Malcolm Bull
Training and Consultancy Publications

MB-Guide to

Group format errors

Malcolm Bull

MB-Guide
to
Group format errors

MB-Guide

to

Group format errors

by

Malcolm Bull

MALCOLM BULL Training and Consultancy Services

Malcolm Bull
Training and Consultancy Publications
19 Smith House Lane
BRIGHOUSE
HD6 2JY
West Yorkshire
United Kingdom

Telephone: 0484-713577

ISBN: 1 873283 00 8

Edition: 4.2
Updated: 25:09:93

P r e f a c e

The *MB-Guide to group format errors* is produced for those who need a quick introduction to the subject of GFEs on the Pick system and some ways in overcoming their effects.

This *MB-Guide* contains:

* A discussion of the way in which the physical items are held on disk. This looks at R83 generic Pick, Advanced Pick and Reality implementations.

* A general introduction to the nature of group format errors.

* A description of the causes and effects of group format errors.

* A description of the ways in which you can recover from GFEs.

The material will be of interest to operations personnel and to those who work in a systems support function.

You may find the following titles in the *MB-Guide* beginner's guide series useful in conjunction with the present volume:

File-save and file-restore
Files: monitoring and sizing
Operations and systems management
Security
System debugger

You may find the following *MB-Master* self-tuition courses of interest in conjunction with the material presented in this *MB-Guide*:

PICK1: Starting Pick
PICK2: Pick systems management

A suite of supporting software called MB-GFES is available for use in conjunction with this *MB-Guide*. This is a menu-driven system allowing you to carry out the operational activities associated with the detection of, and recovery from GFEs.

All *MB-Software* includes a TCL stacker utility, and many individual routines within the system are available directly from TCL. The *MB-Software* is particularly valuable to users of native PC Pick, native AP, and AP/DOS which do not provide such utilities. Please write or call for a leaflet giving more information, or to place an order.

This *MB-Guide* is not intended to present an exhaustive description of the subject but merely to place it in context and give the reader enough information to use the facilities and to survive.

Best use can be made of this *MB-Guide* if it is read in

conjunction with the reference literature which is provided for your system. You should amend your copy of this guide so that it accurately reflects the situation and the commands which are used on the implementation which you are using. By doing this, your *MB-Guide* will become a working document that you can use in your daily work.

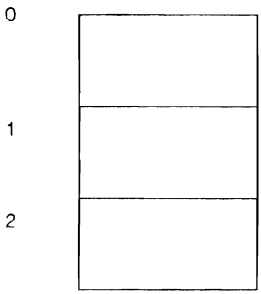
I hope that you enjoy reading and using this *MB-Guide* and the others in the series, and welcome your comments.

C o n t e n t s

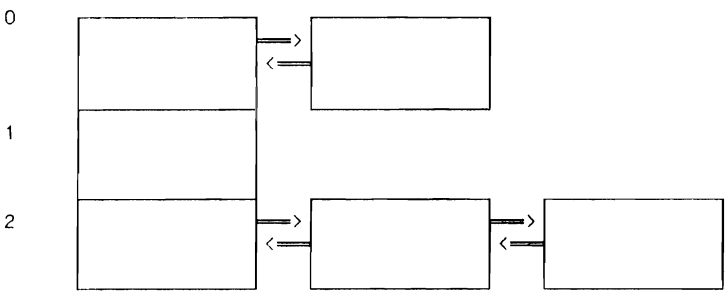
1 What is a group format error?

A *group format error* - commonly known as a *GFE* - is a serious error in the data which is recorded in a frame of the virtual memory. The presence of a GFE may result in the operating system being unable to reach and process data which lies in that frame and also in the group of frames which are linked on beyond the site of the error.

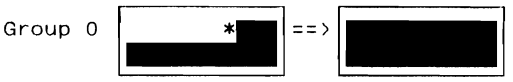
It is called a *group* format error because, within a file, the disk space is organised as a sequence of *groups*. When a new file is created, the user will specify that it is to consist of several groups of frames, like this file which was created with three groups.



I have numbered the groups in the conventional manner, starting at 0. As items are added to the file, they are added to one of the three groups according to the item-id of the particular item. As more and more items are added to the file, one or more of the groups may *overflow* into additional frame(s) of disk space. So, in time, the situation may look like this:



If we consider the items which are held in Group 0, we see that this group overflows into another frame of disk space. So if a GFE occurs at the position indicated by the * in the first frame:



then any data which lies in the shaded area *beyond* the GFE will be inaccessible by the normal Pick processing operations such as:

the Editor,
the Access processor,
the Basic READ/WRITE operations,
the COPY processor,

and others. In fact, the only processing routines which *can* access all the data in the affected frame(s) are the:

the CLEAR-FILE processor,
the DELETE-FILE processor,
the DUMP processor, and
the interactive system debugger.

Note that *only the group in which the GFE occurs* is affected. The other groups in the file and any other parts of the system are not *directly* affected. We use the word *directly* because if the affected group holds items which are used to access other parts of the system, then other areas may be *indirectly* affected. For example, if a GFE occurs on your MD, then any files whose D-pointers - the file-definition items - lie in the affected group will be inaccessible.

Note also that the error affects only the *logical* meaning of the data and it is not a permanent *physical* error which will affect that frame indefinitely.

1.1 Nature of GFEs

In order to explain the nature of GFEs, we need to know something about the physical structure of the frames and the groups which make up a file.

If we were to look at the contents of a typical frame of virtual memory storage, we might find that it looks like:

```
FID: 10136 : 0 10502 47771 0 ( 2798 : 0 2906 BA9B 0 )

  1 :OAK^15^LS/15/77^5000^25^2000^8340^_00602200^SETTEE:
 51 :, BROWN, OAK^36]20]10^DN/1/43]LS/4/65]DN/11/70^100:
101 :0^30^2000]1000^8320]8321]8312^_00384444^SETTEE, BL:
151 :ACK, ASH^^DN/6/81^1000^30^1000^8320^_00511200^DESK:
201 :, GREY, ASH^16]32^LS/17/1]LS/17/2^5600^30^1000]300:
251 :0]2000^8320]8302^_003C4500^CHAIR, RED, LAMINATE^18:
301 :^LN/3/29^2500^60^1000^8345^_00409000^SIDEBOARD, YE:
351 :LLOW, ASH^99^DN/14/70^13000^15^1000^8345^_00415656:
401 :^SETTEE, BLUE-GREEN, MAPLE^6^DN/56/40^1000^30^2000:
451 :^8306^_00361234^CHAIR, AUBERGINE, MAHOGANY^7^DN/5/:
```

This display was produced by means of the:

DUMP 10136

command on a PC implementation of Pick R83 version 3.1, and shows that the frames here are 512 bytes in length (12 bytes for the control information which we discuss later plus 500 bytes for the data). On some systems, the frames are 1024 bytes long (24 control bytes plus 1000 data bytes), or 2048 bytes long (48 plus 2000 data bytes) or 4096 bytes long (96 plus 4000 data bytes).

By inspection, this frame would seem to be somewhere in the *middle* of the group since the data starts with what looks like the end of one record, and it ends with what appears to be the first part of another record. The numbers 1 to 451 show the position of the first character on that line within the data area of the frame. Thus, the letter O of OAK is in byte 1 of the data area, the character ^ (immediately before the 8306) is in byte 451 of the data area, and the final / character is in byte 500 of the data area. Strictly speaking, we should add 12 to these figures to allow for the 12 control bytes which are held at the front of each frame.

As this illustrates, each frame of disk space consists of:

1) The control or linkage bytes. These are first 12 bytes of each frame and enable that frame to be associated with its overflow frames.

In this instance, the actual linkage bytes are not displayed, but their contents are shown on the first line:

```
FID: 10136 : 0 10502 47771 0 ( 2798 : 0 2906 BA9B 0 )
```

This tells us that:

+ This shows the contents of frame 10136 (that is, frame 2798 in the hexadecimal notation);

+ That the data in this frame is continued from that in frame 47771 (BA9B). This information is known as the *backward link*;

+ That this data in this frame overflows into frame 10502 (2906). This information is known as the *forward link*.

This information is held in the first 12 bytes (of a 512-byte frame), the first 24 bytes (of a 1024-byte frame), and so on.

2) The data items.

The general format of a physical item is:

```
ccccciii^ddd^eee^fff^ggg^_
```

where *cccc* is the four-byte item length count, *iii* is the item-id, *ddd, eee, fff* and so on are the data attributes of the item and the character shown here as _ is the end-of-item indicator.

In this illustration, familiarity with the data contents allows us to identify that the section:

```
101 :0^30^2000]1000^8320]8321]8312^_00384444^SETTEE, BL:
151 :ACK, ASH^^DN/6/81^1000^30^1000^8320^_00511200^DESK:
```

contains the physical item:

```
00384444^SETTEE, BLACK, ASH^^DN/6/81^1000^30^1000^8320^_
```

The *item length counter* (0038 in this instance) is maintained by the operating system whenever the item is written to disk and tells the operating system the exact physical length of this item. The item length counter is held as a hexadecimal number (0038 hexadecimal is the equivalent of decimal 56, in this instance). On some implementations, this may not be readable.

The item length counter field gives the total length of:

+ All the data attributes and field separators, plus
+ The item-id, plus
+ The *end-of-item marker*, plus
+ The four-byte item length counter field itself.

The data in this particular example indicates that:

+ The item-id is 4444
+ Attribute 1 contains SETTEE, BLACK, ASH
+ Attribute 2 is null
+ Attribute 3 contains DN/6/81
+ Attribute 4 contains 1000
+ Attribute 5 contains 30
+ Attribute 6 contains 1000
+ Attribute 7 contains 8320

3) The *end-of-group marker*. This is a final marker indicating that there is no further data belonging to this group.

This is one of the system separator characters. On most implementations, as in the illustration shown below, this is the segment mark (character 255), on other implementations, it may be the attribute mark (character 254).

If we were to look at the *last* frame of the group, like this:

```
      FID: 23981 : 0 0 18131 0 ( 5DAD : 0 0 46D3 0 )

  1 :^_00584200^SETTEE, GOLD, ASH^55]10^XN/19/35]LS/6/6:
 51 :^1000^30^1000]2000]3000]4000^8343]8345^_004C8117^S:
101 :IDEBOARD, GREY, OAK^0]0^MN/5/66]LN/91/55^13000^15^:
151 :1000^8300]8339^_005B2936^SETTEE, GREY, OAK^36]20]1:
201 :0^DN/1/2]LS/4/1]DN/11/1^700^30^2000]1000^8344]8316:
251 :]8338^_003D1000^DESK, GREEN-BLUE, ASH^8^MN/17/81^5:
301 :600^30^2000^8346^__.............................:
351 :................................................:
401 :................................................:
451 :................................................:
```

we would observe several points:

* This is frame 23981 and it is a continuation of the data in frame 18131 (as shown by the backward link), but the data is not continued into any further frames (the forward link is zero).

* The final item in the file is 003D (hexadecimal bytes) in length with the item id 1000, and the data contents are:

```
DESK, GREEN-BLUE, ASH^8^MN/17/81^5 600^30^2000^8346^
```

* Of the two final _ characters, the first is the *end-of-item* marker, and the second is the *end-of-group* marker.

* The rest of the frame (shown here as dots) is of no concern to our file and is just what happened to be in this frame when it was last used prior to being seized for use by our file.

The operating system may sense a group format error in any of several situations:

* It may be that there is an invalid character in the linkage information which results in the operating system being unable to follow up any linked frames.

* It may be that there is any invalid character in the item length count field, so that the processor does not know the expected length of the item.

* It may be that the final end-of-item marker is inconsistent with the item length count.

* It may be that the final end-of-group marker is an unacceptable character.

1.2 Item format on Advanced Pick

On Advanced Pick, the items are held in a slightly different manner.

Let us look at the DUMP of a typical frame on Advanced Pick. We have shown only the first few bytes in this instance, the remainder of the frame is filled with rubbish not related to our file.

```
000 :........AAAA^this is item aaaa and it will be held:
050 : directly within the file space.^_........BBBB^002:
100 :268^__.....................................
```

Here we see:

* An 8-byte field: this is the item-length field and an indication of the nature of the item (whether it is direct or an indirect pointer item) are held within a control string of 8 bytes before each item-id.

* The end-of-item marker (^_) and the end-of-data marker (^__) are used as before.

* Items such as AAAA of which the data is 847 bytes in length or shorter are held in the primary file space (as on previous versions). This figure of 847 bytes relates purely to the length of data in the item and excludes the field-length count, the item-id and the final end-of-item marker.

  If we look at this 8-byte field as a hexadecimal string, we find that it is of the form:

  00000000yyxxz000

  where xxyy is the hexadecimal length of the item (this represents the 8-byte field itself, plus the length of the item-id, plus one attribute-mark, plus the length of the true data and field separators, plus 1 for the final attribute mark); note the way in which the xxyy field is held. We should also mention that this value is held as an odd number (1 being added, if necessary). The z in the 13th digit is either 0 or 1 and indicates that this item is held in the file spacer; a 0 denotes that the item has not changed since the last full file-save, a 1 denotes that it has changed.

* Longer items (848 bytes or more) such as BBBB are held indirectly, with the actual data being written directly to one or more frames of disk and only a pointer-item is held in the file space. If we dump the frame to which the pointer for item BBBB directs us (hex 2268 = decimal 8888), we see the full item:

  For items held in this manner, the 8-byte field would be:

  000000001100z000

  with the z being 8 (if the item has not changed since the last full file-save) or 9 (if it has changed), both

indicating that the item is held indirectly.

```
          fid: 8888 : 0 0 0 0 ( 2268 : 0 0 0 0 )

000 :_BBBB^this is item BBBB and, since it exceeds 847 :
050 :bytes, it is held as a pointer in the file space. :
100 :   This pointer points to the frame where the true:
150 : data is held.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
200 :xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
:
850 :xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
900 :xxxxxxxxxxxxxxxxxxxxx^__.....................:
950 :.............................................:
```

File-definition items - D-pointers - are held in this indirect manner, with the z indicator being A or B. Object programs are held in this indirect manner, with the z indicator being C or D.

Incidentally, after a full file-save has been performed, the system resets the z indicators to the clean states of 0, 8, A and C during the process of clearing the *dirty bits*.

The forward/backward linakge information is held in the *final* 24 bytes of the frame (not at the front, as is the case on generic Pick systems). This example was produced on an Advanced Pick system which uses 1024-byte frames. The effect is similar on those systems which uses 2048-byte frames.

As the fragements below indicate, the output produced by the ITEM and GROUP commands has been modified and shows a code:

F	if the item is a D-pointer,
B	if the item is Basic object code,
P	if the item is held indirectly,

or null if the item is held directly.

```
:GROUP DICT PROGRAMS

 6509.0000  F  0014  PROGRAMS
 6509.0014  F  001A  ORIGINALS
 6509.002E  F  0014  SUBROUTINES
 6509.0050  B  0024  CALC.MONTH
 6509.0074  B  0024  PRNT.SLIPS
 6509.0098  B  002A  SALARY.MAINT
 6509.00C2  B  0022  SALARY.SAVE
 ──┬── ──┬── ─┬─ ──┬──
   │     │    │    └ item size (hex)
   │     │    └ type code
   │     └ starting byte address
   └ frame address
```

```
:ITEM PROGRAMS INDEX.SUB

INDEX.SUB
```

```
3563.0000      0178  GET.LABEL
3563.0178      0124  AMEND.WKBK.STOCK
3563.029C  P   001E  MB.PAGE.COUNT
3563.02BA  P   001E  SCROLL.SELECT
3563.02D8  P   001A  INDEX.SUB
3563.02F2  P   001E  MB.TIDY.INDEX
3563.0310  P   001E  CLEAR.SCREEN
3563.032E      0070  QUOTATION.SUB.2
3563.039E      0292  ADD.CONTACT
```

On Advanced Pick, GFEs are handled in exactly the same manner as on R83 systems.

1.3 Item format on Reality

On early releases of the Reality operating system, the physical items were held as described for R83 generic Pick. Later releases, however, hold large items in an indirect manner similar to that of Advanced Pick.

Each frame of disk space consists of:

* The linkage bytes. These are first few bytes of each frame and enable that frame to be associated with its overflow frames.

  These are the first 24 bytes of the 1024-byte frame.

* The data items.

  The general format of a physical item is:

  ```
  xxxxxxxxiii^ddd^eee^fff^ggg^_yyyy_
  ```

  where *xxxxxxxx* is an eight-byte control field, *iii* is the item-id, *ddd*, *eee*, *fff* and so on are the data attributes of the item and the character shown here as ^ is the attribute-mark (hexadecimal FE) and the _ is the segment-mark (hexadecimal FF). The string *yyyy* is the padding, as described below.

* Each item is padded out with a string of null characters (hexadecimal 00) so that the total length of the item (including the eight-byte control string, the end-of-item marker and the final segment-mark) is a multiple of eight bytes. This padding is represented by the string *yyyy* above.

* The eight-byte control field is maintained by the system whenever the item is written to disk and contains the following information:

  * An indication of whether the item is held directly or indirectly, as discussed below.

  * The date when the item was last written to the file. This field can be interrogated by the NU operand of the A processing code in an ENGLISH attribute definition.

* The physical length of this item.

The structure of this eight-byte field is:

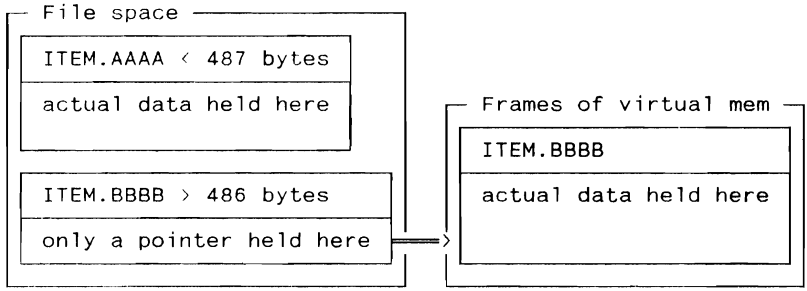
00 00 dd dd 00 00 xx xx

for a direct item (in which dd is the hexadecimal date when the item was written and xx is the hexadecimal length of the item), and for an indirect item:

00 01 ff ff ff ff xx xx

(in which ff is the hexadecimal frame number at which the true data is stored, and xx is the hexadecimal length of this pointer-item as it is held in the file space).

* If an item is greater than 486 bytes in length then it is not held in the *direct* manner in the file space (as it was on early versions of Reality and still on generic R83 Pick). Instead, the actual data is written into one or more dedicated frames of virtual memory, away from the file itself, and the file contains a much smaller item. The eight-byte control field now indicates that this item is held *indirectly*. This is illustrated in the diagram below.



* The *end-of-data marker*, or end-of-group marker, indicating that there is no further data belonging to this group in the frame. This is the segment mark.

In all cases, the physical item is padded out so that it is a multiple of 8 bytes in length. This improves the efficiency of the system when searching, reading and writing the data. This can clearly be seen in the illutsration below where we see a sequence of items of various lengths: item A holds one byte of data (A); item B holds two bytes (BB), and so on.

The physical layout can be seen if we dump a frame which which contains some of the items of our file. For example, the command:

DUMP 518401 (X

will display the contents of the frame 518401 which the

GROUP or ITEM command may have told us holds some of the items of our file. The left section shows the hexadecimal data, and the left section shows the character format.

Typical output might look like this:

```
FID:  518401 :  0      0      0   0  ( 7E901 :  0     0     0  0 )

0000  00000000 00000000 00000000 00000000      0 :................:
0010  0007E900 00800000 00002493 0000000F     16 :...........$.....:
0020  41FE41FE FF0000FF 00002493 0000000F     32 :A^A^_._.._$.....:
0030  42FE4242 FEFF00FF 00002493 0000000F     48 :B^BB^_._.._$.....:
0040  43FE4343 43FEFFFF 00002493 0000000F     64 :C^CCC^__.._$.....:
0050  44FE4444 4444FEFF 00002493 00000017     80 :D^DDDD^_.._$.....:
0060  45FE4545 454545FE FF000000 000000FF     96 :E^EEEEE^_......._:
0070  00002493 00000017 46FE4646 46464646    112 :..$.....F^FFFFFF:
0080  FEFF0000 000000FF 00002493 00000017    128 :^_......_.._$.....:
0090  47FE4747 47474747 47FEFF00 000000FF    144 :G^GGGGGGG^_...._:
00A0  00002493 00000017 48FE4848 48484848    160 :..$.....H^HHHHHH:
00B0  4848FEFF 000000FF 00002493 00000017    176 :HH^_..._.._$.....:
00C0  49FE4949 49494949 494949FE FF0000FF    192 :I^IIIIIIIII^_.._:
00D0  00002493 00000017 4AFE4A4A 4A4A4A4A    208 :..$.....J^JJJJJJ:
00E0  4A4A4A4A FEFF00FF 00002493 00000017    224 :JJJJ^_._.._$.....:
00F0  4BFE4B4B 4B4B4B4B 4B4B4B4B 4BFEFFFF    240 :K^KKKKKKKKKK^__:
0100  00002493 00000017 4CFE4C4C 4C4C4C4C    256 :..$.....L^LLLLLL:
0110  4C4C4C4C 4C4CFEFF 00002493 0000001F    272 :LLLLLL^_..$.....:
0120  4DFE4D4D 4D4D4D4D 4D4D4D4D 4D4D4DFE    288 :M^MMMMMMMMMMMM^:
0130  FF000000 000000FF 00002493 0000001F    304 :_.......__..$.....:
0140  4EFE4E4E 4E4E4E4E 4E4E4E4E 4E4E4E4E    320 :N^NNNNNNNNNNNN:
0150  FEFF0000 000000FF 00002493 0000001F    336 :^_....._..$.....:
0160  4FFE4F4F 4F4F4F4F 4F4F4F4F 4F4F4F4F    352 :O^OOOOOOOOOOOO:
0170  4FFEFF00 000000FF 00002493 0000001F    368 :O^_...._..$.....:
0180  50FE5050 50505050 50505050 50505050    384 :P^PPPPPPPPPPPP:
0190  5050FEFF 000000FF 00002493 0000001F    400 :PP^_..._..$.....:
```

The top line shows the FID (and any forward and backward linkage fields); the data in parentheses is the hexadecimal equivalent of the decimal information.

To illustrate the difference between direct and indirect items, let us consider a file which contains just two items. the item with the item-id DIRECT contains just the letters of the alphabet; item INDIRECT contains 20 attributes each containing the letters of the alphabet. The first output is a fragment of the display produced by the command:

```
DUMP 518401 (X
```

and shows the data which is actualy held on the file. Here, the eight-byte control field preceding item INDIRECT tells us that although the item here is only 23 (hexadecimal 17) bytes long, the data of item INDIRECT is actualy held in frame 518373 (shown here as hexadecimal 7E8E5). The second set of output was produced by the command:

```
DUMP 518373 (X
```

```
FID:  518401 :  0     0     0    0  ( 7E901 :  0     0     0  0 )

0000  00000000 00000000 00000000 00000000      0 :................:
0010  0007E900 00800000 00002493 0000002F     16 :..........$..../:
0020  44495245 4354FE41 42434445 4648494A     32 :DIRECT^ABCDEFHIJ:
0030  4B4C4D4E 4F505152 53545556 5758595A     48 :KLMNOPQRSTUVWXYZ:
0040  FEFF0000 000000FF 00010007 E8E50017     64 :^_.....^_.......:
0050  494E4449 52454354 FEFF0000 000000FF     80 :INDIRECT^_.....^_:
0060  FF000000 00000000 00000000 00000000     96 :_...............:
0070  00000000 00000000 00000000 00000000    112 :................:
0080  00000000 00000000 00000000 00000000    128 :................:
```

```
FID:  518373 :  0     0     0    0  ( 7E8E5 :  0     0     0  0 )

0000  00000000 00000000 00000000 00000000      0 :................:
0010  0007E901 00840000 00800222 0007E8E5     16 :..........".....:
0020  00812493 00000001 494E4449 52454354     32 :..$.....INDIRECT:
0030  FE414243 44454648 494A4B4C 4D4E4F50     48 :^ABCDEFHIJKLMNOP:
0040  51525354 55565758 595AFE41 42434445     64 :QRSTUVWXYZ^ABCDE:
0050  4648494A 4B4C4D4E 4F505152 53545556     80 :FHIJKLMNOPQRSTUV:
0060  5758595A FE414243 44454648 494A4B4C     96 :WXYZ^ABCDEFHIJKL:
0070  4D4E4F50 51525354 55565758 595AFE41    112 :MNOPQRSTUVWXYZ^A:
0080  42434445 4648494A 4B4C4D4E 4F505152    128 :BCDEFHIJKLMNOPQR:
0090  53545556 5758595A FE414243 44454648    144 :STUVWXYZ^ABCDEFH:
00A0  494A4B4C 4D4E4F50 51525354 55565758    160 :IJKLMNOPQRSTUVWX:
00B0  595AFE41 42434445 4648494A 4B4C4D4E    176 :YZ^ABCDEFHIJKLMN:
00C0  4F505152 53545556 5758595A FE414243    192 :OPQRSTUVWXYZ^ABC:
00D0  44454648 494A4B4C 4D4E4F50 51525354    208 :DEFHIJKLMNOPQRST:
00E0  55565758 595AFE41 42434445 4648494A    224 :UVWXYZ^ABCDEFHIJ:
00F0  4B4C4D4E 4F505152 53545556 5758595A    240 :KLMNOPQRSTUVWXYZ:
0100  FE414243 44454648 494A4B4C 4D4E4F50    256 :^ABCDEFHIJKLMNOP:
0110  51525354 55565758 595AFE41 42434445    272 :QRSTUVWXYZ^ABCDE:
0120  4648494A 4B4C4D4E 4F505152 53545556    288 :FHIJKLMNOPQRSTUV:
0130  5758595A FE414243 44454648 494A4B4C    304 :WXYZ^ABCDEFHIJKL:
0140  4D4E4F50 51525354 55565758 595AFE41    320 :MNOPQRSTUVWXYZ^A:
0150  42434445 4648494A 4B4C4D4E 4F505152    336 :BCDEFHIJKLMNOPQR:
0160  53545556 5758595A FE414243 44454648    352 :STUVWXYZ^ABCDEFH:
0170  494A4B4C 4D4E4F50 51525354 55565758    368 :IJKLMNOPQRSTUVWX:
0180  595AFE41 42434445 4648494A 4B4C4D4E    384 :YZ^ABCDEFHIJKLMN:
0190  4F505152 53545556 5758595A FE414243    400 :OPQRSTUVWXYZ^ABC:
01A0  44454648 494A4B4C 4D4E4F50 51525354    416 :DEFHIJKLMNOPQRST:
01B0  55565758 595AFE41 42434445 4648494A    432 :UVWXYZ^ABCDEFHIJ:
01C0  4B4C4D4E 4F505152 53545556 5758595A    448 :KLMNOPQRSTUVWXYZ:
01D0  FE414243 44454648 494A4B4C 4D4E4F50    464 :^ABCDEFHIJKLMNOP:
01E0  51525354 55565758 595AFE41 42434445    480 :QRSTUVWXYZ^ABCDE:
01F0  4648494A 4B4C4D4E 4F505152 53545556    496 :FHIJKLMNOPQRSTUV:
0200  5758595A FE414243 44454648 494A4B4C    512 :WXYZ^ABCDEFHIJKL:
0210  4D4E4F50 51525354 55565758 595AFE41    528 :MNOPQRSTUVWXYZ^A:
0220  42434445 4648494A 4B4C4D4E 4F505152    544 :BCDEFHIJKLMNOPQR:
0230  53545556 5758595A FEFF0000 00000000    560 :STUVWXYZ^_......:
0240  00000000 00000000 00000000 00000000    576 :................:
```

Note that the options must be preceded by the open parenthesis on the Reality DUMP command.

1.4 Causes of GFEs

Any of these situations may be caused by a mechanical problem, such as:

* A power surge or power failure or someone switching the machine off as the operating system was writing a frame back to disk;

* Someone performing a coldstart operation (or rebooting the machine) as the operating system was writing a frame back to disk;

* A hardware problem;

* Someone physically amending the modulo and/or separation in an account-definition item or a file-definition item (D-pointer);

* Someone using the interactive system debugger and patching a frame incorrectly;

* A software problem, particularly in software which utilises non-standard Assembler language routines.

Any GFE is a cause for concern. If your system appears to be prone to GFEs, then you should call in the support staff from your hardware supplier and/or your software supplier.

1.5 What a GFE is not!

A GFE must not be confused with the harmless *frame fault* which is a term used to describe the situation in which a process is interrupted momentarily as a new frame of data is brought in from disk.

Nor should a GFE be confused with the more serious *disk error* which may be the result of permanent physical damage to the disk. Such a disk error renders a frame (or frames) inaccessible to the Pick processors. If the operating system fails to read a part of the disk it will retry ten times; if all the rereads fail, the operating system will display a

```
&
```

sign on the terminal whose process encountered the fault. A string of such:

```
&&&&&&&&&&
```

characters indicates that the system is continuing to attempt to read the faulty sector(s). Such faults will be logged on the SYSTEM-ERRORS (or SYS-ERRS) file.

These disk errors may go away when the system is next reloaded. At such times, the operating scans the entire disk and ignores any erroneous frames.

Until then, care should be taken to ensure that the offending frame is not encountered during ordinary

processing. This can be done by *black-holing* the frame(s). Black-holing is the name given to the task in which the user deliberately makes a frame inaccessible to the operating system. This can be done by:

1) Locating the file in which the faulty frame lies.

2) Dumping as much of the file data to backing storage (or copying the data to another file).

3) Using the Editor to delete the D-pointer by which the operating system locates the file. This may be the file-definition item on the MD or the data-level identifier on the DICT section of the file.

   Since the file is not deleted formally by the DELETE-FILE command, the frames are not returned to the POVF table and are therefore considered to be in use. The space will therefore be ignored during any subsequent need for frames (by the CREATE-FILE command or for overflow).

4) Creating the file anew and restoring the data.

Such *lost* frames will not be returned to the POVF table and will only be recovered during a subsequent file-restore.

2 How is a GFE detected?

Whenever a GFE is detected, the operating system will protest the fact with a message such as:

```
     GFE HANDLER INVOKED
```

or:

```
     GROUP FORMAT ERROR AT .xxxx
```

where *xxxx* is the hexadecimal address of the frame which contains the error. But, since a GFE is sensed by the operating when it needs to access data in the group which contains the error, a GFE may lie unnoticed for some time before it is detected.

The file statistics records produced on the STAT-FILE by the file-save and account-save procedures includes an attribute which indicates the number of GFE on the file. This can be monitored by inspecting the STAT-FILE report and checking the GFE column. This indicates which section (DICT or data) of which file contains a GFE.

If a file is suspected of having a GFE, then you must scan all the groups of the file in order to determine where the GFE(s) occur(s). This is most simply done by means of a command such as:

```
     COUNT filename
```

or:

```
     COUNT DICT filename
```

which causes the operating system to proceed through all the groups of the file. When a GFE is detected, the process will stop with the message shown above.

The COUNT verb may not reveal a GFE which is caused by an invalid end-of-item marker. The use of a command such as:

```
     CT filename *
```

or:

```
     COPY filename * (T
```

will reveal all types of GFE. As before, the process will stop with the message shown above when a GFE is detected.

You may scan *all* the files on a specific account for GFEs by performing a *dummy* account-save by means of the command:

```
     SAVE (DFI
```

or on Pick Release 3.1:

```
     CHECK-ACCOUNT
```

or on McDonnell Douglas systems:

```
     SAVE SYSTEM * (DFKI
```

entering the account-name when invited. As before, the process will stop with the message shown above when a GFE is

detected.

You may scan *all* the files on the system for GFEs by performing a *dummy* file-save by means of the command:

```
SAVE (DF
```

or on Pick Release 3.1:

```
CHECK-FILES
```

or on McDonnell Douglas systems:

```
SAVE SYSTEM * (DFK
```

As before, the process will stop with the message shown above when a GFE is detected.

Since the ABS frames of virtual memory are only accessed when needed, a GFE in the ABS section will only be reported when the offending frame is used. The:

```
VERIFY-SYSTEM
```

command can be used to reveal GFEs in this section. Only the GFE(s) reported by this process are of interest; any other reported mismatches may be a result of special user-exits and software such as Jet, Accu/Plot and CompuSheet+ being loaded on the system.

3 What to do if you detect a GFE

When the operating system detects a GFE, it will interrupt the process and invoke the GFE handler. This will first display a message such as:

```
GFE HANDLER INVOKED
```

or:

```
GROUP FORMAT ERROR AT .xxxx
```

where *xxxx* is the hexadecimal FID of the frame which contains the error.

In some circumstances, there may occur a *soft group format error*. This is a GFE which may be detected at one moment but may not exist later; if you repeat the same process a moment later, there will be no GFE detected. Although this situation is not very common, it may be a consequence of the fact that you were attempting to access data whilst someone else was in the middle of writing away the same data. Soft GFEs normally *go away by themselves*, but in this present *MB-Guide*, we are concerned with the more serious and more durable *hard* group format errors.

What happens when a GFE is detected by the operating system, depends upon which implementation you are using. The *GFE handler* may continue with the processing, ignoring any items in the affected area, making the best of a bad job, or it may give you the opportunity to:

* Fix or Kill the error. Because of the situation, and the fact that the GFE handler cannot interpret the data completely accurately, either of these two choices will almost certainly lose *some* of your data.

  Fixing generally results in your losing the data beyond the offending point, whilst data up to (and possibly including) the offending item will not be lost.

  Killing generally results in your losing all the data in the group where the error occurs.

* Enter the interactive system debugger.

* Abandon the process and return to TCL.

On some implementations, the GFE handler may prompt you for the action to be taken:

```
Enter F(ix) / K(ill) / D(ebug) / E(nd)
```

or it may simply keep repeating:

```
GROUP FORMAT ERROR AT .xxxx
```

and expect you to enter your command at that point. In a later section, we shall look at the various options which are available at this point.

Unless you are sure that it is a hard GFE, it is not a good

plan to fix or kill the GFE without thinking about the problem. So, if in doubt, you should abandon the process (enter the letter E to end the process) and return to TCL.

On Advanced Pick systems, a GFE is reported by the message:

*** GFE encountered
F=fix item/T=truncate group & quit/Q=quit/<CR>=go to debugger

and the response, F or T or Q or <Return>, determines the action taken. As above, if you are in doubt, you should enter the letter Q to end the process and return to TCL.

Advanced Pick allows GFEs to be resolved automatically, without such user intervention. This is achieved by placing an M in the OPTIONS field of the USERS item for each user who is to have automatic fixing.

When a GFE is detected and automatic fixing has been requested, the system will fix the error, if this is possible. If the error cannot be fixed, then the offending data will be dumped to the ERRORS,GFE file where it can be used to reconstruct the damaged data manually.

3.1 A GFE in the ABS frames

The ABS section of virtual memory is the set of frames used by the operating system itself. This typically extends from frame 0 up to frame 1024. The upper limit is shown under the heading PCB0 on the output produced by the:

```
WHAT
```

command.

Since the ABS frames are only accessed when needed, a GFE will only be reported when the offending frame is used. The:

```
VERIFY-SYSTEM
```

command can be used to reveal GFEs in this section. Only the GFE(s) reported by this process are of interest; any other reported mismatches may be a result of special user-exits and software such as Jet, Accu/Plot and CompuSheet+ being loaded on the system.

If a GFE occurs in the ABS section of virtual memory, the ABS frames should be reloaded from the original Pick system diskettes/tapes. As described in the *MB-Guide to operations and systems management,* this is done by rebooting the machine with the system diskette/tape on-line and then selecting the:

```
A)BS only
```

option. On systems such as McDonnell Douglas, the frames can be loaded from the ABS section of a file-save tape by means of the

```
:ABSLOAD
```

command.

The ABS frames should not be reloaded whilst there are other users on the system.

Furthermore, you should also take care to de-install the standard Pickware and re-install it (and any special user exits) after reloading the ABS frames.

3.2 Can we recover the data?

Because the contents of the ABS section are relatively static, the restoration of the ABS frames is a fairly straightforward matter. But if a GFE occurs amongst the users' files, there is a greater problem since the files will almost certainly have changed since the last file-save or account-save was performed.

Fixing a GFE - no matter whether you do it by way of the GFE handler or by hand - will almost certainly result in the loss of some data. So, before any attempt is made to fix a GFE, you should ask the following questions:

? What caused the error? If the cause can be identified, then the appropriate action should be taken to avoid a repetition in future. If it is a mechanical problem, then the system engineer should be notified.

? Are any other users involved and are they likely to be affected if we fix the error? This may be the case if the GFE was a result of amending the modulo and separation of a file and thereby overwriting someone else's data. All such users should be notified.

? How much data will be lost if we fix the error? If there is only a small amount of data loss, then it may be possible to reconstruct the data by manual means such as using the Editor or rerunning some processing routine(s).

? Is it possible to recover the data by any other means? It may be that there is a perfectly good back-up copy of the data, such as a file-save, account-save or T-DUMP.

If there is such a back-up copy, then you should:

1) Clear all the data from the offending file, using the appropriate CLEAR-FILE command.

2) Restore all the data from the back-up copy. This may be achieved by performing a selective restore from a file-save diskette/tape or an account-save diskette/tape or recovery from a T-DUMP diskette/tape. This topic is discussed in the *MB-Guide to file-save and file-restore.*

3) Check the file to ensure that there are no residual GFEs,

and abandon any further attempts to resolve the problem.

In almost every case, your first reaction should be to try to reinstate the file and recover the data by this last method.

3.3 If you do not have a back-up copy of the file

If you do not have a back-up copy which from which you can reinstate the file, then you must prepare to fix the GFE.

Before taking any action to fix a GFE, it is a good plan to print a copy of the data held in the offending frame(s). This is most simply done by means of commands of the form:

DUMP *fid*
which will display the contents of the single frame *fid*. Here, *fid* is the FID of the frame at the start of the group which is to be dumped. This may be expressed as a decimal number:

```
DUMP 12345
```

or as a *hexadecimal* number (this is the form in which the GFE handler displays the FID):

```
DUMP .3039
```

The prefix . identifies this as a hexadecimal number.

Typical output from the DUMP command might look like this:

```
FID:  12345 :  0  0  0  0          ( 3039 : 0  0  0 0 )

  1 :003D1000^DESK, GREEN-BLUE, ASH^8^LS/17/81^5600^30^:
 51 :2000^8205^_003C2000^SETTEE, YELLOW, OAK^18^DN/1/69:
101 :^10000^30^1000^8176^_003D3000^SIDEBOARD, BLUE, ASH:
151 :^58^MN/5/56^13000^15^2000^8178^_003A4000^DESK, BLA:
201 :CK, MAPLE^68^LS/7/87^5600^30^1000^8173^_003B5000^S:
251 :ETTEE, ORANGE, ASH^24^DN/19/3^1000^30^1000^8180^__:
301 :.................................................:
351 :.................................................:
401 :.................................................:
451 :.................................................:
```

The top line shows the FID (and any forward and backward linkage fields); the data in parentheses is the hexadecimal equivalent of the decimal FIDs. We saw examples of such DUMP output earlier. Here are some more examples:

DUMP *fid* X
will dump the contents of the frame, displaying the contents in both character and hexadecimal format. The hexadecimal display is useful for identifying any non-printing characters.

DUMP *fid* G
will dump the contents of the frame(s), displaying the characters held in the frame(s).

DUMP *fid* GX
will dump the contents of the frame and all the following frames in that group, displaying the contents in both character and hexadecimal format.

To *print* a hard copy of the data on the printer, the P option should be included:

```
DUMP fid P
DUMP fid XP
DUMP fid GP
DUMP fid GXP
```

The options may be specified in any order.

McDonnell Douglas implementations require parentheses:

```
DUMP fid (P
DUMP fid (XP
DUMP fid (GP
DUMP fid (GXP
```

According to which implementation you are using, the DUMP output may or may not include the first 12 (or 24 or 48) bytes at the start of the frame which are used to hold the linkage information. If the linkage information is *not* shown in the body of the frame, then the numbering down the side of the frame (representing the byte address within the frame) may be incorrect.

Any data beyond the end-of-group marker in the group (we have just a string of full stops in our example above), is the data which just happened to be in that frame when it was allocated to our file. It may even contain relics of old items which once existed in that group. Whatever lies there is of no concern to our processing and is rendered inaccessible by the presence of the end-of-group marker.

Users who are unfamiliar with the jargon should note that DUMP here means *output a copy of* and not *get rid of*.

If you do not know the FID of the frame, but you do know the item-id of an item near the GFE, then you can use a command of the form:

```
ITEM filename itemid
```

For example:

```
ITEM STOCK 1234
ITEM DICT STOCK DESCRIPTION PRICE
ITEM STAFF B001 N003 N034 B888
ITEM STAFF B234 (P
```

and this will display (or print if the (P option is used) the FID and all the items in the group. There is also the:

```
GROUP filename
```

command to display the FIDs of the first frames in each group of the file. For example:

```
GROUP STOCK
GROUP DICT STOCK
GROUP MD (P
```

These and other commands which are used to investigate the structure of your files, are described in other titles in the *MB-Guide* series.

4 Fixing GFEs

Each time a GFE is detected, the GFE handler will display a message such as:

```
GFE HANDLER INVOKED
```

or:

```
GROUP FORMAT ERROR AT .xxxx
```

On some implementations, it will keep repeating this message until you enter a valid response; on other systems it may go on to display a message indicating the action which you may take at this point:

```
Enter F(ix) / K(ill) / D(ebug) / E(nd)
```

The responses to this message are:

D to enter the interactive system debugger.

This options leaves the GFE handling software and enters the standard system debugger (with the ! prompt). From there, you may issue one of the standard responses:

```
END
OFF
G
```

or you may use the other facilities of the system debugger to display and/or patch the offending frame.

E to terminate the process and return to TCL.

F Fix the error. This action generally results in your losing the data beyond the offending point, whilst data up to (and possibly including) the offending item will not be lost. For this reason, it is a good plan first to DUMP the contents of the frame(s), as described above.

The following action is normally taken by the GFE handler in *fixing* a group format error:

* Set the end-of-group marker at the front of the offending item, thereby deleting this and all following items in the group.

* Delete the offending item, but retain the other items in the group.

K Kill the error. This action generally results in your losing *all the data in the group* where the error occurs.

The following action is normally taken by the GFE handler in *killing* a group format error:

* Set the end-of-group marker at the front of the offending group, thereby deleting all the items in the group.

The Kill option is not available on all implementations.

4.1 FIX-FILE-ERRORS

Some implementations, notably McDonnell Douglas Reality and Ultimate, have a utility to assist users in recovering from GFEs. This operates upon a specific file by dumping the entire contents of any faulty frames to another file called TSYM. Having dumped the frame contents to the TSYM file, the offending group(s) are cleared by setting an end-of-group marker at the front of the group. The erroneous frames can then be inspected and recovered manually.

The format of the command is:

```
FIX-FILE-ERRORS fffff
```

where *fffff* is the name of the file which is known to contain GFEs. If the GFEs occur in the DICT section of the file, then you will use the command:

```
FIX-FILE-ERRORS DICT fffff
```

The sequence of operations when fixing a GFE by means of the FIX-FILE-ERRORS command is:

1) Create a file called TSYM, using a command of the form:

```
CREATE-FILE DICT TSYM 17,1
```

The size of the file is arbitrary.

If the TSYM file already exists, then you can re-use it clearing the results of any previous FIX-FILE-ERRORS activity:

```
CLEAR-FILE DICT TSYM
```

2) Issue the command:

```
FIX-FILE-ERRORS fffff
```

The FIX-FILE-ERRORS operation will scan through the file *fffff* and transfer to the TSYM file the contents of any frames which have a GFE, saving each set of data as a separate item on TSYM.

3) Recover the data items from TSYM

Because the FIX-FILE-ERRORS processor is unable to interpret the data correctly, the data items which are left on TSYM are in a very raw state, and include the four-byte hexadecimal item-length field and the item-id of the original items.

The item-ids of the newly-created TSYM items consist of a single letter indicating the nature of the error, followed by the FID of the frame where the error occurred, followed by a sequence number if there are several errors in any one frame. The following codes indicate the nature of the error:

O premature end of data.

N invalid character in item-length counter.

C item-length counter invalid, that is, less than 5 or greater than 31764 bytes.

A invalid end-of-item marker.

I item-id invalid / is longer than 50 characters.

H item in wrong group.

null a segment mark was found within the item.

You should use one of the editors to edit the items on DICT section of TSYM so that they reflect - as accurately as possible - the correct format of the items as they were on the file *fffff*.

Finally, you should copy the items back to the original file *fffff* and at the same time allocate the required item-ids.

The following points should be borne in mind when using the FIX-FILE-ERRORS verb:

* The item recovered as a result of some types of data error - such as a non-hex character in the item byte count - may lose much of its data.

* The FIX-FILE-ERRORS operation may overwrite *all* existing items TSYM file. For this reason, the facility should not be used again - by any user on the same account - before the results of a previous FIX-FILE-ERRORS have been recovered, and the TSYM file cleared or deleted.

* On some implementations, only the DICT section of the TSYM file is used. Since the FIX-FILE-ERRORS operation may overwrite *all* existing items of the DICT section of the TSYM file, *including the data-level identifier*, you may lose any associated data section. For this reason, it is best to use TSYM as a DICT-section only file,

* If the FIX-FILE-ERRORS routine encounters the segment-mark (hexadecimal FF) in the item, then it will generate a message of the form:

    SM @ fid,d

  showing the position of the offending character (as a displacement *d* from the beginning of frame *fid*), and the erroneous items will be amended and returned to the original file, the offending character being replaced by the < character. In this case, no item will be placed on TSYM.

4.2 Another technique for recovery

If you do not have a facility such as FIX-FILE-ERRORS, then it is possible to write a Basic program to perform a similar

task. The action of the program would be:

* Create a workfile to hold the recovered data.

* Determine the FID of the frame(s) in which the GFE is encountered. This is a matter of programming ingenuity.

* Execute the DUMP verb to capture the contents of the offending frame(s).

* Manipulate the captured display to remove the extraneous information added by the DUMP processor.

* Write the captured output to the workfile.

The items on the workfile can then be edited manually (using one of the Pick editors) and returned to the original file, in much the same manner as that described for the FIX-FILE-ERRORS items.

This solution does not remove the GFE. This must still be fixed by one of the other methods described below, and then the items may be returned to the file.

4.3 Patching a frame

The last - and most desperate - means of recovering lost data is by using the interactive system debugger to *patch* the offending frame, replacing the actual incorrect bytes by correct data.

We include this technique for the sake of completeness.

THIS IS NOT TO BE RECOMMENDED AND SHOULD ONLY EVER BE USED BY SOMEONE WHO IS COMPLETELY AWARE OF THE POSSIBLY DISASTROUS CONSEQUENCES OF THE ACTION.

In order to patch a frame you must know:

* The exact *contents* of the *incorrect* data. Write down this information.

* The exact *location* of the incorrect data:

  1) The FID of the frame,

  2) The position within the frame at which the incorrect data occurs. This is the *displacement* of the start of the incorrect data string from the front of the frame. When deciding this information, you should remember the earlier observation about the first 12 (or 24 or 48 or 96) bytes of the frame which are used to held the linkage information; these must still be included in the displacement count.

     Thus, if you wish to patch the first byte of data in a 512-byte frame, this will have a displacement

of 12 (not 0 and not 1).

3) The *exact length* of the incorrect data string. This is referred to as the *window*.

Write down this information.

* The exact contents of the *correct* data. Write down the exact characters which you want to write into the frame.

If the data contains any special characters, such as the field-separators, the end-of-item marker, or the end-of-group marker, then the replacement data should be expressed in hexadecimal notation. The hexadecimal notation can be found in the table of ASCII characters and can also be seen in the hexadecimal part of the:

DUMP *fid* X

output.

If the data consists entirely of display characters, such as would the four-byte item-length counter, then this may be expressed in character form, exactly as it appears on the DUMP display.

We can illustrate the concept of displacement by looking at the first few bytes of the frame 12345 which we dumped earlier:

```
FID:  12345 :  0  0  0  0          ( 3039 : 0  0  0 0 )

  1 :003D1000^DESK, GREEN-BLUE, ASH^8^LS/17/81^5600^30^:
 51 :2000^8205^_003C2000^SETTEE, YELLOW, OAK^18^DN/1/69:
101 :^10000^30^1000^8176^_003D3000^SIDEBOARD, BLUE, ASH:
151 :^58^MN/5/56^13000^15^2000^8178^_003A4000^DESK, BLA:
201 :CK, MAPLE^68^LS/7/87^5600^30^1000^8173^_003B5000^S:
251 :ETTEE, ORANGE, ASH^24^DN/19/3^1000^30^1000^8180^__:
301 :.................................................:
351 :.................................................:
401 :.................................................:
451 :.................................................:
```

The locations of various pieces of data in this particular instance are as follows:

* The four-byte item length count field 003D of the first item starts in the very first data byte at displacement 12 (that is, the displacement is 1+12-1 = 12).

  The number 12 is used when the frames are 512 bytes in size; for 1048-byte frames, you will use the number 24; for 2048-byte frames, you will use the number 48, and so on.

* The four-bytes count field 003C of the second item starts at displacement 73 (that is, 62+12-1).

* The end-of-group marker after the very last item starts at displacement 311 (that is, 300+12-1).

Summarising this, the displacement of any character shown in position *p* on this display is:

p + 12 - 1

Some versions of the DUMP verb may include the 12 control bytes (or 24, 48 or 96) in the display and show the true byte-positions, ranging from 1 to 512 (or 1024, 2048 or 4096).

Armed with this information, you are now ready to patch the frame. The steps to be taken are as follows:

1) DUMP a copy of the frame before you start to make any changes.

2) Enter the interactive system debugger by hitting the <Break> key.

3) When you get the ! prompt character, enter your specifications in the form:

f,d;w

where *f* is the FID of the frame which is to be patched; *d* is the displacement of the first byte in the string which is to be changed; *w* is the window (the length of the data string to be changed).

Pay particular to the punctuation here: a *comma* follows the FID and a *semi-colon* follows the displacement. An invalid command will be rejected by the system debugger.

4) The debugger will then display the *w* characters starting at byte displacement *d*. Confirm that this is the offending string.

If this is the wrong string, hit <Return> and try again from step (3).

5) Enter the correct data as a series of *pairs* of hexadecimal characters (preceded by a full stop) or as a series of display characters (preceded by an apostrophe). For example:

.FF

or:

'003D

6) When you are satisfied that you have entered the data, hit the <Return> key.

7) Abandon the system debugger by entering:

END

followed by the <Return> key.

8) DUMP a copy of the frame after you have made the changes and confirm that the action has taken place successfully.

Further deatils about the system debugger can be found in the reference literature for your system and also in the *MB-Guide to the System Debugger.*

4.4 Patching a frame - an example

Thus, to set an end-of-group marker in the first byte of the above frame, the first part of the dialogue with the debugger might look like this:

```
!C12345,12;1 0=
```

the system debugger indicating that this byte currently contains a zero, and then, when I enter the data to reset this to an end-of-group marker (hexadecimal 55 in this instance), the display might look like:

```
!C12345,12;1 0=.FF
```

If you feel that you will be likely to need to practise this technique, then it is a good plan to create a dummy file and a large dummy item with recognisable data and use this to patch parts of the *data* of the item. This will be perfectly safe, provided that you do not alter the item-length counter or the final end-of-item marker. When you have practised sufficiently, you should delete the file.

As we have seen, what happens when a GFE is detected by the operating system, depends upon which implementation you are using. The *GFE handler* may continue with the processing, ignoring any items in the affected area, making the best of a bad job, or it may give you the opportunity to:

* Enter the interactive system debugger.

* Abandon the process and return to TCL.

Unless you are sure that it is a hard GFE, it is not a good plan to fix or kill the GFE without thinking about the problem. So, if in doubt, you should abandon the process and return to TCL.

4.5 Software for fixing GFEs

There are several pieces of commercial software available for fixing GFEs. These are advertised in the journals and magazines for the Pick industry. It is hoped to be able to include information on these in a future edition of this *MB-Guide.*

5 Glossary

This section describes some of the terms which are used in this *MB-Guide*. These and other definitions may also be found in the *MB-Guide to Pick terminology*.

End-of-group marker
: A sequence of characters (typically the system delimiter segment mark) which is used to indicate the physical end of the data in a group of a file as it is recorded on disk. This marker is written and used by the operating system and is not accessible to the user.

End-of-item marker
: A sequence of characters (typically the system delimiters attribute mark and segment mark) which is used to indicate the physical end of an item on disk. This marker is written and used by the operating system and is not accessible to the user.

FID
: Abbr: *Frame identifier*, the number which identifies each frame of virtual memory. It can be thought of as the *address* of each frame on the hard disk.

GFE
: Abbr: *Group Format Error*.

Group format error
: A serious error in the data contents of a frame of the Pick operating virtual memory. The presence of a GFE results in the operating system being unable to interpret fully the contents of that frame and to reach and process data which lies in that frame and also in the group of frames which are linked on beyond the site of the error.

Item-length counter
: A control field (usually a four-byte hexadecimal number) which precedes each physical item on disk to indicate the total length of the string of data representing that item. This field is written and used by the operating system and is not accessible to the user.

Soft GFE
: A situation which is interpreted momentarily as a GFE but which later resolves itself and does not appear again. It may be caused by a transient condition when two users are handling exactly the same data at the same moment.

In the *MB-Guides*, and elsewhere in the literature, the keyboard control keys are represented by their name enclosed in angle brackets:

<Break> identifies the break key or the equivalent sequence which interrupts the current process.

Depending upon which terminal you are using, this may be any of

<Break>
<Ctrl> <Break>
<Shift> <Break>
<Ctrl> <Shift> <Break>

<Ctrl> identifies the control key.

Certain characters are entered at the keyboard as a combination of one or more of the above keys together with other keyboard characters. For example, the subvalue-mark (character 252) may be entered as:

<Ctrl> \

that is, by holding down the <Ctrl> key and typing the normal \ character at the same time. Similarly, the value-mark can be keyed in as the sequence <Ctrl>] and the attribute-mark as the sequence <Ctrl> ^

<Enter> identifies the ENTER key which is used to transmit each piece of data to the system. The sequence <Ctrl> M is equivalent.

This is generally represented by the <Return> key in the text.

<Esc> identifies the ESCAPE key.

<Line feed> identifies the line-feed key. On some keyboards, this may be the down-pointing arrow. The sequence <Ctrl> J is equivalent.

<Return> identifies the RETURN key which is used to transmit each piece of data to the system.

On some keyboards, this may be the <Enter> key or the down-left-pointing arrow key. The sequence <Ctrl> M is equivalent.

Your reference literature might also use:

<Carriage-return>
or
<CR> to represent the key which we have shown here as <Return> or <Enter>. In general there is no difference between <Return> and <Enter>.

I n d e x

MB-Guides

The booklets in the *MB-Guide* series cover a range of fundamental topics of interest to users and those responsible for running Pick systems.

Each *MB-Guide* deals with a specific aspect of the operating system and the booklets represent an economical introduction to the various topics and the whole series forms an integrated presentation of the subject matter.

The booklets are intended to be a working document and, for this reason, space is provided for the user's notes, and the reader is encouraged to amend the booklet so that it applies to his/her own system.

It is anticipated that the series of *MB-Guides* will be of special interest to new users, and it should prove useful for software houses and others who are responsible for the instruction of their clients and staff in the fundamental aspects of the Pick operating system.

Malcolm Bull

Training and Consultancy Publications