# HP260 – HPBB
# COMPATIBILITY DRAFT : 1986

# HP260 TO HP3000
# MIGRATION TRAINING
# STUDENT WORKBOOK : 1986

* * * DRAFT * * *

# HP260-HPBB COMPATIBILITY

## Programming For Compatibility

EDIT
SEQUENCE

HEWLETT PACKARD
BGD



Location Code: HP-B200

Project Number: xxxx-xxxx

October 14, 1986

Subtitle: Tips and Tricks for the conversion

Dr. Ulrich Fauser

SEND SYSTEM OUTPUT TO

3 4/2

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

Section 7
**ARRAY OPERATIONS**



Section 8
**SCREEN AND PRINTER I/O**



Section 9
**USER DEFINED FUNCTION KEYS**



Section 10
**DATA BASE MANAGEMENT**



Section 11
**DATA BASE SEARCH AND SORT**



Section 12
**THE BASIC REPORT WRITER**

Table of Contents

# INTRODUCTION

SECTION 1

This is a technical paper for HP260 customers who are thinking of converting existing programs to the HP3000 with the help of HP BUSINESS BASIC/V and the existing conversion tools.
It is a summary of the experience known today from the conversion of existing HP260 applications.

The terms HP BUSINESS BASIC/V and HPBB will be used interchangeably.

Its purpose is to document all the little incompatibilities not mentioned in the HPBB manuals (programmer and reference manual) give some tips and tricks gathered from early start users in their conversion experience and help you around some obstacles still existing in HPBB by providing possible workarounds.

A special emphasis is put on HP260 users developing new applications on the HP260 with its known programmer's productivity but who are also considering later migration to the HP3000.

This paper will tell you what to watch out for, how to program for minimal conversion effort, maximum usage of the conversion tools and also minimizing the codespace generated by compiling your programs in the HP BUSINESS BASIC/V environment.

The experience given in this paper reflects the state of HPBB's released MR version in fall 1986.
For later versions not everything mentioned might still be true.

The reader is required to have a working knowledge of BASIC/260 and HP BUSINESS BASIC/V as facts documented in the manuals are not repeated in this paper.

This manual is built up mostly upon the structure of the HPBB Programmers Guide for easy crossreference.
An additional chapter 18 is included describing bugs and workarounds in the current version.

As a final comment please remember that HPBB is running under a different operating system (MPE) and on a different hardware and provides an interpreter as well as a compiler. Therefore it was not possible to achieve a 100% compatibility to BASIC/260.

# DATA TYPES, STRING OPERATIONS

## DATE$

This is a hint for non US customers only. The DATE$ function now returns the american date format. To get the date format you are used to use DATE$(<NLS number>) or overwrite the default by setting the date format in CNFGHPBB.

## DATA TYPES


In using data types there is one thing to consider. On the HP260 it was almost normal to leave variables undefined e.g. like FOR variables, counters etc without too much penalty. Remember these undefined variables are by default REAL.

Now on the HP3000 in HPBB there are two target data types to choose from REAL or DECIMAL, with DECIMAL being the exact corresponding type concerning accuracy in computations.

The conversion program BBCT250 allows you to select your target data type for HP260 REAL variables with OPTION REAL or OPTION DECIMAL.

Choosing OPTION DECIMAL

SHORT is converted to SHORT DECIMAL
REAL is converted to DECIMAL

In the data base schema this corresponds to K2 and K4 types.
It should be noted that a future version of the TURBO IMAGE schema processor will accept D2 and D4 but internally these are still treated as K types.
This choice gives you the exact same behaviour as on the HP260.
The disadvantage is QUERY does not yet recognize K2 and K4.
Other languages cannot interpret these formats
There is a problem sorting DECIMAL values (see SORT).
Performance is quite slow, as the HP3000 has only little hardware support for DECIMALs, most is done in software.

Choosing OPTION REAL

SHORT is converted to SHORT REAL
REAL is converted to REAL
In the data base schema this corresponds to R2 and R4

Other languages can read this data base too.
QUERY can handle this variables
Performance is much faster.
But you may have rounding errors in your computations which may affect your programs behaviour.

As a rule of thumb the performance ratio between the different data types in HPBB is roughly:

| SHORT INTEGER | INTEGER | SHORT REAL | REAL | DECIMAL | SHORT DECIMAL |
|---|---|---|---|---|---|
| 1 | 1 | 3 | 3 | 30 | 45 |

So for optimal performance and less codespace use INTEGER on the HP3000 whenever possible.
Do not use DECIMAL for bulk computations.
It it also good programming practice to declare all variables.
Use GLOBAL OPTION DECLARE to enforce it.

## SHORT INTEGER EXPRESSIONS

If all operands in an expression are of type SHORT INTEGER then the result is SHORT INTEGER also, even if the target variable is something else e.g. REAL.

```
REAL Date
SHORT INTEGER Year,Month,Day
Date=Year*10000+Month*100+Day
```

results in Short Integer overflow in HPBB, not on the HP260.

To avoid this change one operand to a different data type e.g.

```
Date=Year*10000.+Month*100+Day
```

Please realize that this discussion actually applies to all data types but is most noticeable with SHORT INTEGER due to the restricted range. Some users may have problems with INTEGER or SHORT DECIMAL too.

## STRINGS

Please note that the string concatenation operator on the HP260 is &, in HPBB it is + .

The behaviour of HPBB with empty substrings is different in two ways.

Ex.:
```
A$=""
IF A$[1,3]<>"XYZ" THEN ....
```
works on the HP260 and results in FALSE

in HPBB this gives you an error 18: SUBSTRING OUT OF RANGE
So to be compatible avoid this construct or add an additional check
e.g. IF LEN(A$)>=3 AND A$[1,3]<>"XYZ" THEN .....

Due to HPBBs partial evaluation of boolean expressions the right hand side of AND will not be evaluated if the left hand side is FALSE.

Accessing a substring immediately at the end of a string returns an empty string on the HP260

```
A$="123"
B$=A$[4]    !   this is valid on the HP260
```

HPBB has implemented this different in that it doesn't return an empty string but rather results in error 18 again.
This is quite a nuisance when implementing e.g. recursive string algorithms where an empty string is usually the ending condition.
Or if you program a FIND and REPLACE strategy in a textprocessing system this is usually done like the following example:

```
Posold=POS(Line$,Oldstring$)
IF Posold THEN
  Line$=Line$[1,Posold-1]&Newstring$&Line$[Posold+LEN(Oldstring$)]
END IF
```

Again this works ok in both BASIC versions except when the Oldstring$ happens to be at the very end of the line. Then only the HP260 succeeds as the remainder of the Line$ is the empty string. HPBB gives you an error 18.

To avoid this kind of stringhandling you have to add quite a few more statements to check if you have reached the end of the string.

## STRING OPERATIONS

The POS function has an undesirable effect when the second parameter is the empty string.

```
N$=""
I=POS("YyNn",N$)
```

returns 0 on the 260 giving the correct answer that "" is not contained in the first string, HPBB always returns a 1 in this case giving your program the impression that the user entered a Y.

To be compatible add checks to make sure there is no empty second stringparameter to POS.

Besides the above mentioned incompatibilities all string functions are available in HPBB too but they have a big disadvantage, when compiling

your program they use up the available codespace very fast. This applies especially to substring access, string concatenation and RPT$.
It is highly worthwhile to consider a different approach: building a small string library.

So instead of using the concatenation operator
write a concatenation function.

```
DEF FNConcat$(A$,B$)
  RETURN A$+B$   ! & on the HP260
FNEND
```

In this way the expensive concatenation operation appears only once in your programs, i.e. in your library.
To give you some ideas of the savings in HPBB:

```
C$=A$+B$       uses 68 words of code (with CONCAT=0)
C$=FNConcat$(A$,B$) uses only 35
```

the same is valid for substring access:

```
DEF FNSub$(A$,From,Length)
  RETURN A$[From;Length]
FNEND
```

```
C$=A$[1;5]+B$[6;11]  uses 244 words of code
C$=FNSub$(A$,1,5)+FNSub$(B$,6,11)  only 105 words
and
C$=FNConcat$(FNSub$(1,5),FNSub$(B$,6,11))   only 69 words
```

By doing this in a centralized way you can even work around the empty string problem with the following solution

```
DEF FNSub$(A$,From,Length)
  IF From=LEN(A$)+1 THEN RETURN ""
  RETURN A$[From;MIN(Length,LEN(A$)+1-From)]  ! may cause error 18
FNEND
```

For a single character substring there is still another codesaving solution possible, instead of A$[I;1] use CHR$(NUM(A$[I;1]))
The NUM function is clever enough to know that it always needs only one character.

Unfortunately the FNSub$ is only usable on the right hand side of an expression. E.g. you want to change the filename of your temporary file to get a name for a second workfile:

If Temp$="WRK"&USRID is your general workfile name, you can change it like:

```
Work$=Temp$[1;2]&"2"&Temp$[4]
```

which works fine in both BASIC versions but uses up 299 words of code in HPBB.
A less costly coding is:

```
Work$=Temp$
Work$[3;1]="2"
```

These two lines use only 41 words of code.

Hint: if you are using special characters in your filenames on the HP260 and take them out in converting to HPBB do not forget to change substring indexes too, otherwise your results may be surprising.

This also applies to RPT$ and possibly other string functions as well in a similar way.

A less costly and faster way, but requiring a little bit more effort, is to include the FN.. functions as single line functions (whenever possible) in every program where they are needed. This will result in less code and faster execution compared to using multi line functions.

## ENHANCED STRINGS

If you are using video enhanced strings the escapesequences are not converted correctly. The HP260 uses an CHR$(128) as <end of enhancement> what the HP3000 terminals do not recognize.
Replace this with '27"&d@"

# OPERATORS AND FUNCTIONS

SECTION 3

This chapter intentionally left blank. Its contents are integrated into chapter 2.

# PROGRAM CONTROL STATEMENTS

SECTION 4

## STRUCTURED STATEMENTS

The HP260 allows you to use some structured statements in an unstructured way like

```
IF X THEN NEXT I
or
IF A>9 THEN FOR I=1 TO 15
```

This is not allowed in HPBB but can easily be changed

```
IF X THEN Nexti
.
.
Nexti: NEXT I
or
IF A>9 THEN
  FOR I=1 TO 15
```

To be compatible do not use structured statements in the above manner.

## COMMAND

COMMAND can only be used in the interpreter. As a mixture of compiled and interpreted routines is not desirable due to performance reasons and because the main program then has to stay interpreted too, it is best to avoid COMMAND at all.

If it is totally unavoidable use it only in a separate son process by calling the interpreter with SYSTEMRUN and passing information via INFO or through files.

## FOR LOOP

Use only INTEGER or SHORT INTEGER control variables if possible. This speeds up performance and generates less code.

There is a heavy performance penalty and much more code is generated when using DECIMAL as control variable.

## SELECT

A SELECT statement can be very codespace consuming when selecting the wrong data type

in the following program excerpt Mode is DECIMAL

```
SELECT Mode
CASE 1
.
.
CASE 50
.
END SELECT
```

These 50 cases can generate around 800 words of compiled code as because Mode is DECIMAL all numbers are internally stored as DECIMAL and also all comparisons are made for DECIMAL.

You can improve this by either declaring Mode as SHORT INTEGER or by typing

```
     SELECT SINTEGER(Mode)
```

This generates only around 200 words of compiled code !!!! and runs much faster.

## HANDLING COMPILED AND INTERPRETED ROUTINES

When debugging your subroutines you'll be using them most certainly sometimes interpreted, sometimes compiled. In order to make the mixed usage easier consider the following piece of code:

```
! MENU Program  this is interpreted
.
CALL Pgma1       ! call the overlay routine
.
.
END    !MENU
SUB Pgma1    ! this is also interpreted
EXTERNAL Pgma11     ! this is the original Pgma1
.
  ON ERROR GOTO Loadsub
Callsub: CALL Pgma11
  SUBEXIT
Loadsub: ON ERROR GOTO Errorhandling
  IF INTERPRETED THEN GET SUB "PGMA11"
   GOTO Callsub
.
SUBEND
```

Its benefit is: it will first look if your routine is already loaded if yes it is calling the interpreted version.
If not and you have an EXTERNAL declaration it will look into your SLs from grouplevel up to systemlevel, if it still doesn't find the subroutine it jumps to the errorhandling routine where it tries to load the requested subroutine as interpreted code.
So if you already have a compiled version in a SL in order to test an interpreted version just do a GET SUB before executing the CALL.

## SUBPROGRAM NAMES

If you compile your programs in HPBB the SEGMENTER needs unique entrypoint names. On the HP260 it was quite common to name all subroutines identical like Pgm or Prog and distinguish them only through their filename.

To be compatible use unique subroutine names.

## MAIN COMMON BLOCK FOR SUBUNITS

When you compile a standalone subroutine in HPBB the compiler needs to know certain information to succeed in compiling COMMON areas. As in a subroutine no length or dimension information is available a dummy main program containing only the COMMON definition is required in front of the subroutine.

Example:

```
OPTION SUBPROGRAM NONEWCOM
COPTION SEGMENT="DUMMY"
COM String$[512],SHORT INTEGER K(10,15)
END
SUB Pgma1
COPTION SEGMENT="PGMA1"
COM String$,K(*,*)
.
SUBEND
```

There are two ways to do this, either merge the dummy main in front of your subroutine or include it in your HP260 source too. That works because the LOAD SUB only loads the real subroutine into your workspace.

The only difference then to be changed manually is that the HP260 does not allow you to specify the number of dimensions of an array in the subprogram, there you write K(*) only whereas HPBB requires an asterisk for each dimension or K() as shorthand for K(*).

## SUBUNIT SIZE

This is the most crucial point you will encounter when trying to convert an existing application. The HP3000 due to its segmentation poses more restrictions on the size of a subunit (main, subprogram, function) than the HP260. There you have the 64 KB space and you handle your own segmentation. On the HP3000 your available space is much larger, you can have a large number of subunits but the size of a subunit is restricted by the hardware to the size of a codesegment which is 32 KB.

Now interpreted code being compact by nature has a tendency to grow when it is compiled. Our experience showed us that the way programs are written on the HP260 often produces to much code to be successfully compilable.

To be compatible segment your programs really well. A good rule of thumb is that if your programs use from 50 to 60 sectors as PROG on the HP260 you have a good chance that it will fit compiled in a code segment. Of course there should not be too many space consuming statements in

that subunit like SEARCH, SORT, PACK, DETAIL LINE 1, excessive string-handling and so on.

Unfortunately there is a penalty on the HP260 for having more than necessary segments. Your performance will decrease because you probably have to use more LOAD SUBs and your programs may become more error 2 prone. The performance degradation might not be such a problem with the coming disccaching in operating system B08.0.

## DELETED STATEMENTS

The following file handling statements are no longer available in HPBB and should be avoided to be compatible. Most of them are no longer needed or make no sense under MPE.

AVAIL
BUFFER#
CATLINE / CATFILE
CHECKREAD ON/OFF
DIRECT / INDIRECT / NOUPDATE
DOOR LOCK/UNLOCK
DUPTEST
HOLE
PRINT / READ LABEL

## ASSIGN

The returncode of the ASSIGN statement is different, there is no way to implement it the same in both versions.

To be more compatible use constants for errorcodes and during the conversion process simply substitute their values.

| E.g. | HP260 | HPBB |
|---|---|---|
| Filenotfound= | 1 | 52 |
| etc. | | |

## FILE NAMES

The HP3000 filesystem allows you less freedom in choosing your filenames. First of all no distinction is made between upper and lowercase characters, so avoid a file naming strategy where names are only different in casing.

No special characters are allowed in filenames. On the HP260 some people started e.g. temporary workfiles with a special character

like $ or % in order to find these easier with CAT to purge them when they are no longer used.

To be compatible use only alphabetic characters and digits after the first letter.

It is worth mentioning that USRID can have up to three digits on the HP3000. As userspecific filenames are sometimes build with the USRID as suffix filenames can get longer than 6 characters. But this is no problem as MPE-filenames are up to 8 characters long.

However the longer USRID can cause problems if you have reserved somewhere space to store them and this space is not large enough.

To avoid any such problems the best solution is to allow for 8 character filenames in your storage considerations and make your program easily changeable to handle 6 chars or 8 after the conversion.

## VOLUME IDENTIFIERS

On the HP260 it was very important to distribute your fileload evenly across the available disc drives, so some techniques have been developed by adding the volume identifier to the filename thus having e.g. a separate disc for the databases or for work/spoolfiles This volid can be up to seven characters on the HP260 plus the comma to identify it as a volume name.

There are no volumenames on the HP3000. The filesystem itself takes care of the distribution of files across the available disc space.
It may however be of advantage to keep your strategy to logically distribute your files. Luckily there is a simple way to do this without too many changes if you translate HP260 volume identifiers to HP3000 groupnames.

What we have said about filenames, storage allocation etc applies to volumenames as well, group names can be up to 8 chars plus the additional point to separate it from the filename.

So we have an easy conversion if you put your whole application within one account.
If you want to use accountnames as well to qualify your filenames you better reserve some additional space in your newly developed system. Adding things like this later on introduces too many possibilities for new errors.

## SIZE

The SIZE function with argument -1 is no longer available in HPBB.

On the HP260 the argument -1 returns the logical recordlength of a workfile and by dividing this by 2 you get the THREAD length defined for this workfile.
HPBB does not support the notion of logical recordlength in a workfile. There is only a physical recordlength and there is no way to find the THREAD length in a workfile.
To be compatible use other methods to remember the THREAD length e.g. by assigning a value to a variable Threadlen.

## PRINT #N;END

On the HP260 you can write

```
FOR I=1 TO 10
  PRINT #1;I,END
NEXT I
```

and get a file filled with values from 1 to 10.
You can then read this file serially and get all your values back.
The EOF marks are overwritten each time.

In HPBB however you will get a file filled as well with 10 values, but after every value there is an end of file mark.
By reading it sequentially you can only access the first record, then you will get an error 59 (EOF).
Only by using direct word access you are able to read the remaining 9 values.

To be compatible replace the above program by:

```
FOR I=1 TO 10
  PRINT #1;I
NEXT I
PRINT #1;END
```

## DIRECT WORD ACCESS

The need for manual conversion may arise if you use direct word access to data files when you are using SHORT and REAL variables.
On the HP260 these used 4 and 8 bytes in a data file, on the HP3000 in a BDATA file they use 6 and 10 bytes either as REAL and DECIMAL.
So your wordpointers are no longer the same.
In HPBB now every data item has a descriptor (2 bytes), while on the HP260 REAL numbers didn't have them.

To be compatible do not use this feature, so do not program

```
READ #1,1,18;Var
```

rather use dummy variables to skip over undesired values

```
READ #1,1;Dummy$,Dummy,Var
```

This is compatible whatever data types you use.

## STORAGE REQUIREMENTS IN BDATA FILES

Length of variables in bytes

| Type | BDATA File vs | IMAGE/3000 |
|---|---|---|
| SHORT INTEGER | 4 | 2 |
| INTEGER | 6 | 4 |
| SHORT DECIMAL | 6 | 4 |
| DECIMAL | 10 | 8 |
| SHORT REAL | 6 | 4 |
| REAL | 10 | 8 |
| String | X+4 | X |

## DET

The matrix function DET is in HPBB only available with parameters.
To be compatible do not use DET without parameters.

# SCREEN AND PRINTER I/O

## THE CURSOR STATEMENT

The following CURSOR options no longer exist
IF, OF, RIF, ROF, PL, UL, PALL, UPALL
To be compatible do not use any of these options.

Hint: IF and OF is something else than IF# and OF#.

If you really need something like PALL or UPALL there is a manual workaround to achieve similar results on HP3000 terminals.

```
SUB Pall(SHORT INTEGER Line)
  CURSOR (Line)
  DISP '27"l";
SUBEND
SUB Upall
  DISP '27"m";
SUBEND
```

## EDIT

the EDIT statement is not available in HPBB.
To be compatible do not use it.

## RPOS AND CPOS

There are some problems in using RPOS and CPOS in combination with input statements.
Imagine you want to set the current linelength in a textprocessing program.

On the HP260 :

```
Line=Defaultlinelen
ON KEY#1:"LINELEN :"&VAL$(Line) GOSUB Set_line_length
LOOP
  LINPUT "";Line$
END LOOP
```

```
Set_line_length: Line=XPOS
ON KEY#1:"LINELEN :"&VAL$(Line) GOSUB Set_line_length
RETURN
```

This lets you set your current linelength easily by moving the cursor to the desired position first and pressing the softkey.

Not so in HPBB: the input statements all move the cursor to the beginning of the next line (except TINPUT with NOLF) so that CPOS is always 1.
RPOS is actual line +1, except when using NOLF option with TINPUT, then it is the actual line.

The only input statement which gives correct results is ACCEPT which is of course of no use in above example.

So the only compatible way is using a less userfriendly solution by inserting an empty line in the current position and actually asking the user for numeric input of his desired value.

Another workaround in HPBB (tested on 2392A and HP150-II)
Replace above LOOP by

```
LOOP
  DISP '27"&k1M";       ! Modify on
  ACCEPT Line$
  DISP '27"&k0M";       ! Modify off
ENDLOOP
```


In this solution CPOS returns the desired cursor position.

## LINPUT

LINPUT in HPBB reads only what you type in and not what is already in that line.
A HP260 example

```
DIM A$[80]
LINPUT "HELLO ";A$
DISP A$
```

If the user enters YOU the program displays HELLO YOU,
in HPBB it would only print YOU.
To be compatible do not use this feature of LINPUT. In the above example an easy workaround is possible, just do the string concatenation within the program.

An even more compatible solution is not using LINPUT in this way at all but doing the actual screen read with LENTER. This gives you the complete line in both implementations.

# SOFTKEYS DURING INPUT

Consider the following program:

```
ON KEY#8:"PRESS HERE" GOSUB Softkeyroutine
LOOP
  A$=B$=""
  INPUT A$,B$
  DISP A$,B$
END LOOP
STOP
Softkeyroutine : RETURN
```

As on the HP260 softkeys are real interrupts the INPUT sequence can be interrupted in any situation and returns to the same point where it came from. I.e. softkeys on the HP260 do not end the INPUT statement.

In HPBB however pressing a softkey terminates an INPUT so if you have already entered A$ and type half of the second input then decide to press a softkey the second input is lost, only A$ then has a value.

Upon RETURN INPUT is not continued where it was interrupted.

To have at least a similar behaviour situations like this in HPBB are usually coded as follows: (BBCT250 does the same)

```
ON KEY 8 GOSUB Softkeyroutine;LABEL="PRESS HERE"
LOOP
  A$,B$=""
  LOOP
    INPUT A$,B$
    EXIT IF RESPONSE > 1
  END LOOP
  DISP A$,B$
ENDLOOP
STOP
Softkeyroutine:RETURN
```

Please note that this is closer to the above example but still not exactly the same as INPUT is simply repeated from the beginning, meaning the user has to redo the whole input again.

**NOTE**

If you are printing graphic characters or enhancements to the printer your output string gets quite long if there are many escapesequences in the output stream.
However in a usual configuration a printer is configured with length 132 chars. If the string is longer an automatic CRLF is inserted making your output look different than you wish.

So make sure your system manager reconfigures the printer to your needs e.g. to 300 - 400 characters.

# USER DEFINED FUNCTION KEYS

SECTION 9

## ONLY 8 SOFTKEYS

HPBB has only 8 user definable softkeys available.
To be compatible do not develop new programs with multiple and different softkey levels.

## VIDEO ENHANCEMENTS

There are no videoenhancements possible in the softkeylabel.
On the HP260 it is common to highlight selected softkeys with some videoenhancements.
To be compatible do not use enhancements in the softkeylabel, use well recognizable characters instead e.g. an *.

## LABELTEXT LENGTH

The HP260 allows an 18 chars string (2*9) as text in a softkeylabel, the HP3000 has only 16 chars.
To be compatible use exactly 16 chars on the HP260 in the label. The key display algorithm breaks this in two parts of 8 chars each.

## REAL SOFTKEY INTERRUPTS

On the HP260 a softkey can interrupt a program after every statement or during all INPUT statements in their idle status.
I.e. pressing a softkey can take you right out of the middle of a processing sequence.
Ex.
```
ON KEY#8:"abort" GOTO Exit
LOOP
  do some print processing
END LOOP
```

This allows a user to stop the actual processing, here printing a list.

The HP3000 does not have real softkey interrupts, it traps pressed keys only in an input state. The only real interrupt is CTRL-Y or HALT which can be acted upon with the ON HALT statement.

To be compatible use no dynamic softkeys, use them only together with an input statement or WAIT. WAIT is automatically converted to

```
LOOP
  ACCEPT
END LOOP
```

to have the same effect in HPBB.

If you really need a dynamic softkey like in the above example use ON HALT as well, otherwise you have to change it manually later on. Pressing softkey 8 in the example under HPBB has absolutely no effect, the LOOP continues until end of processing is reached.

## DELETED STATEMENTS

The following statements are no longer existing and there is no one to one workaround so do not use them.

IN DATA SET DIM ALL
IN DATA SET USE ALL
IN DATA SET IN COM
IN DATA SET USE REMOTE LISTS
IN DATA SET LIST
IN DATA SET FREE
DBINFO mode 4xx (there are no more volumes)
READ Lock (a READ lock is automatically converted to a WRITE lock, thus decreasing throughput)
Check if you really need the READ lock.
X variables longer than 254 bytes
DBCLOSE mode 4
DBCREATE
DBERASE
DBMAINT
DBPASS
DBPURGE
DBRESTORE
DBSTORE
READ/WRITE DBPASSWORD
XCOPY
There are no QUERY controlnumbers and no DBINFO for them

Some of the deleted functions are now available through DBUTIL or other utilities provided by MPE.

## DBCLOSE

To be compatible do not use an empty string as the data base parameter. HPBB gives you an error.
DBCLOSE mode 4 is not available in IMAGE/3000.

## DBFIND

In HPBB DBFIND requires that the argumentvalue (KEY) be exactly of the same type as the key item in the data base, otherwise you will get a status error 53.

So you might have to add some conversion statements before passing your value to the DBFIND statement.

## DBGET/DBUPDATE

There is a compatibility problem in a DBGET/DBUPDATE sequence. The HP260 usually has an IN DATA SET active for the variables the program needs, mostly not the whole dataset and DBGET reads the IMAGE buffer into the program buffer and does implicit unpacking. Then you change the variables to their new values and do an update eventually with a different IN DATA SET list.

```
IN DATA SET "XYZ" USE A,B,SKP 5,C
DBGET (Dbas$,Dset$,.....,Buff$)
.
<update variables>
.
IN DATA SET "XYZ" USE <changed variables only>
DBUPDATE(Dbas$,Dset$,.....,Buff$)
```

This is ok on the HP260 as long as your program does not destroy the buffer variable.

In HPBB however DBGET allows you either to use the USING clause or the INTO clause but not both simultaneously. As DBGET USING does not keep an internal buffer yo: can be in trouble because when your program reaches the DBUPDATE statement you no longer have all your variables neither is an unpacked buffer available.

To be compatible do not use above technique in a DBGET/UPDATE sequence. Use DBGET without an IN DATA SET and do explicit unpacking and then for DBUPDATE an explicit PACK. This is fully compatible.

Attention: a very common source of error with PACK is to forget to SKIP to the end of the buffer after the last variable. This clears the remainder of the buffer to blank instead of keeping the old contents.

The second solution is just an enhancement to the first. There you need to lock your record from the DBGET to the DBUPDATE to get consistent results what can severely degrade your multiuser performance because the sequence can be quite long , could be even asking for user input who of course is just out having his lunch break.

It is a better technique to DBGET the record without locking but keeping it an a buffer too. Then do your changes but before the UPDATE do a reread of the record with a lock and check if it is still the original one on which you worked, if yes then update otherwise redo your computations.

## DETAIL LENGTH

Minimum record length in a detail is 4 bytes, not 2 like on the HP260.

## IN DATA SET

On the HP260 programmers mostly use DIM ALL or USE ALL for all DB statements of a certain data set without considering its cost. These two statements are no longer available in HPBB but it is not a good solution to simply replace them with a IN DATASET LIST using all variables of the set or a PACK/UNPACK doing all variables as well. There is one reason even for the HP260 not to do this: performance. It can speed up your program if you use in a certain DB statement only those variables you really need. Also DIM ALL and USE ALL are very cpu intensive operations requiring some time to execute.
IN DATA SET USE <list> is much better.
In HPBB this has a second even more important benefit: savings in compiled codespace. The fewer variables you have in an IN DATASET LIST or a PACKFMT the smaller your DB and PACK/UNPACK statements are.
The same applies to SEARCH (FIND) and SORT.
This is a point which cannot be stressed enough as the savings are really enormous.
And to help you do this IN DATASET itself generates no code at all so be really generous using it.

By the way if you are using IN DATASET LIST remember when changing your program to run under HPBB the SKIP option on the HP260 counts variables, in HPBB it counts bytes.

The dynamical relationship between IN DATA SET and DBGET/PUT/UPDATE is no longer available in HPBB.
On the HP260 you can write centralized routines for generalized database access, like:

```
Dbread: DBGET(Dbas$,Dset$,mode,S(*),Buf$,Arg)
        RETURN
```

and depending from where you call it, how your variables are set and which IN DATA SET definitions are currently active this statement can access totally different databases and datasets.
In HPBB with the USING clause this kind of flexible relationship is no longer possible as for the sake of the compiler the statement needs

to know statically which formatting list (IN DATA SET, PACKFMT) belongs to this access.

To be compatible do not use this dynamic way of database access or use it only in combination with explicit PACK/UNPACK.

## DBINFO

When converting to HPBB check all your programs containing DBINFO very carefully as the returned information is not always in the same format and coding as on the HP260.
There is no way to really be compatible. The best you can do is keep your DBINFOs in well defined places, maybe libraries or only in some special well documented programs.

## DBLOCK

The HP3000 does not automatically allow successive calls to DBLOCK without prior DBUNLOCK. To do this the program and its group must have MR capability which is something not everybody likes because it can lead into a deadlock situation. The affected processes wait forever and cannot be aborted, only a system shutdown helps.

There is however a possibility to do multiple locks within one DBLOCK even on the HP260. Have a closer look at the syntax of the PREDICATE statement, it allows for multiple predicates.

To write programs not requiring MR use multiple locks.

A small incompatibility exists when issuing a second DBLOCK with a finer granularity and the process has already locked on a higher level. E.g. the first lock has locked a complete dataset a second lock then does a PREDICATE lock on a single entry. The HP260 recognizes that the lock is already fullfilled and grants it. In IMAGE/3000 the statusword returns a 23 meaning already locked by another process.

To be compatible do not use this contruction or do a DBUNLOCK before you require the finer lock.

## PACK/UNPACK

PACK/UNPACK provide the most compatible way of packing and unpacking variables into a database buffer. You can use it in a one to one way in both implementations.

To be compatible use only PACK/UNPACK and not the USING clause.

Realize however that PACK/UNPACK is one of the really costly statements considering generated codespace when compiled.

## PREDICATE

There is a minor restriction in that the relational operator can only be >=, <= or =.

To be compatible restrict the usage to these three operators.

If you have implemented a central database locking routine you will certainly have to change it in HPBB because two features keep you from doing it the HP260 way.

- the relational operator is no longer a string expression, it is built into the syntax, so you will need a SELECT block probably.

- the expression for the itemvalue must be exactly the same type as the item itself, so you may have to add an additional parameter for the itemtype and some conversion statements before the PREDICATE.

## DATABASE STATUSARRAY

The following array indexes are valid for a statusarray defined as S(1:10) or S(10) using OPTION BASE 1

In the case of a database statuserror S(7) no longer returns the erroneous line of your program as IMAGE/3000 is no part of HPBB but rather a standalone subsystem.

The HP260 with its 16 bit address space did not use the complete array. As IMAGE/3000 uses 31 bits for recordaddresses neighbour fields in the array are also used. Fortunately the HP260 does not use these fields and leaves them 0, so there is a way to implement a compatible record address calculation scheme.

Instead of accessing the current recordpointer via S(4) use S(3)*2 ^ 16 + FNCorrect(S(4)).

(FNCorrect is the correctionroutine for negative numbers, see HP260 IMAGE manual)

The same applies to S(6): number of found records in chain
S(8): backwardspointer in chain
S(10):forwardspointer in chain

Instead of substituting the above expression for every occurence of S(4), S(6), S(8) or S(10) you better write a little function doing this calculation. Necessary parameters are the statusarray and an index with value 4,6,8 or 10.

## SEARCH/FIND

If you are using a function in the SEARCH expression which accesses the same workfile as SEARCH itself the behaviour in HPBB is different than on the HP260.

Consider the following piece of code:

```
WORKFILE IS #1
.
SEARCH USING Thread;FNSomething(#1)
.
.
DEF FNSomething(#1)
READ #1;Pointer
.
FNEND
```

The HP260 obviously keeps two different file control blocks so FIND does not disturb the user's file pointers and on the first call to the function the first record of the workfile is read too.

In HPBB this is not the case, SEARCH is reading the first pointer then calls the function which then is reading sequentially the second pointer.

Workaround: use direct read by adding another parameter to the function call thus keeping track of where you are.

```
Record=1
SEARCH USING Thread;FNSomething(#1,Record)
.
DEF FNSomething(#1,Record)
READ #1,Record;Pointer
Record=Record+1
.
FNEND
```

## SORT

Right now the MPE sortroutine does not know how to handle decimal data and therefore cannot sort it. In order to provide a workaround for the most common needs the SORT statements treats decimal sortkeys internally as strings and sorts them in sequence of their binary representation.

This works fine as long as all decimal values are >=1 or 0.
All other values will be in the wrong sequence.

## HINTS FOR SEARCH AND SORT

As both statements are very expensive in terms of compiled codespace do not use them extensively within one subunit. There are several ways to decrease their number:

- for identical SEARCH/SORT statements use a GOSUB routine

- move the complete SEARCH/SORT block out to a separate subunit.

- in the referenced IN DATASET USE statement define only the variables you really need here saving you tremendous amounts of codespace.

- if you are searching an already existing workfile use FILTER

- if you are sorting an already existing workfile use SORT ONLY

FIND in SORT/260 can be used in the same flexible way like DBGET ( see there ) in respect to IN DATASET and THREAD. Whatever is active at the time FIND/SORT is executed is used.

In HPBB however SEARCH and SORT for the sake of the compiler require well defined backreferences to their THREAD and IN DATASET statements.

To be more compatible do not use this flexibility in new programs.

## WORKFILE

The WORKFILE IS statement does not define an empty workfile per se. If a SEARCH results in no records found then there is a marker in the file saying this file was used in SEARCH but is empty. A following SORT or SORT ONLY recognizes that.

However if you do searching with your own tools and PRINT your qualifiing pointer in the workfile and you do not find any records at all this marker is missing.
A subsequent SORT then thinks it is first and sorts the whole thread-list out of the database.
A subsequent SORT ONLY however fails because it expects a valid workfile

## ERRN 239

Errornumber 239 (Workfile too small) no longe exists in HPBB, there it is replaced by 59 (EOF found).

Programs dealing with this kind of error have to be changed.
To be more compatible use a constant in your program header like:

```
Wftoosmall=239
.
IF ERRN=Wftoosmall THEN
.
```

## WORKFILE POINTERS

The only supported data types in workfiles are 32 bit INTEGERs. There is no way to program fully compatible as the HP260 does not have 32 bit integers.

To be more compatible document the occurence of workfiles well and do a manual change during the conversion.

## PAGE LENGTH

The default values for <blank top> and <blank bottom> are different, as well as for <pagelength>

| | HP260 | HP3000 |
|---|---|---|
| Page length | 66 | 60 |
| top/bottom | 2 | 0 |

Even if the conversion takes care of that it is good practice to define all values in the PAGE LENGTH statement explicitly.

## REPORTS ON THE SCREEN

If printing a report to the screen do not print beyond column 79 as the HP3000 terminal does an automatic linefeed and besides that your report might look strange, the line counting gets out of order and your program may result in an error 260 which does not appear when you output to a printer. This is especially true if you do double printing of a line to enhance it on the paper, if you print it on the screen beyond column 79 you have two identical lines there.

## DETAIL LINE 1

Use DETAIL LINE 1 only if necessary, i.e. only when break and totals have to be controlled, otherwise use DETAIL LINE 0 or PRINT.

See the HPBB programmers manual page 12-60 on hints on reducing code size for DETAIL LINE 1 and also TRIGGER BREAK

## LASTBREAK

LASTBREAK has a slightly different behaviour in HPBB.
If executed outside of a report it returns a value of -1 meaning no report is active. On the HP260 it caused an error

when executing it before a report actually started and it kept its value 10 after END REPORT processing. In HPBB it returns the value 10 only during the actual END REPORT processing.

This difference should not affect any program.

## HP260 BUG WITH PAGE TRAILER AND DETAIL LINE

On the HP260 it is possible to write a detail line in the page trailer area due to a bug. When you are using format specifications like:

IMAGE #,/.....

a detail line can be printed in a page trailer line and the trailer line is then added to the same line.

HPBB has not implemented this bug.

To be compatible do not rely on this to work any longer.
Add a line to the DETAIL LINE WITH n LINES statement because in reality you are printing n+1 lines what the HP260 does not recognize.

## ADDITIONAL SCREEN OUTPUT

It should be mentioned that in HPBB DISP and system output no longer counts as REPORT WRITER output thus making tracing and debugging easier. This difference should not affect any program.

## PRINT CONTROL FUNCTIONS

The functions LIN, TAB, PAGE and CTL have no effect when they are printed into an active form in HPBB. On the HP260 they acted in a strange way, it was not planned to use them within forms.
To be compatible do not use this functions in a form at all.

## TFNUM

TFNUM which becomes TFLD in HPBB behaves different in that in HPBB the TAB key is not recognized. So, if you are placing the cursor in inputfield 5 and the user presses TAB twice TFLD does not return 7 like on the HP260 but 5 instead.

When the user hits <RETURN> TFLD returns the number of the last input-field which was entered legally using the CURSOR function or normal field sequencing, but not via TAB.

To be compatible train your customers not to use the TAB key in a form at all.

It should be also noted that back tabbing does not work either.
If you want to allow your user to use TAB also your program must read the screen in blockmode style..
I.e. program an INPUT loop which simply gives the user the ability to enter his data values and provide an ACCEPT DATA softkey and then reread the screen using ENTER.

If you are using field by field data entry provide a FIELD BACK soft-key.

## VIDEO ENHANCEMENTS IN FORMFIELDS

When simulating graphic output to a form like barcharts you could use videoenhanced characters on the HP260 like inverse, blinking etc.
The same is possible in HPBB but not to the full extent as the enhanced characters contain escapesequences which count fully as

printed characters within the field, so very fast you will get an error 120 (output field overflow).

To be more compatible do not use DISP with enhanced characters but rather use the CURSOR statement with the enhancement options.

Instead of:

```
CURSOR OF#20
DISP RPT$("<inv.blank>",Full)&RPT$("<halfbr.blank>",Half)
```

use:

```
CURSOR OF#20,IV(Full)
CURSOR (XPOS+Full),IV(Half),HB(Half)
```

which is converted in HPBB to:

```
CURSOR OFLD(20),("I",Full)
CURSOR (,CPOS+Full),("I",Half),("H",Half)
```

This unfortunately does not completely behave like the HP260 and has to be manually changed to:

```
CURSOR OFLD(20),("I",-Full)
CURSOR (,CPOS+Full),("IH",-Half)
```

otherwise one enhancement overwrites the other.
The minus sign is for optimization and not required.

## CURSOR BEHAVIOUR

When the user reaches the end of a field in a form the cursor automatically skips to the beginning of the next field, maybe creating the impression that the input was already entered and processed. This is not the case, additional input is ignored.

This is a little area where the customer needs training to get used to this feature.

## OUTPUT BUFFERING

If you use a PRINT statement ending with a semicolon the output is buffered to allow to append more output. Remember that the effect of the semicolon is to be able to print several variables in one formfield. The buffered output is not printed before a PRINT without ending semicolon appears.

This may affect you in error situations or if you end the last output to the form with a semicolon. Then your output may not appear at all.

## SOFTKEYS DURING FORMINPUT

Pressing a softkey during an INPUT e.g. after you have already entered some characters terminates the input state without assigning the input to the variable.

**NOTE**

Do NOT have the MODIFY mode ON during input from a form.

Do NOT type or press softkeys while output into a form is in progress, or while your form is painted on the screen, this may destroy your form.

Also do NOT have the INSERT CHAR key set when a form is painted to the screen. On some terminals not supported by JOINFORM anyhow your form may be destroyed.

# ERROR HANDLING

SECTION 14

This chapter intentionally left blank.

## DELETED STATEMENTS

No longer available are the following statements:

ON / OFF DELAY
REQUEST / RELEASE
RES
SET DATE / SET TIME
SYSID$
all TASK statements
all PERFORM statements
all TIO statements
all MEDIA statements

If you are using ON/OFF DELAY only for timed INPUT, manually convert these programs and use the TINPUT statement with the TIMEOUT option instead. The RESPONSE function returns a 2 in case of a timeout.

## CHECKPGM

This is a utility which roughly estimates the amount of code generated by the compiler before you really compile it by looking at an ASCII version of your program (SAVE LIST).

The utility uses a file CODESPCE which contains constants for the amount of code most of the statements generate.
When you receive this utility from your SE these values may not be uptodate but you can easily change them using an editor.

## TESTSIZE

This is a real helpful utility for finding out statements in your program using up a lot of your precious codespace. In order to be able to use this routine your program must be already small enough to be compilable. If it is not just take out a part of the program, recompile and identify large statements.

Testsize was developed by a member of the HPBB team and is probably available through your local SE.

## DEFINITIONS, PACKFORMATS AND IN DATASET LISTS

For replacing DIM ALL and USE ALL during the conversion it is quite helpful to write a few simple routines for automatic generation of MERGE files for variable definitions, PACKFMT lists and IN DATASET lists instead of rewriting them by hand everytime.

These routines can be implemented very easily using DBINFO and formatting the returned information accordingly. This is also a good exercise to get acquainted with the DBINFO differences.

## OPTION INPUTLOOPS

If you are using softkeys do not forget to enable OPTION INPUTLOOPS in the conversion utility BBCT250.PUB.SYS. Otherwise your converted program will behave strangely.

## FILE REDIRECTION IN THE INTERPRETER

A very efficient way is available to write own tools doing more specific conversions and changes than BBCT250 is able to do. HPBB allows you to redirect its input command file to your own data file.

Let us suppose you have the following UDC:

```
CONV prog file=help.pub.<acct>
file bascom=!file
run HPBB.PUB.SYS;PARM=1;INFO="infile$=""!prog"""
```

Then your help file can contain things like:

```
COPY ALL OUTPUT TO "*L"
DISP "Conversion analysis for "+infile$
get infile$
! get rid of all the comments BBCT250 adds to the source
changeq "!** SYNTAX CHANGE" to "" in all
changeq "!** UNTRANSLATABLE" to "" in all
changeq "!** ADDED LINE" to "" in all
! add additional lines and dummy main
1 GLOBAL OPTION BASE 1,DECIMAL,SUBPROGRAM NONEWCOM
MERGE MAIN,2
101 COPTION SEGMENT="XXXXXX"
! do some very specific changes
changeq "DATE$" to "DATE$(1)" in all
changeq "! RE-STORE" to "! RESAVE" in all
```

```
changeq "Prt=8" to "Prt=888" in all
cwarnings
verify
ren
indent 5,3
resave
! find some more statements for further considerations
find "IN DATASET"
find "REQUEST"
find "UPALL"
find "239"
find "DBINFO"
find "THREAD"
find "Pointer"     ! could be your workfile pointer , replace with
                   ! INTEGER variable
! *****
! and whatever else you want to have checked
! *****
copy all output to display
exit
```

In this command file you can put all your little personal changes which are so specific to your programs that BBCT250 cannot take care of them. After careful investigation of a few of your programs you should be able to do almost all necessary manual changes via this command file giving you more time to break up your big programs into smaller segments.

Besides redirection of BASCOM HPBB allows you to redirect all its files by providing a certain PARM value.

Here are the HPBB parameter/file equations:

```
:file bascom=                                   parm=1
:file basin=                                    parm=2
:file bascom and basin                          parm=3
:file basout=                                   parm=4
:file bascom and basout                         parm=5
:file basin and  basout                         parm=6
:file bascom and basin  and basout              parm=7
```

```
BASCOM = command file, which is read when the > appears
BASIN  = data input
BASOUT = program output
```

## PERFORM FOR TRANSFER

The task of DBUNLOADing your data bases on the HP260 and doing all the TRANSFER can be rather tedious and timeconsuming. Do not even consider to do it manually step by step. Before you do any conversion at all

write some PERFORM files for the different steps of the conversion which can then run over night or over the weekend.

As an example how to do this here is a PERFORM file which selects all files from a certain volume starting with FB (financial bookkeeping) copies them to a different volume whereby PROG files are converted to DATA files and files are given a new name according to their type. Another interesting feature is that this PERFORM file runs the EDITOR in a secondary task and at the same time when it copies the files creates a batch file for the use of TRANSFER where it can be used by selecting the BATCH softkey.

The little program WAIT does nothing else than dimensioning the A$ variable and waiting for the parallel process to be in INPUT state.

```
! WAIT
DIM A$[80]
LOOP
  EXIT IF TSTAT(7)=1
END LOOP
END
```

Here is the PERFORM example:

```
REQUEST #7
SEND CONTROL HALT #7
SEND COMMAND#7,"RUN "&CHR$(34)&"EDITOR"&CHR$(34)
RUN "WAIT"
SEND INPUT#7,"S LENGTH=160"
RUN "WAIT"
SEND INPUT#7,"A"
RUN "WAIT"
:DIM PARM(8)
:SET PARM(1) TO 0
:LOOP
:SET PARM(1) TO PARM(1)+1
CATLINE PARM(1) ON ":S",A$
:EXIT IF A$[1,3]="EOD"
:IF A$[1,2]="FB" THEN
:IF A$[14;4]="PROG" OR A$[14;4]="DATA" OR A$[14;4]="FORM" THEN
:SET PARM(5) TO NUM(A$[4;1])
:SET PARM(6) TO NUM(A$[5;1])
:SET PARM(7) TO NUM(A$[6;1])
:IF A$[14;4]="PROG" THEN
LOAD A$[1;6]&":S"
B$="FBP"&CHR$(PARM(5))&CHR$(PARM(6))&CHR$(PARM(7))&":M"
SAVE B$
SEND INPUT#7,"DATA_FILE_TRANSFER "&B$&" TO FBH"&B$[4,6]
RUN "WAIT"
:ELSE
:IF A$[14;4]="DATA" THEN
B$="FBD"&CHR$(PARM(5))&CHR$(PARM(6))&CHR$(PARM(7))&":M"
:END IF
:IF A$[14;4]="FORM" THEN
```

```
B$="FBF"&CHR$(PARM(5))&CHR$(PARM(6))&CHR$(PARM(7))&":M"
:END IF
SEND INPUT#7,"ARCHIVE_TRANSFER "&B$&" TO FBH"&B$[4,6]
C$="FBH"&B$[4,6]&":S"
COPY C$ TO B$
RUN "WAIT"
:END IF
:END IF
:END IF
:END LOOP
SEND INPUT#7,"//"
RUN "WAIT"
SEND INPUT#7,"K "&CHR$(34)&"<filename>:M"&CHR$(34)&",UNN"
RUN "WAIT"
SEND INPUT#7,"E"
:END
```

# USING THE COMPILER

SECTION 16

## CLEANUP OF PROGRAMS

Very often modified programs tend to be larger than necessary as some internal table entries are kept even if the program code or variable definitions have been removed from the program. To clean that up do:

```
GET <programfile>
SAVE LIST <name>
GET <name>
RESAVE <programfile>
```

## COPTIONS

In addition to all the codespace saving techniques described in the various chapters you can of course use all the different COPTIONs to further reduce the amount of space you need.

These COPTION are not described here. For further details look into chapter 16 of the programmers manual or appendix D in the reference manual.

## COMPILED GET

There is no compilable GET in HPBB , however there is an easy workaround for something quite similar. For details see the programmers manual chapter 16.

# EXTERNAL ROUTINES

SECTION 17

## EXTERNAL NAMES

The HP3000 SEGMENTER handles only external names up to 15 chars so there is no problem for HP260 users as their names are at most 15 chars long. This is a point to remember when you invent subroutine names in HPBB, distinguish them within the first 15 chars.

## EXTERNAL DECLARATIONS

There is no way to include the necessary EXTERNAL declarations for compiled subroutines as code in your HP260 programs. You have to add them manually during the conversion or you can already include them in your HP260 programs as comments.

# BUGS AND WORKAROUNDS,PROBLEMS

SECTION 18

The following bugs are still existing in revision 1359.
If possible workarounds are provided.
If you have a newer revision parts of this chapter may no longer be true.

## ASSIGN BUG

ASSIGN currently cannot handle file equations to terminal devices.

```
10 SYSTEM "FILE X;DEV=32"   ! 32 is a 2392 or 3081A terminal
20 ASSIGN #1 to "*X"
```

does not work.

Workaround: add the following lines:

```
15 SEND OUTPUT TO "*X"
25 SEND OUTPUT TO DISPLAY
```

## DBGET COMPILER BUG

When DBGET incorrectly backreferences a THREAD statement instead of a IN DATASET statement the compiler crashes.

Workaround: correct the DBGET statement

## DECIMAL EXPONENTIATION

A statement like A=(A+1)^(A<5) crashes a compiled program with bounds violation.

Workaround: change the statement using multiplication instead of exponentiation

## JOINFORM BUG

The # IMAGE option is not recognized when DISP USING "#... is done in a form. The following DISP output in the next field of the form.

Workaround: use a single combined DISP for output.

## JOINFORM PROBLEM

When using a JOINFORM below the first window on the display the screen is jumping a lot when format mode is enabled. The results are correct but the behaviour is not acceptable for a customer.

There is no simple workaround. If possible avoid forms below the first screen window or paint your form explicitly with DISP or LDISP in your program without using format mode.

## IN DATASET BUG

A bug like in SORT exists with IN DATASET if a stringvariable used in the variablelist is passed as a parameter to the subroutine. In this case the compiled IN DATASET does not work properly.

Workaround: assign the parameter to a local variable first and use this in the variablelist.

## LABELS (BUG)

VERIFY in HPBB does not recognize if you define more than one label with the same name. When jumping to a label it uses the first one encountered. So if your program doesn't behave like you want check this bug as a possible source of failure.
The HP260 has the same behaviour except that LOAD and RUN give a warning (doubly defined label).

## MPE LOADER PROBLEM

Due to the fact that HPBB has a lot more runtime routines than other languages the MPE loader has to resolve many more externals than usual. As all entries are kept internally the available space of

32000 words is quickly exhausted resulting in LDERR 71: unable to obtain enough DL space. The loader aborts and your program is not runnable.

Workaround: put everything in your program USL even the HPBB runtime libraries so the SEGMENTER can already take care of the references during PREP time.

## REPORT WRITER BUG

Right now there is still a bug in the BEGIN REPORT statement in combination with BREAK WHEN. Unlike BASIC/260 BEGIN REPORT in HPBB checks the REPORT DESCRIPTION and if it contains
BREAK WHEN String$[1,6] CHANGES
and String$ actually is less then 6 chars long it causes an error 18.

Workaround: before executing BEGIN REPORT assign to String$ a string of at least the length asked for in the BREAK statement.

## REPORT WRITER BUG

PAGE LENGTH 0 in a compiled HPBB program is not yet working correctly, so instead of 0 use a large number.

## REPORT WRITER BUG

If a PAGE TRAILER contains something like
WITH 3*((A=0) AND (B=1)) LINES
the compiler crashes at the BEGIN REPORT statement.

Workaround: use LAND instead of AND

## SORT BUG

There is a bug still existing in compiled HPBB when you pass a string as a parameter to a function or subroutine and use this parameter as a sortkey. This does not work because the compiler does not know how long this string is at compiletime.

There is however a simple workaround: assign this parameterstring to a local variable and use that as sortkey, eventually reassign the changed value to the parameter before returning.

# SYSTEMRUN PROBLEM

When your program starts NOSUSP son processes the CTRL-Y trapping is taken away from the father and transfered to the son. If you need CTRL-Y trapping in the father process simply start a son process with no NOSUSP option after all NOSUSP processes are started. This son should immediately return. This gives CTRL-Y handling back to the father.

This is a MPE feature, only one process running on a specific terminal can have control over CTRL-Y.