ACTION! BUG SHEET #3 - part 1 to 6#

This document supercedes the previous two bug sheets published for ACTION!

November 6, 1984

Table of Contents

GENERAL INFORMATION#

Before getting to the bad stuff (the bugs), here are some goodies about ACTION! which we would like to pass on to you:

P1. TIPS ON TEMPS#

A magazine article titled "Lights, Camera, Action!" (by Dave Plotkin) which appeared in the July 1984 issue of ANTIC featured a set of routines to facilitate writing ACTION!-based interrupt handlers.

The article gave the listings for two routines (more properly, two DEFINEs) named "SaveTemps" and "GetTemps". These routines are adequate only if no math beyond addition and subtraction is performed in the interrupt service routine. The following versions of these two routines will work properly in the more general case:

Make the following DEFINEs in your program before you declare your interrupt routine (comments may be omitted-they exist only for clarification):

```
DEFINE SaveTemps=[
     $A2 $07     ;       LDX #7
     $B5 $C0     ; LOOP  LDA $C0,X
     $48         ;       PHA
     $B5 $A0     ;       LDA $A0,X
     $48         ;       PHA
     $B5 $80     ;       LDA $80,X
     $48         ;       PHA
     $B5 $A8     ;       LDA $A8,X
     $48         ;       PHA
     $CA         ;       DEX
     $10 $F1     ;       BPL LOOP
     $A5 $D3     ;       LDA $D3
     $48         ;       PHA
  ]

DEFINE GetTemps=[
     $68         ;       PLA
     $85 $D3     ;       STA $D3
     $A2 $00     ;       LDX #0
     $68         ; LOOP  PLA
     $95 $A8     ;       STA $A8,X
```

```
    $68         ;       PLA
    $95 $80     ;       STA $80,X
    $68         ;       PLA
    $95 $A0     ;       STA $A0,X
    $68         ;       PLA
    $95 $C0     ;       STA $C0,X
    $E8         ;       INX
    $E0 $08     ;       CPX #8
    $D0 $EF     ;       BNE LOOP
  ]"
```

Use these routines inside your interrupt routine as follows:

Your interrupt routine.

```
PROC InterruptRoutine()
  ; Local declarations, if any.
    BYTE a, b, c, etc.
  ; First line of code within
  ; procedure SaveTemps

    ... ; Your interrupt
        ; code goes here.

    GetTemps      ; Last line of code
                  ; within procedure.
[$6C OldVBI]      ; A special way to
                  ; end for VBIs- see
                  ; below.
```

For example, the following program will set up the routine ChangeColor as a vertical blank interrupt routine (hit the START key to exit the program):

```
DEFINE SaveTemps=
   [ $A2 $07 $B5 $C0 $48
     $B5 $A0 $48 $B5 $80
     $48 $B5 $A8 $48 $CA
     $10 $F1 $A5 $D3 $48 ]

DEFINE GetTemps=
   [ $68 $85 $D3 $A2 $00 $68
     $95 $A8 $68 $95 $80 $68
     $95 $A0 $68 $95 $C0 $E8
     $E0 $08 $D0 $EF ]

CARD OldVBI    ; Will hold previous
               ; contents of vertical
               ; blank interrupt
               ; vector.

; This procedure will change the background color to random values.
; The main routine will set up this code to operate during the
; deferred vertical blank interrupt.
PROC ChangeColor()
    BYTE hue, lum

    SaveTemps
    hue = Rand( 16 )
    lum = Rand( 16 )
    SetColor(2,hue,lum)
```

```
    GetTemps
[ $6C OldVBI ]  ; Vertical blank
                ; interrupts must end
                ; like this ($6C is a
                ; 6502 indirect jump
                ; instruction).

PROC Test()     ; Main routine
 BYTE critic=$42, ; Critical I/O flag
      console=$D01F ; Console key
         ; hardware location
 CARD VBIvec=$224 ; Deferred vertical
         ; blank interrupt vector

    ; You must install a VBI routine like this:
    critic = 1    OldVBI = VBIvec
    VBIvec = ChangeColor
    critic = 0

    ; ChangeColor is now running as the vertical blank interrupt
    ; routine-- since our mainline code has nothing more to do,
    ; we just go into a loop waiting for the START key to be
    ; pressed.
    WHILE console&1
      DO
      OD

    ; Now turn off the VBI routine.
    critic = 1
    VBIvec = OldVBI
    critic = 0
RETURN
```

This method of saving and restoring ACTION zero page variables may also be used to write BASIC machine language subroutines in ACTION! Your main ACTION routine should then have SaveTemps as the first executable line, and GetTemps as the last executable line before the RETURN statement.

P2. BUGS IN THE ACTION! CARTRIDGES#

The following is a list of all bugs we currently know exist in the ACTION! cartridge. We list these bugs separately from those in the RunTime library and/or the PAD disk or ToolKit, which occur in following pages. Each bug is described in detail and, when possible, bug fixes are given. Many of these bugs deal only with specific versions of ACTION!. To find out which version of ACTION! you own, type the following from the ACTION! monitor:

```
 ?$B000 [[RETURN]
```

Below is an actual copy of what printed following that command for one of our cartridges.

```
  45055,$B000 = 0 $0730 48 1840
                ^
```

To find out the version number, look at the character to the right of the equals sign (here printed with a caret under it). The "0" in this case implies that the cartridge is version 3.0. If yours has a "6", you own version 3.6, etc. As of the date of this bug sheet, the current cartridge version is 3.6.

1. OFFSETS

Using a TYPE declaration will generate a spurious error whenever the code offset (contents of location $B5) is non-zero.

Affects: All versions of the cartridge to date. (Presumably only noticed if using RunTime disk, though.)

Fix: Make all TYPE declarations before changing the code offset. Example:

```
; Beginning of program --
; First, declare TYPEs
TYPE IOCB =
 [
 BYTE Id, Devnum,
      Command, Status
 ]
 ; Then, if desired,
 ; change offset
 SET $B5 = $1000
 ; example: offset=4096
```

2. OFFSETS#

Using a code offset greater than $7FFF (i.e., a negative offset, if you consider it to be of type INT) causes the compiler to generate improper code.

Affects: All versions, especially when used with the RunTime disk.

Fix: No direct fix, but you may use the relocator program described later in this document (which is also usable with assembly language).

3. ATARI DOS#

Exiting to Atari DOS from ACTION! can cause a system crash if DUP.SYS is not present on the disk in drive 1.

Affects: All versions, but only when used with Atari DOS.

Fix: Use DOS XL (or be careful when exiting to DOS).

4. ARRAYS AND ELSEIF#

We have just learned that there is a relatively obscure bug in ACTION! related to the use of ELSEIF. In particular, statements similar to the form ELSEIF a(i) = 0 THEN ... (where a is an ARRAY and i is a CARD OR INT), or statements like ELSEIF p^ = 0 THEN (where p is a POINTER) produce incorrect code.

Affects: All versions

Fix: There is no direct fix at this time. The best way around the problem seems to be to code something like this:

```
t = a(i) ; t is an INTEGER
  ...
ELSEIF t=0 THEN ...
```

This works properly.

5. WRITING OBJECT FILES#

If a monitor Write command fails because of a disk error (e.g., disk full, 162, or device done, 144), the IOCB is not properly closed. If the disk is hanged before another disk operation is performed, the new disk can have invalid data written to it.

Affects: All versions

Fix: If you get an error when writing an ACTION! object file, type the following command to the monitor:

```
X Close( 1 ) [RETURN]
```

You can then erase the file which caused the error.

6. HEX ARRAY SIZES#

Hexadecimal values as array dimensions cause incorrect code to be generated.

Affects: All versions

Fix: Use decimal array dimensions.

7. TYPE POINTER ARGUMENTS#

PROC/FUNC declarations with record pointer arguments other than the first don't compile correctly. For example, the following code generates an error 7 (invalid argument list):

```
TYPE REC=[...]
   ...
PROC Test( BYTE x, REC POINTER p )
```

Affects: All versions

Fix: Omit the comma in the argument list for the PROC/FUNC, as in:

```
PROC Test( BYTE x
           REC POINTER p )
```

As this is just a temporary fix, it may not work in future versions, but the correct declaration (with the comma) will.

8. MONITOR LOCKUP#

Typing the following command from the monitor will lock up the system:

```
R* [[RETURN]
```

Affects: All versions

Fix: Don't do it! If you do type that command, hit [RESET]

9. PADDLE FUNCTION#

The Paddle function does not work properly in all versions of the ACTION! cartridge.

Affects: Versions 3.0 to 3.5

Fix: Make the following declaration in your program:

```
BYTE ARRAY Paddle(4) = 624
```

10. SOUND ON CHANNELS 3 AND 4#

If you use a Sound() procedure call after having done any disk I/O, sound channels 3 and 4 will remain silent. This is because Atari's OS does not reset some of the serial control registers completely.

Affects: Versions 3.0 to 3.5

Fix: Type in and use the following procedure. You should call this before doing any Sound() calls and/or in place of any SndRst() calls:

```
; Contributed by Michael Ross
  PROC SoundOff()
      BYTE AudCtl = $D208,
           SSKCtl = $232,
           SKCtl  = $D20F
      SSKCtl = 3
      SKCtl  = 3
      AudCtl = 0
      SndRst()
  RETURN
```

11. TYPE FIELDS AS PARAMETERS#

Using fields of TYPEs as parameters to PROCs or FUNCs generates incorrect code. For example,

```
MoveBlock( rec.addr1,
   rec.addr2, length )
```

Affects: Versions 3.0 to 3.5

Fix: Assign the TYPE field to a temporary variable and pass that as a parameter:

```
temp1 = rec.addr1
temp2 = rec.addr2
MoveBlock(temp1,temp2,length)
```

12. SASSIGN PROBLEMS#

SAssign does not work properly when the source string has a length of zero.

Affects: Versions 3.0 to 3.5

Fix: No fix available at this time.

13. CARD FIELDS IN TYPES#

Accessing CARD fields of TYPEs generates incorrect code.

Affects: Versions 3.0 to 3.2

Fix: No fix available at this time.

14. MOVEBLOCK PROBLEMS#

MoveBlock does not move more than 256 bytes of data.

Affects: Versions 3.0 to 3.2

Fix: No fix at this time. You could write an ACTION! routine to do the equivalent.

15. CONTROL-SHIFT RETURN#

Using [CS] RETURN to split a line into two lines generates garbage in the second line.

Affects: Versions 3.0 and 3.1

Fix: No fix available, but not a disastrous problem.

16. DIVISION ERRORS#

On old cartridges, neither the "/" operator nor the "MOD" operator works properly under certain conditions.

Affects: Versions 3.0 and 3.1

Fix: Insert the following code into your program before any of your own PROCedure or FUNCtion declarations (this can be done easily using INCLUDE):

```
; Copyright (c) 1983 by
; Action Computer Services
;
; Permission is granted to duplicate and/or distribute
; the contents of this file to ACTION! users.  Copies of
; this file may not be sold or used for monetary gain.

PROC DivI=*()
[$20 $A06C $85 $86 $A2 $10
 $26 $82 $26 $83 $26 $86 $26
 $87 $38 $A5 $86 $E5 $84 $A8
 $A5 $87 $E5 $85 $90 $04 $85
 $87 $84 $86 $CA $D0 $E5 $A5
 $82 $2A $26 $83 $A6 $83
 $4C $A032]

PROC RemI=*()
[$20 DivI $86A5 $87A6 $60]

SET $4EA=DivI
SET $4EC=RemI
```

17. ERROR ROUTINE NOT INITIALIZED#

The address of the Error PROCedure is not restored by ACTION! if a user program has changed it.

Affects: Versions 3.0 and 3.1

Fix: Make sure to restore the original Error vector upon exiting a program, if you changed it.

18. COMPLEX EXPRESSIONS IN UNTIL#

Complex relational expressions in an UNTIL statement generate incorrect code. For example,

```
DO
 ...
 UNTIL a>0 AND b=3
OD
```

Affects: Versions 3.0 and 3.1

Fix: Assign the expression to a temporary variable and test that variable, instead:

```
DO
 ...
 temp = a>0 AND b=3
 UNTIL temp
OD
```

19. BANK SWITCH BUG#

When loading and running compiled ACTION! object files from DOS, the system can crash when using older cartridges. This is because the ACTION! library is not accessible.

Affects: Version 3.0 only

Fix: Put the following program lines at the VERY BEGINNING of your main procedure (i.e., the last procedure in your program):

```
BYTE bank = $D500
; This declares the variable
; 'bank' to reside at $D500.

bank = 0 ; This must be the
; first executable statement.
```

20. .COM PROGRAMS#

Running compiled ACTION! programs as .COM files under OS/A+ causes those programs to execute twice.

Affects: All versions, but only when using a version of OS/A+. DOS XL is not affected.

Fix: Insert the following as the first global variable you declare:

```
BYTE RTS=[$60]
; This MUST be the first line in your program,
; aside from comments and SET commands.
```

21. PROC ADDRESSING#

Under certain conditions, specifying the address of a procedure (e.g., to interface to a machine code routine) causes ACTION! to generate incorrect code which could cause your program to"hang".

Affects: Versions 3.1 and 3.4

Fix: Insert an empty code block after the declaration of a procedure whose address is specified. For example:

```
PROC CIO = $E456() []
; An empty code block!
```

22. ERROR #3#

If you get an ERROR 3 during a compile, the system hangs when you return to the editor.

Affects: All versions

Fix: Do not go to the editor until you type the following line to the monitor. This command resets the ACTION! memory pointer.

```
SET $E=$491^
```

23. STRING INPUT#

When using the string input library functions (InputS, InputSD, and InputMD), there must be room in the string for the termination EOL, even though the resulting string length will not include it.

Affects: All versions

Fix: Adjust your declaration appropriately.

P3. BUGS IN THE ACTION! RUNTIME LIBRARY#

We have found a few bugs in the original version(s) of the RunTime Library Disk. Fortunately, they are all easy to fix. (The RunTime library is independent of the cartridge, so bugs affect all versions.)

In the fixes given below, the portion to be changed (to implement the fix) is underlined. The rest of the line remains the same. To make the fixes, simply load the library file containing the affected PROCedure, edit, and save it back to disk.

1. Hex numbers are printed incorrectly by PrintH and the %H parameter of PrintF.#

Fix: Change second line of CCIO:

```
PROC CCIO=*()
[$A386$A0A$A0A$AA$A3A5$9D$342 ...
 ---             ---
```

2. PrintBDE can cause a spurious compile time error.#

Fix: Change first line of PrintBDE:

```
PROC PrintBDE =*(BYTE d,n)[$A0$0]
```

--

3. A minor error exists in ChkErr.#

Fix: Change second line of ChkErr:

```
PROC ChkErr=*(BYTE r,b,eC)
     [$1610$88C0$8F0
       $98$80C0$12F0 ...
              ---
```

4. If your program redefines a

library procedure (e.g., one which declares its own version of PROC Graphics), it will compile with no errors using the cartridge only (because declared procedures take precedence over built-in ones). However, since the RunTime library uses this same precedence trick to include its own definitions of library procedures, your program will generate Error 6 (doubly defined name) if you do not delete the appropriate PROCedure (or FUNCtion) from the RunTime library before INCLUDEing it.

Fix: Make a custom version of the RunTime library on a COPY (please, only on a copy) of your RunTime disk which does not contain the routines you wish to replace.

5. On page 17 of the Reference Guide for the Runtime Package, the DEFINE for ROM will cause incorrect code if you use local variables.

Fix: Use the following form of definition, instead:

```
DEFINE ROM = "BYTE ZZQQJUNK
              -------------
             SET $680 = $E^
             SET $B5  = $5800
             SET $E   = $682^"
```

P4. PROBLEMS WITH Programmer's Aid Disk

We will list the problems (and solutions) regarding the Programmer's Aid Disk (PAD) here in reasonably compact form.

1. BGET/BPUT PROBLEMS#

The BGet and BPut routines in the IO.ACT file do not work properly under certain conditions. To fix this bug, replace the BGet and BPut routines with the following ACTION! code:

```
;*******************************
;Burst (Block) I/O routines to do
;quick disk I/O, utilizing a call
;to CIO
;*******************************

PROC CIO=$E456( BYTE areg, xreg )

;*******************************

CARD FUNC Burst( BYTE chan, mode,
            CARD addr, buflen )

  TYPE IOCB=[BYTE id,num,cmd,stat
             CARD badr,padr,blen
             BYTE a1,a2,a3,
                  a4,a5,a6]

  IOCB POINTER iptr

  chan ==& $07
  iptr = $340+(chan LSH 4)
  iptr.cmd  = mode
  iptr.blen = buflen
  iptr.badr = addr
  CIO( 0, chan LSH 4 )
```

```
RETURN( iptr.blen )

;*******************************

CARD FUNC BGet( BYTE chan,
                CARD addr, len )
  CARD temp

  temp = Burst(chan,7,addr,len)
RETURN( temp )

;*******************************

PROC BPut(BYTE chan,
          CARD addr,len)
  Burst( chan, 11, addr, len )
RETURN
```

2. PRINTF#

The PRINTF routine has a bug which was reported and fixed in the Spring, 1984 newsletter. In the file PRINTF.ACT, use the ACTION! editor to find

```
   args ==+ s
```

and change it to

```
   args ==+ 2
```

3. PLAYER/MISSILE GRAPHICS#

Because S: uses some memory just below the display list (undocumented), our method of finding the base address for Player/Missile Graphics needs a slight revision. Use the ACTION! editor with the file PMG.ACT to find

```
PM_BaseAdr=(HiMem-
           PM_MemSize(mode))
           &PM_AdrMask(mode)
```

and change it to

```
PM_BaseAdr=(HiMem-
           PM_MemSize(mode)-$80)
           &PM_AdrMask(mode)
```

4. PMMove#

If you use the PMMove procedure and specify a vertical movement of zero, the horizontal movement does not take place (it should). To fix this, change the lines in PMG.ACT which read

```
IF deltay=0 THEN
 RETURN ; do nothing
FI
```

to the following:

```
IF deltay=0 THEN
 ; do horizontal anyway
 PMHpos(n)=x
```

```
 RETURN
FI
```

5. PLAYER/MISSILE GRAPHICS#

The documentation for PMG.ACT states that you may read the contents of PMHpos to find the horizontal position of a player or missile. This is simply not true. PMHpos is a set of write-only hardware registers. (Note that in the ToolKit we have added a shadow array and changed the name of the hardware registers, so this works correctly. If you wish, you could consider doing something similar on your PAD.)

6. REAL NUMBER ROUTINES#

There are two discrepancies in PROCedure names in the REAL.ACT library as compared to the REAL.DOC documentation, as follow:

Name in .DOC	Name in .ACT
StrR	RealToStr
ValR	StrToReal

We suggest that you change the source code in REAL.ACT to reflect the names given in the documentation (rather than vice versa), since this makes the names appear compatibile with the library's other number-string conversion routines.

7. REAL NUMBER ROUTINES#

In that same area, the routine RealToStr (or should that be StrR?) needs to change the line which reads ptr=LBuff to the following:

```
ptr=InBuff
```

8. ALLOC.ACT#

The free list pointer may not be set up properly. Also, when freeing a block, right adjacency is not handled properly if left adjacency has already been found. Fix these problems as follows:

In the PROCedure Free, after the line reading:

```
last.size==+nBytes
```

insert the line:

```
target=last
```

Also, in the same procedure, change the line reading:

```
IF target+nBytes=current THEN
```

to read:

```
IF target+target.size
     =current THEN
```

In the PROCedure AllocInit, replace the line reading:

```
FreeList.next=p
```

with the following lines:

```
FreeList=p
```

```
p==+4
FreeList.next=p
```

P5. TOOLKIT TROUBLES#

It's hard to believe that a product as new as the ACTION! ToolKit can already have bug reports. Sigh. Anyway, there are already three versions of the ToolKit. Version 1 has 31 free sectors (when you list its directory). Version 2 has fewer free sectors and the second line of the file MUSIC.DEM reads ";Version 2". On version 3, the file ABS.ACT starts with the version number. This last convention will be followed in future versions. The comments here are organized by affected version(s).

VERSION 1 ONLY#

1. I/O ROUTINES#

The manual describes a routine called Format (in the IO.ACT library), but no such procedure exists on the disk. However, the routine is there--it's just called Init instead. You should change your disk to match your manual.

2. MUSIC.DEM#

The program called MUSIC.DEM will not work as is on older 400/800 machines. This is because it uses a call to Graphics(15), which is only available on XL machines. You may change the program to use Graphics(8) with no effect except that the true colors of mode 15 become artifact colors in mode 8 instead.

VERSIONS 1 AND 2#

1. REAL ROUTINES#

There are two discrepancies in PROCedure names in the REAL.ACT library as compared to the REAL.DOC documentation, as follow:

Name in .DOC	Name in .ACT
StrR	RealToStr
ValR	StrToReal

We suggest that you change the source code in REAL.ACT to reflect the names given in the documentation (rather than vice versa), since this makes the names appear compatibile with the library's other number-string conversion routines.

2. SORT ROUTINES#

There are four discrepancies in PROCecure names in the SORT.ACT library as compared to the SORT.ACT documentation, as follows:

Name in .DOC	Name in .ACT
SortB	BSort
SortC	CSort
SortI	ISort
SortS	SSort

Please change your disk file to agree with your manual.

3. PRINTF#

The PRINTF routine has a bug which was reported and fixed in the Sprint, 1984 newsletter. In the file PRINTF.ACT, use the ACTION! editor to find

```
args ==+ s
```

and change it to

```
args ==+ 2
```

VERSIONS 1, 2, AND 3.#

1. ALLOC ROUTINES#

The manual indicates that the procedure AllocInit requires that you pass it the address of the first free byte of memory (because Alloc "dispenses" memory from the first free byte through the top of memory, as correctly described in the manual). However, since you MUST follow the procedure described in the introduction to ALLOCATE.ACT (that is, you must declare in your program a CARD called EndProg and use the command

```
SET EndProg=*
```

after compiling), the parameter to AllocInit is not really needed and so has been eliminated. (AllocInit uses EndProg just as Alloc does.) If you pass a parameter to AllocInit, it will be ignored.

2. WARP.DEM#

No mention is made in the Toolkit manual that this file can only be run when compiled from disk (unless you are using DOS XL to gain extra memory). WARP.DEM is just too big for ACTION! to hold both the source and object in memory at one time.

3. ALLOCATE.ACT#

The free list pointer may not be set up properly. Also, when freeing a block, right adjacency is not handled properly if left adjacency has already been found. Fix these problems as follows:

In the PROCedure Free, after the line reading:

```
last.size==+nBytes
```

insert the line:

```
target=last
```

Also, in the same procedure, change the line reading:

```
IF target+nBytes=current THEN
```

to read:

```
IF target+target.size
      =current THEN
```

In the PROCedure AllocInit, replace the line reading:

```
p=EndProg
```

with the following lines:

```
FreeList=EndProg
p=EndProg+4
```

P6. ACTION MANUAL ERRATA#

First of all, you need to know which version of the manual you have. If Part III is the Language, then you have the first version of the manual. Otherwise, you have the second (newest) version. Unfortunately, both manuals contain content as well as typographical errors. We'll skip the typos and concentrate on the content errors, since typos don't impair your understanding of the language (although you may wonder where we learned to spell).

VERSION 1 ERRATA:#

PAGE ERROR

2 In the last paragraph, it says that the library is on the disk. This is not true. It's in your cartridge.

23 Under the description of <BACK-S>, the comparison with the Atari screen editor is exactly reversed. If you are in REPLACE mode, this key works as in the Atari editor.

26 Under <CTRL><SHIFT>T, it says you may not use lower-case characters as tags. This is untrue.

48 In the NOTE preceeding 4.3, you should add "The *, /, and MOD operators result in an implied INT type. For this reason, multiplication, division, and modulus of large CARD numbers does not always work properly."

49 Section 4.4 says that you may only have one special operator in a complex relational expression. This is untrue. For example, the following is perfectly legal: (x=7 AND y#10) OR z<100

82 Section 6.2.3 implies that you may not use a function as a procedure. This is not true. You may call a function as though it were a procedure, but the value returned from the function is ignored.

97 Section 8.1.1 states that you may either initialize a POINTER to an address or give it a value. Only the second is possible, and you should use this form: BYTE POINTER x=<value> Not this: BYTE POINTER x=[<value>]

99 In example #1 there are two PrintF statements which have "ptr" as one parameter. These should be "bptr", not "ptr".

101 In the last example of ARRAY declaration (BYTE ARRAY tests(5)...), the dimension is overruled by the initialization options, and so its dimension is only three. To fill only the first 3 of 5 elements, do the following: BYTE ARRAY tests(5)=[4 7 18 0 0]

104 In example #3 you see the program line "PrintE(b)". This should read "PrintE(barray)".

108 Section 8.3.1.2 states that you can initialize the fields of a record when you declare it. This is untrue; you may only initialize its address.

110 The program line "rec.level = InputB()" should read "rec.level = GetD(7)".

112 Same as previous error.

115 Same as previous error.

112 The program line "continue=InputB()" should read "continue=GetD(7)"

120 The program line "mode=InputB()" should read "mode=GetD(7)", and the program line "PrintE(name)" should read "PrintE(nameptr)".

115 Same as previous error.

122 The program line "incctr=chgclr" should read "incclr=chgclr".

142 Section 5.3 states that you should not use channel 7. ACTION! uses this channel to get characters from the keyboard, and you may use it to do this also. However, don't close this channel or alter its configuration in any way.

153 The example of declaring an ACTION! procedure at an address is wrong! If you do this, the internal pointer to the procedure will point to the specified address, but the code generated by the procedure will not be there. Instead, it will be in with your main code. Use procedure and function addressing ONLY to call machine language routines.

161 Where the table of contents lists the routines in section 2.3, it should read: PrintBD NOT PrintDB PrintCD NOT PrintDC PrintID NOT PrintDI

162 Where the table of contents lists the routines in sections 6.7 and 6.8, it should read: PeekC NOT CPeek PokeC NOT CPoke

165 Error in section 2.3. See changes for pg. 161 and make similar corrections.

179 Section 6.4 states some information concerning the results of misusing the SCopy routine, detailing that the routine does string trucation, etc., to make the procedure work. This is not true. You must make sure that the strings are compatible in size.

181 Section 6.8 states that the parameters to Poke and PokeC consist only of an address. Instead, they consist of an address and a value, as follows: Poke(<address>,<BYTE value>) PokeC(<address>,<CARD value>)

182 Section 6.11. MoveBlock will move a maximum block of 256 bytes in versions 3.0 to 3.4 of ACTION! Versions 3.5 and up will move any number of bytes.

191 Some error numbers are wrong. The corrections are: 14 Out of Space 15 Missing DO 19 Missing OD 24 Illegal FOR statement 26 Nesting Too Deep 27 Illegal TYPE reference 28 Illegal RETURN 128 BREAK key abort Also, error 62 is error 61, and 54 & 56 do not exist.

197 In the PrintF statement, %D should be changed to %U.

VERSION 2 ERRATA:#

PAGE ERROR

38 Section 2.7, paragraph 3. The last sentence states that you can RUN compiled ACTION! programs from disk. This is untrue. The RUN command will only compile and run ACTION! source files. Use DOS to run compiled object files.

39 The last RUN example (RUN PrintE()) will not work, since RUN expects a file name. Use the "Xecute" command instead.

63 In the TECHNICAL NOTE preceeding section 4.3, "*" should be changed to "*, /, or MOD".

126 The last assignment on the page makes newrecord point to the current record in the array, not the end of the array.

132 The program line "mode=InputB()" should be changed to "mode=GetD(7)".

138 The program line "IF sub(1)=str(ctr)" should read "IF sub(1)=str(ctrl)".

163 The PutDE procedure requires only a channel as a parameter, and does not put out both a character and a <RETURN>. Rather, it puts out a <RETURN> only.

172 In graphics mode 0 and all text windows, color 1 is the character luminance, color 2 is the background color, and color 3 is unused.

174 In section 5.6, references to the "lower right corner" should instead be "lower left corner".

180 Section 6.1.2 states some information concerning the results of misusing the SCopy routine, detailing that the routine does string truncating, etc. This is not true. You must make sure that the strings are compatible in size.

182 Section 6.11. MoveBlock will move a maximum block of 256 bytes in versions 3.0 to 3.4 of ACTION! Versions 3.5 and up will move any number of bytes.

ACTION OBJECT CODE RELOCATION PROGRAM#

The program SIMPLREL.ACT on this BBS may be used to cause an ACTION! program to load and run at a different address than that address at which it was compiled. The same program will also work for assembly language object files, providing you also follow the given instructions.

The program takes two object files as input and produces a third file which will load and run at a desired address. The relocating program prompts the user for the two input files, which must have been compiled one page (256 bytes) apart. It then prompts for an output file name (the relocated file), the page number of the starting address of the first file, and the page number of the desired destination address. Both page numbers must be decimal values. For example, specifying 32 as the destination page will cause the output file to load at address 32*256 ($2000), not $3200. See part V, "The ACTION! Compiler", chapter 2, page 144, for information on compiling programs to a specified address (Used to compile the two object files one page apart).

In order to use the relocating program, download SIMPLEREL.ACT and read the instructions therein.