Compile to disk#

General Information

Author: Action Computer Services
Language: ACTION!
Compiler/Interpreter: ACTION!

version 1.0 last modified October 22, 1984

How to use#

Compile to disk for ACTION! compiler. Note that all ARRAY declarations that generate storage must be before the first procedure declaration or else the address of the storage will not be setup correctly (all dimensioned ARRAYs which are not assigned an initial value except BYTE/CHAR arrays of size 256 or less). Local ARRAY declarations in the main PROC (last procedure in program) are also allowed. Note: there must be at least one PROC/FUNC in program.

Output file name will be same name as program being compiled with extention .OBJ

IF AN ERROR OCCURS DURING COMPILATION, YOU SHOULD USE "/" to close all open files:

change dev in SPLEnd below to direct output to printer.

```
MODULE ; CMPTODSK.ACT

; Copyright (c) 1983
; by Action Computer Services
; All Rights Reserved

; version 1.0
; last modified October 22, 1984

; Compile to disk for ACTION!
; compiler.  Note that all ARRAY
; declarations that generate storage
; must be before the first procedure
; declaration or else the address of
; the storage will not be setup
; correctly (all dimensioned ARRAYs
; which are not assigned an initial
; value except BYTE/CHAR arrays of
; size 256 or less).  Local ARRAY
; declarations in the main PROC (last
; procedure in program) are also
; allowed.  Note:  there must be at
; least one PROC/FUNC in program.

; Output file name will be same name
; as program being compiled with
; extention .OBJ

; IF AN ERROR OCCURS DURING
; COMPILATION, YOU SHOULD USE
; "/" to close all open files:
;       >/
```

```
; change dev in SPLEnd below to direct
; output to printer.

DEFINE STRING = "CHAR ARRAY"
DEFINE JMP = "$4C" ; JMP addr16

TYPE INSTR=[BYTE op CARD addr]
INSTR Segvec=$4C6
INSTR SPLvec=$4DD
INSTR MonCmd=$4FB
INSTR OldMon

BYTE oldDevice, curBank=$4C9
BYTE pf, Zop=$8A, tZop, dev
CARD curproc=$8E, code=$E
CARD codeBase=$491, codeSize=$493
CARD codeOff=$B5
CARD globals, gsize
CARD totalSize, codeStart
CHAR ARRAY cmdLine(0)=$590
BYTE ARRAY bank(0)=$D500
BYTE ARRAY zpage(32), temps(16)


PROC InitMon()
; add "/" command to monitor which
; closes channels 1-5 and warm
; starts cartridge.

  CHAR cmdchar=$591
  BYTE i, WARMST=$8
  DEFINE JMPI="$6C"

; make sure right command
  IF cmdchar  = '/ THEN [JMP OldMon] FI

  bank(0) = 0 ; init library routines
  FOR i = 1 TO 5 DO
     Close(i)
  OD

  WARMST = 1
[JMPI $BFFA] ; warm start cart.


INCLUDE "BLKIO.ACT"


PROC Save()
; save state of variables used by
; both compiler and library routines

  bank(0) = 0 ; init library routines
  tZop = Zop
  MoveBlock(zpage, $B0, $1B) ; to $CA
  MoveBlock(temps, $5F0, 16)
RETURN
```

```
PROC Restore()
; restore state of variables used by
; both compiler and library routines
  CARD tcodeOff

  Zop = tZop
  tcodeOff = codeOff
  MoveBlock($B0, zpage, $1B) ; to $CA
  MoveBlock($5F0, temps, 16)
  codeOff = tcodeOff

  bank(curBank) = 0
RETURN


PROC WriteHdr()
  PutCD(5, $FFFF)
  PutCD(5, codeStart)
  PutCD(5, codeStart+totalSize-1)
  WriteBlock(5, globals, gsize)
RETURN


PROC WriteCode()
  codeSize = code - codeBase
  PrintD(dev, curproc)
  PrintD(dev, ": ")
  PrintCDE(dev, codeSize)
  totalSize = totalSize + codeSize
  WriteBlock(5, codeBase, codeSize)
  code = codeBase
  codeOff = codeOff + codeSize
RETURN


PROC SegEnd()
  Save()
  IF pf THEN ; print locals
     WriteCode()
  ELSE
     pf = 1
     globals = codeBase
     gsize = code - codeBase
     codeBase = code
     totalSize = gsize
     codeStart = globals + codeOff
     WriteHdr()
  FI
  Restore()
RETURN


PROC SPL() ; dummy proc for call below


PROC SPLEnd()
  CHAR c
  BYTE nxttoken=$D3, i, n, buf=$9B^
```

```
  CARD nxtaddr=$C9, start=$2E2
  STRING inbuf(0)=$5C8, name
  STRING out(17)

  DEFINE PLA = "$68",
           STA = "$8D"

  Save()

  dev = 0
; to get output to printer:
; dev = 4
; Close(4)  Open(4, "P:", 8, 0)

; get output name
  IF nxttoken=30 THEN ; command line
     name = nxtaddr
  ELSE ; editor buffer
     name = inbuf
  FI

; see if device needed
  n = 0
  IF name(2)#': AND name(3)#': THEN
     out(1) = 'D    out(2) = ':  n = 2
  FI

; get name without extension
  FOR i = 1 TO name(0) DO
     c = name(i)
     IF c='. THEN EXIT FI
     IF c>'Z THEN c = c {#&} $5F FI
     out(i+n) = c
  OD

; add extension
  out(i+n) = '.
  out(i+n+1) = 'O
  out(i+n+2) = 'B
  out(i+n+3) = 'J
  out(0) = i + n + 3

  PutE()
  Print("output file is ")
  PrintE(out)
  PutE()

  Close(5)  Open(5, out, 8, 0)
  buf = 0 ; clear buf used by Open

  pf = 0 ; no proc decl yet

; JSR for return so that we come
; back here after compilation
  [
     PLA
     STA SPL+1
     PLA
     STA SPL+2
```

```
  ]
  SPL = SPL + 1 ; get right address
  Restore()

  SPL()

  Save()

; ignore space for arrays
  code = codeBase + codeSize

  WriteCode()
  PutCD(5, $2E2)
  PutCD(5, $2E3)
  PutCD(5, start)
  Close(5)

  Open(5, out, $C, 0)
  WriteHdr()
  Close(5)

  PutDE(dev)
  PrintCD(dev, totalSize)
  PrintDE(dev, " bytes of code")

  Restore()
  codeOff = 0
RETURN


; only code generated before Init is
; allocated space.  Init will be
; garbage collected (well kind of).

PROC Init()
  CARD codeBlock, bsize, csize, nBlock
  CARD POINTER cur, next

; link in our routines
  Segvec.op = JMP
  Segvec.addr = SegEnd
  SPLvec.op = JMP
  SPLvec.addr = SPLEnd
  OldMon.op = MonCmd.op
  OldMon.addr = MonCmd.addr
  MonCmd.op = JMP
  MonCmd.addr = InitMon

; allocate our routine so it won't
; go away.
  codeBlock = codeBase - 4
  next = $80 ; AFbase
  DO
     cur = next
     next = next^
  UNTIL next=0 OR next=codeBlock OD

  IF next=0 THEN
     PutE() Put($FD)
```

```
    PrintE("I can't allocate space for your code")
    PrintE("You better Boot and try again!")
    RETURN
  FI

; assume we can split block
  csize = @codeBlock-codeBlock
  nBlock = next^
  bsize = next(1) - csize
  next = @codeBlock
  cur^ = next
  next^ = nBlock
  next(1) = bsize
  codeBase = next + 4
RETURN
```