive must be specified. If this option is not included, the printer will be assumed to have a platen width of 80 characters.

Sample Input File

The following is a sample input file and the tabular listing file produced from it:

```
Define Group
   Address 40
   Line 0
   Protocol SP3270
   Enable General_poll
  Define Device
     Name Support.1
     Address 40
     Enable Read, Write, Block, Command
     User 32
  Define Device
     Name Support.2
     Address C1
     Enable Read, Write, Block, Command
     User 33
```

```
Line  0 Protocol: SP3270 EBCDIC
        Name          Addr  Inq Outq   Type        Enable
                            Maxmsg                Platen VFC   User
Group Address: 40, General poll

SUPPORT.1             40     7   3  Terminal Read Write Block Cmd
                            2048                              32
SUPPORT.2             C1     7   3  Terminal Read Write Block Cmd
                            2048                              33
```

DPTCFG WARNING AND ERROR MESSAGES

Error and warning messages can be categorized by their origin: errors in the command line parsing or in file system operations, and those caused by violations of the source grammar (or binary data structure rules -- if the REVERSE option is specified) in the input file.

Command line and file system errors are fairly simple, and the error messages are very clear. As these error messages are generated more by the operating system than they are by DPTCFG, this document will not attempt to describe them. For further information on these errors, the reader is referred to The Prime User's Guide IDR 4130.

Command line syntax violations result in messages describing the correct command syntax, and /termination of the program. No output

files are created, nor is the input file affected.

File system errors encountered while processing results in an immediate termination of the program. Furthermore, any output files that have been created as a result of the invocation of DPTCFG are deleted. The input file is unaffected.

Any type of error inhibits the output of the DISPLAY or LISTING options.

Source Input File Errors

DPTCFG produces only two severity levels: warning and error. Warnings are caused either by redundant option specifications for a group or device, or by conflicting device and/or group numbering in the source statements. Group and device numbering is to be considered obsolete henceforth for the reasons given under Compatibility with Previous Revs, below. Other than producing warning messages, DPTCFG ignores numbering conflicts and produces the output specified.

Errors are caused by conflicting or missing information in the source statements which cannot be supplied or resolved by DPTCFG. No output files are produced (with the exception of the ERROR file -- if requested).

Warning Messages

Here are the warning messages in alphabetical order. When a word in a message is entirely capitalized, as "GROUP", it refers specifically to a keyword, or its arguments. When only the first letter is capitalized, as "Group", a more general meaning is intended.

DPTCFG maintains its own internal count of devices and groups. The device and group numbers to which most warning messages refer are obsolete. When the user specifies a group or device number, or the numbers of the devices within a group incorrectly, DPTCFG flags the occurrence with a warning message. o DEVICE count of previous group incorrect.

The number of devices inferred from the DEVICE mm, nn specifier of the DEFINE GROUP statement differed from the actual number of device definitions for this group.

- Device number out of sequence/range.

In the DEFINE DEVICE nn statement, the device number nn is not that expected.

- First device number out of sequence.

In the DEVICE mm, nn specifier of the DEFINE GROUP statement, the number of the first device (nn) is either less or greater than the next number in sequence.

- Group number out of sequence.

The group number given in the DEFINE GROUP statement is out of sequence.

- Last device in Group out of sequence.

In the DEVICE nn, mm specifier in the DEFINE GROUP statement, mm is less than the current count of devices already defined.

- More devices than defined in GROUP.

In the DEFINE GROUP statement, the DEVICE nn,mm specifier mm did not match DPTCFG's internal count of the devices specified when the end of that group definition was reached.

- Redundant <option> specification.

The user has specified some option twice (e.g.: given two addresses for a group). The second and all subsequent re-specifications are ignored. option may be one of the following:

For groups:

```
ADDRESS
DEVICE (obsolete)
LINE
PROTOCOL
```

For devices:

```
ADDRESS
ENABLE: COMMAND | BLOCK | WRITE | READ
NAME
PRINTER PLATEN
PRINTER VFC
USER
```

If a warning message is generated that indicates trouble with the following device keywords, INPUTQ, MAXMSG, or OUTPUTQ contact your Prime System Analyst.

Error Messages

Error messages come from two sources, those resulting from conflicts in the syntax of the source statements and those resulting from conflicts in the information given, or from insufficient information. DPTCFG's error messages are as follows:

- ADDRESS missing in previous Device definition.

The user did not specify an address for the device.

- ADDRESS missing in previous Group definition.

As above, but for a group.

- Address already used in Group.

Two devices in the same group have the same address. Note that this may result from a bad DEFINE GROUP statement.

- Address already used on line.

Two groups, defined to be on the same line, have the same address.

- Duplicate USER id.

Two DEFINE DEVICE statements specified the same user number for a command terminal (USER nn option).

- Duplicate name in configuration.

A previous DEFINE DEVICE statement provided the identical (ignoring case) device name as the current DEFINE DEVICE statement. Device names must be unique.

- ENABLE COMMAND illegal on printer.

The current DEFINE DEVICE statement specified that the device is a printer and that it is a user (PRIMOS) terminal. These two options are mutually exclusive.

- ENABLE GENERAL_POLL illegal under EM3270 protocol.
  or
- GROUP ENABLE GENERAL_POLL illegal for non SP3270.

The ENABLE GENERAL_POLL option is legal only for SP3270 groups. (EM3270 groups must always accept general polls.)

- Heterogeneous protocol on line.

Two DEFINE GROUP statements conflict in their definition of the line protocol. SP3270 and EM3270 cannot exist simultaneously on the same line.

- LINE missing in previous Group definition.

In the previous DEFINE GROUP statement, the mandatory LINE n clause was omitted.

- Missing: [<keyword>|<object value>]

The file is incomplete. A keyword option (for example the user number in the USER clause), or possibly a DEFINE DEVICE statement for a group is missing.

- NAME missing in previous Device definition.

The mandatory NAME clause was missing from the previous DEFINE DEVICE statement.

- Non 3271 address.

The value specified in the ADDRESS clause of a DEFINE DEVICE or DEFINE GROUP statement is not a legal 3271 address. (This address must be expressed in hexadecimal.) The "IBM 3270 Information Display System Component Description" manual (GA27-2749) provides a list of valid control unit and device addresses.

- Printer platen length too large.

The platen length specified must not exceed 255 characters.

- PROTOCOL missing in previous Group definition.

The user did not include the mandatory PROTOCOL clause in the DEFINE GROUP statement.

- Unexpected source_literal.
  Expecting: [keyword|object identifier
- ...keyword|object identifier]

This indicates that the user either omitted some keyword or object, or did not correctly specify the object. Specifying two numbers where one is required, or specifying a group's protocol as other than SP3270 or EM3270 are two examples.

- USER and ENABLE COMMAND illegal on printer.

The USER and PRINTER clauses were specified in the same DEFINE DEVICE statement. The grammar defines these two to be mutually exclusive. A printer is not a user terminal.

- USER illegal on printer.

The same as above. (The difference being the order in which the two clauses were specified.)

- USER missing in previous Device definition.

The COMMAND option in the ENABLE clause of a DEFINE DEVICE statement was specified, but the corresponding USER clause was missing.

Binary Input File Errors

When the REVERSE option is specified, the input file is expected to be a binary file produced by a previous invocation of DPTCFG. There is only one error message associated this case, "Bad Config file format." which indicates that the binary input file that the user specified contains inconsistent information (according to the structure established for DPTCFG). No further information is given, as it is assumed that the user should be editing only source files, and therefore the file in question cannot be a compatible DPTCFG binary file.

COMPATIBILITY WITH PREVIOUS REVS

Previous Revisions of DPTCFG differed somewhat from the current product. In the older version, the default output filename was DPTCON. This was changed to conform to the recently established filenaming standard, and to make the implementation of the reverse compilation easier. The default output filename is now determined from the input filename (not the treename, but merely the filename).

The format of the source file statements has changed somewhat, although

DPTCFG will still correctly process files created for the old compiler. The old compiler required keywords (as defined in the section on Source Input File Grammar) to be prefaced by a hyphen ("-"). This is no longer required, and the presence of such on keywords is ignored.

The device and group numbers have been eliminated. These too are recognized by the current compiler and although they are checked for syntax and correct values, they are largely ignored.

In removing the requirement in DPTCFG for ordering the groups by protocol in the input source file, it was necessary to put the ordering process in the compiler itself, as the binary output file (currently) must have this ordering. The group numbers and device numbers are therefore somewhat irrelevant, and are not preserved.

This does mean that the device number returned in a BD$ATT call (see the Block Device Interface section of this document) may not have any discernible relationship to the device number specified in the configuration. (Although the source and display files produced by DPTCFG will have this ordering.) As the original design of the interface intended that users reference a particular device only by its name and some label defined by the interface itself, this is not seen to be a significant hindrance.

Also as a result of removing the ordering requirement from the source file (as described above), the DEVICE keyword of the DEFINE GROUP statement became superfluous. Again, it will still be recognized by DPTCFG, but it is no longer a required entry in a group description.

Violations of the old DPTCFG grammar that relate to group and device numbers result in warnings, but no fatal errors.

Error and warning message reporting has changed significantly, mostly as a result of the separation of the reporting function from the parsing and syntax checking. The new format should provide equivalent functionality.

Old DPTCFG Source Files

It is suggested that users of the old DPTCFG remove the device and group numbering from their configuration source files and re-compile them. This will save considerable effort should support for device and group numbering be discontinued at some future date.

DPTX STARTUP

Process Priorities Automatically Set

The information here affects page 4-9 of The Distributed Processing Terminal Executive Guide, IDR4035.

At Rev 18.2, BSCMAN (started up by PH SYSTEM>PH_BSC) automatically sets its priority to 3. EM3270 and TM3270 (started up by PH SYSTEM>PH_EMn and PH SYSTEM>PH_TM) automatically set their priorities to 2. The system operator need not change the priorities of these DPTX processes.

Automatic Line Assignment

The information below supplements page 4-9.

Previously to Rev 18.2, the command file SYSTEM>PH_BSC contained statements to assign the synchronous lines used by DPTX at a particular installation. As shipped, PH_BSC had statements to assign synchronous lines 00 through 03 to the BSCMAN process.

At Rev 18.2 DPTX, the assignment of synchronous lines is done automatically, based upon the DPTX configuration established for each of the DPTX processes. Thus, if PH_EM4 is brought up, BSCMAN will dynamically assign synchronous line 4.

WARM STARTING DPTX

As of Rev 18.2, DPTX automatically recovers from a PRIMOS warm start. When a warm start occurs, the MDLC controller is reset by the Master Clear operation, and BSCMAN disconnects all DPTX processes associated with it. Each of these various DPTX processes will then automatically reconnect with BSCMAN. During this time, which may be as much as a few minutes, host communcation (for DSC and TCF) and control unit communication (TSF and TCF) is temporarily interrupted. Both the IBM host and the control unit should tolerate such an interruption. However, transactions which were incomplete are the application programs' responsiblity to recover. The output log of each DPTX process will indicate when a warm start has occurred.

DPTX at Rev 17.8 has a partial warm start restart capability in that the system need not be cold started to restart BSCMAN. For Rev 17.8, after a warm start has occurred, a DPTX -OFF, then DPTX -ON should be done.

MULTILINE TRAFFIC MANAGER

This section describes the parameters which are supplied to the DPTX-TSF Traffic Manager at initialization and some subtle interactions between them which may affect system performance in unsuspected ways. TM3270 is used only in the DPTX-TSF and DPTX-TCF products, not DPTX-DSC. These notes result from laboratory experience, and will undoubtedly be amended as that experience increases. These comments are specific to Rev 18.2 (and later) DPTX.

Introduction

Several parameters are provided at initialization time to control polling and recovery rates for configured devices and control units.

These parameters are usually provided in the command file used to startup the Traffic Manager phantom (sometimes referred to as TM or the STM process). Should TM be started from a user terminal (a practice which is not encouraged, as this rarely provides any useful information, and then only in the debugging process), the user is prompted for this information.

The information required is:

- Line recovery delay. Sometimes, due to excessive line flutter, TM "shuts" down a line — the line is deconfigured, and both the receiver and transmitter are turned off at the hardware level. Periodically, TM will attempt to recover the line by reconfiguring it. This delay time specifies the period between the time the line is deconfigured and the time recovery is initiated.

- Control Unit recovery delay. When a protocol violation by a control unit occurs, TM ceases to poll that unit for this period of time. The actual delay time between a protocol violation and re-activation of traffic to and from a CU is of course dependent on whether the line is active or in recovery (in which case the line recovery delay — above — has an effect).

- Device recovery delay. When a device status indicates that it is nonoperational due to some hardware failure, TM ceases to direct traffic (specific polls and selects) to that device. This also occurs when a time-out occurs on a specific poll or device select. The delay specified is the maximum period between the time TM "marks the device down" and the time recovery (resumption of specific polling and device selection) occurs. The actual time will vary, depending on the line and CU recovery delays (if relevant) and whether a device status indicating that the device is operational is received during a general poll to the corresponding CU.

- Specific polling period for devices attached to a CU for which general polling is not enabled. The period specified is the minimum period between specific polls to a particular device.

- General polling interval. The period specified is the maximum period between general polls to a particular CU for which general polling has been enabled.

- Specific polling interval for devices attached to a CU for which general polling has been enabled. For such control units, it is occasionally useful to determine the status of a specific device, particularly when there has been no recent traffic to it. Applicable instances include printers which are in the midst of printing; should there be some mechanical failure, the user

process could not otherwise be notified of it. TM therefore performs periodic specific polling of each device on such a CU.

The recovery delays (in seconds) are specified first (all on one configuration line), followed by the polling intervals (in tenths of seconds, on the following configuration line).

Suggested Values

Users will have to derive their own optimal parameters through trial and observation. This is because DPTX configurations and user requirements vary. The following observations should help users to determine their own optimal values.

- Recovery delays should be significantly larger than the minimum polling interval (an order of magnitude should be sufficient).
- The relative ordering of recovery delays should be as follows:

  The line recovery delay should be the shortest.

  The group recovery delay should be a few times longer than the line recovery delay.

  The device recovery delay should exceed the group recovery delay, although it need not be any large multiple (1.5 to 2.0 should be adequate).

- Polling intervals will have a significant effect on response time for simple tasks. Caution is advised when decreasing these parameters, as extremely small polling intervals will result in correspondingly higher line and PRIME CPU overhead.
- Polling intervals should adhere to the following ordering:

  The general polling interval should be the smallest (on the close order of 0.3 to 1.5 seconds).

  The specific polling interval for use when general polling is enabled should be at least an order of magnitude larger than the general polling interval. Specific polling under these circumstances only serves to determine when devices, such as printers, which have been busy become available. Specific polling in this circumstance is important, but it need not be frequent.

  The specific polling interval for use when general polling is inhibited should be approximately the same as the general polling interval, possibly somewhat smaller. When specific polling is used, the terminal system response time is inevitably increased, which implies a correspondingly shorter polling interval, however greatly reduced polling intervals

result in greatly increased overhead. As a "rule of thumb", one might take the required response time and divide it by the total number of specific-polled terminals on a line, to get some feel for the minimum specific polling interval. This rule usually results in intervals which are highly skewed toward high-overhead. (It is probably better to enable general polling wherever possible. The only disadvantage is the loss of the TEST REQUEST key at the affected terminals.)

An Example

The environment is exclusively general-polling (that is, all SP3270 control units in the configuration have general polling enabled) and a relatively large number of devices (printers and terminals) are spread over a small number of control units. Previous experience with the configuration (both physical and logical) has indicated a high mean time between failure (MTBF) for everything except the printers.

Line, control unit and device recovery times (seconds):

1, 2, 12

The determining factor here is that the printers break down (jam) fairly frequently. The 12 seconds chosen as the device recovery time should provide for a fairly timely re-activation when the local operator has corrected the problem. The line recovery time was chosen mostly to insure that recovery attempts occurred in a period fairly close to the time specified.

Specific polling (without general polling), general polling, and specific polling (with general polling) intervals (tenths of seconds):

1800, 5, 100

Once again, the low MTBF for printers was the determining factor. As all control units have been configured for general polling, the specific polling rate for non-general-polled control units is irrelevant. The general polling interval is set fairly low, to yield a short response time. The specific polling interval for devices on control units which have general polling enabled is set to ten seconds to insure a timely determination of the status of a "down" or recently reactivated device (printer).

Another Example

The environment is the same as the same as the above, except that the modems on the lines have an extremely low MTBF and the total Prime system is usually moderately to heavily loaded.

Line, control unit and device recovery times (seconds):

20, 60, 60

As might be expected, the unreliability of the lines is the dominant factor. To avoid excessive traffic on the line and excessive CPU overhead, the line recovery delay is set to a relatively large value. The CU time is also fairly large, partially as a result of the large line recovery time and partially because some modem failures appear (to DPTX) as CU failures (time-outs for example). The device recovery time is set to track the CU recovery period.

Specific polling (without general polling), general polling and specific polling (with general polling) intervals (tenths of seconds):

1800, 20, 100

The general polling interval was increased here to decrease the CPU overhead somewhat, at some sacrifice in response time.

Observed Interactions

- When a control unit or device repeatedly times out, the specific polling rate becomes strongly affected by the device and control unit recovery periods. If these recovery delays are significantly shorter than the specific polling period, the polling period may well decrease due to the recovery algorithm. Conversely, if the recovery delays are significantly longer, polling to the particular CU/device will not resume for some time, and the perceived polling rate will significantly decrease.

- Extremely low polling periods (0 for example) may result in a proponderance of polling on the line as compared to the transmission of output (user process generated text).