Symbol Table Lister#

Table of Contents

The Symbol Table Lister can be used to create an Symbol Import List for linking different parts of ACTION! codes together. This Version prints the symbol table to the Screen, however the Tool can be adapted to print the symbol table to Printer or a file.

Usage:

1. compile your program from memory or disk
2. compile the symbol table lister from disk, at the end of compilation it will print out the symbol table

Symbol Table Lister#

```
MODULE ; ST.ACT

; Symbol table lister for ACTION!
; compiler.  Lists local variables
; per PROC/FUNC and globals at end
; of compilation.

; copyright 1983
; by Action Computer Services
; All Rights Reserved

; version 1.0
; last modified November 6, 1983

; user options:
;
; change Open call in SPLEnd to get
; listing to go to disk


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

TYPE INSTR=[BYTE op CARD addr]
INSTR Segvec=$4C6, DCLvec=$4D4
INSTR SPLvec=$4DD

TYPE ENTRY =
[
; STRING name(?)
  BYTE vtype
  CARD adr
  BYTE numargs
; BYTE ARRAY argTypes(8)
]
```

```
BYTE oldDevice, curBank=$4C9
BYTE pf, Zop=$8A, tZop
CARD curproc=$8E
ENTRY POINTER e
CHAR ARRAY cmdLine(0)=$590
BYTE ARRAY bank(0)=$D500
BYTE ARRAY zpage(32), temps(16)


PROC PrintH(CARD v)
  PrintF("%H", v)
RETURN


PROC BaseType(BYTE et)
  et = et & $7

  IF     et=1 THEN Print("CHAR")
  ELSEIF et=2 THEN Print("BYTE")
  ELSEIF et=3 THEN Print("INT")
  ELSEIF et=4 THEN Print("CARD")
  FI
RETURN


BYTE FUNC GetType(BYTE et)
  CHAR ch
  BYTE pfFlag, t, oldT
  ENTRY POINTER next
  STRING name

  pfFlag = 0

  IF et=39 THEN ; user type
    Print("TYPE=")
    name = e + 3
    next = name + name(0) + 1
    ch = '[
    oldT = 0
    WHILE next.vtype<128 DO
      et = next.vtype & $7
      If et=0 THEN EXIT FI
      IF et=oldT THEN
        Print(", ")
      ELSE
        Put(ch)
        BaseType(et)
        Put(' )
      FI
      oldT = et
      Print(name)
      ch = '
      name = next + 3
      next = name + name(0) + 1
    OD
    IF ch='[ THEN Put('[) FI
    Put(']) 
    RETURN(0)
```

```
  FI

  IF et=27 THEN ; DEFINE
    PrintF("DEFINE = ""%S""", e+3)
    RETURN(0)
  FI

; get basic type
  BaseType(et)

; only record vars less than 128
  IF et<128 THEN ; record
    IF (et&7)=0 THEN
      Print("RECORD")
      IF (et&8)=8 THEN
        Print(" POINTER")
      FI
    ELSE
      Print(" record field")
    FI
    RETURN(0)
  FI

  IF et&$10 THEN ; ARRAY
    Print(" ARRAY")
  ELSEIF et&$40 THEN ; PROC or FUNC
    pfFlag = 1
    IF (et&$F7)=$C0 THEN ; PROC
      Print("PROC")
    ELSE ; FUNC
      Print(" FUNC")
    FI
  FI
RETURN(pfFlag)


PROC PrintEntry(STRING n)
  DEFINE MAX = "15"
  BYTE i, et
  STRING name(MAX+1), t
  BYTE ARRAY argTypes

; get the name
  SetBlock(name+1, MAX, '.)
  SCopyS(name, n, 1, MAX)
  name(0) = MAX

; get address of entry info
  e = n + n(0) + 1

  et = e.vtype
  IF et=$88 THEN RETURN FI ; undeclared

  PrintF("%S ",name)

  IF et=27 THEN ; DEFINE
    Print("     ")
  ELSE
    PrintH(e.adr)
```

```
  FI
  Put(' )

  IF GetType(et) THEN ; PROC or FUNC
    Put('()
    argTypes = e + 3
    t=""
    FOR i = 1 TO e.numargs DO
      Print(t)
      GetType(argTypes(i)%$80)
      t = ", "
    OD
    Put('))
  FI

  PutE()
RETURN


PROC DumpST(CARD POINTER base)
  CARD loc, i
  BYTE low=loc, high=loc+1, ibest
  BYTE ARRAY stLow, stHigh, flags(256)
  STRING best, worst(0)="|"

  Zero(flags, 256)
  stHigh = base^
  stLow = stHigh + 256

  DO
    best = worst
    FOR i = 0 TO 255 DO
      high = stHigh(i)
      IF high#0 AND flags(i)=0 THEN
        low = stLow(i)
        IF SCompare(loc, best)<0 THEN
          best = loc
          ibest = i
        FI
      FI
    OD

    IF best=worst THEN EXIT FI

    flags(ibest) = 1
    PrintEntry(best)
  OD
RETURN


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)
```

```
  device = 5
RETURN


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

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

; device = oldDevice
  bank(curBank) = 0
RETURN


PROC SegEnd()
  Save()
  IF pf THEN ; print locals
    PrintF("%ELocal declarations for %S:%E", curproc)
    DumpST($B3)
  ELSE
    pf = 1
  FI
  Restore()
RETURN


BYTE FUNC DclEnd()
  BYTE token=$C2
  CARD addr1, addr2

  DEFINE PLA = "$68",
         STA = "$8D",
         LDA = "$AD",
         PHA = "$48"

; find out where we came from
  [
    PLA
    STA addr1
    PLA
    STA addr1+1
    PLA
    STA addr2
    PLA
    STA addr2+1
    PHA
    LDA addr2
    PHA
    LDA addr1+1
    PHA
    LDA addr1
    PHA
  ]
  IF addr2<$B000 THEN ; new MODULE
    SegEnd()
    pf = 0
```

```
  FI
RETURN(token)


PROC SPL() ; dummy proc for call below


PROC SPLEnd()
  BYTE nxttoken=$D3
  CARD codeBase=$491, codeSize=$493
  CARD nxtaddr=$C9
  STRING inbuf(0)=$5C8, name

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

; oldDevice = device
  Save()

  Close(5)  Open(5, "P:", 8)

  IF nxttoken=30 THEN ; command line
    name = nxtaddr
  ELSE ; editor buffer
    name = inbuf
  FI
  PrintF("%E%ESymbol Table for %S%E%E", name)

  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()

  IF pf THEN ; print locals
    PrintF("%ELocal declarations for %S:%E", curproc)
    DumpST($B3)
  FI

  PrintF("%E%EGlobal declarations:%E%E")
  DumpST($B1)

  PrintF("%E%ECode base = %H, code size = %U%E",
         codeBase, codeSize)

  Close(5)
  Restore()
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
  CARD ARRAY codeBase=$491

; link in our routines
  Segvec.op = JMP
  Segvec.addr = SegEnd
  Dclvec.op = JMP
  Dclvec.addr = DclEnd
  SPLvec.op = JMP
  SPLvec.addr = SPLEnd

; 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^ = @codeBlock
RETURN
```

Example Symbol Table Import for the ACTION! Runtime Library#

```
MODULE ; SYS.ACT
PROC Clos=$2221 (BYTE d)
PROC Output=$2229 (BYTE d, BYTE ARRAY s)
PROC In=$222F (BYTE d, BYTE ARRAY s)
PROC XIOstr=$2239 (BYTE d,x,c,a1,a2,BYTE ARRAY s)
PROC Opn=$226E (BYTE d,BYTE ARRAY s,BYTE m,o)
PROC Prt=$2277 (BYTE d,BYTE ARRAY s)
PROC Error=$2292 (BYTE err)
PROC Break=$229F ()
SET $4E4=$22A9 ;LShift
SET $4E6=$22B8 ;RShift
```

```
SET $4E8=$2322 ;MultI
SET $4EA=$2357 ;DivI
SET $4EC=$23A5 ;RemI
SET $4EE=$23AD ;SArgs
PROC ChkErr=$23ED (BYTE r,b,eC)
PROC Break1=$2406 (BYTE err)
PROC Open=$2411 (BYTE d,BYTE ARRAY f,BYTE m,a2)
PROC PrintE=$243B (BYTE ARRAY s)
PROC PrintDE=$2442 (BYTE d,BYTE ARRAY s)
PROC Close=$2448 (BYTE d)
PROC Print=$244E (BYTE ARRAY s)
PROC PrintD=$2455 (BYTE d,BYTE ARRAY s)
PROC InS=$245B ()
PROC InputS=$246F (BYTE ARRAY s)
PROC InputSD=$2476 (BYTE d,BYTE ARRAY s)
PROC InputMD=$247C (BYTE d,BYTE ARRAY s,BYTE m)
PROC InputD=$248A (BYTE d,BYTE ARRAY s)
CHAR FUNC GetD=$2490 (BYTE d)
PROC CCIO=$2492 ()
PROC PutE=$24AF ()
PROC Put=$24B1 (CHAR c)
PROC PutD=$24B4 (BYTE d,CHAR c)
PROC PutD1=$24B8 ()
PROC PutDE=$24BD (BYTE dev)
PROC XIO=$24C1 (BYTE d,f,c,a1,a2,BYTE ARRAY s)
PROC CToStr=$24C7 ()
PROC PrintB=$24E7 (BYTE n)
PROC PrintC=$24E9 (CARD n)
PROC PNum=$24EE ()
PROC PrintBE=$24F8 (BYTE n)
PROC PrintCE=$24FA (CARD n)
PROC PrintBD=$2500 (BYTE d, n)
PROC PrintCD=$2502 (BYTE d, CARD n)
PROC PrintBDE=$2511 (BYTE d,n)
PROC PrintCDE=$2513 (BYTE d,CARD n)
PROC PrintI=$251B (INT n)
PROC PrintID=$2522 (BYTE d,INT n)
PROC PrintIE=$253F (INT n)
PROC PrintIDE=$2545 (BYTE d,INT n)
PROC StrB=$254D (BYTE n, BYTE ARRAY s)
PROC StrC=$2555 (CARD n, BYTE ARRAY s)
PROC StrI=$2564 (INT n, BYTE ARRAY s)
BYTE FUNC InputB=$2591 ()
CARD FUNC InputC=$2591 ()
INT FUNC InputI=$2591 ()
BYTE FUNC InputBD=$2593 (BYTE d)
CARD FUNC InputCD=$2593 (BYTE d)
INT FUNC InputID=$2593 (BYTE d)
BYTE FUNC ValB=$25A3 (BYTE ARRAY s)
CARD FUNC ValC=$25A3 (BYTE ARRAY s)
INT FUNC ValI=$25A3 (BYTE ARRAY s)
PROC PrintH=$2616 (CARD n)
PROC PrintF=$263F (BYTE ARRAY f, CARD a1,a2,a3,a4,a5)
PROC PF2=$265C ()
PROC Note=$26B7 (BYTE d,CARD POINTER s,BYTE POINTER o)
PROC Point=$26DE (BYTE d,CARD s,BYTE o)
PROC Graphics=$2708 (BYTE m)
PROC Position=$2736 (CARD c,BYTE r)
PROC Pos1=$273C ()
```

```
PROC GrIO=$2743 ()
PROC DrawTo=$27BF (CARD c,BYTE r)
BYTE FUNC Locate=$2767 (CARD c,BYTE r)
PROC Plot=$276F (CARD c,BYTE r)
PROC SetColor=$277A (BYTE reg,hue,lum)
PROC Fill=$2795 (CARD c,BYTE r)
BYTE FUNC Rand=$279D (BYTE r)
PROC Sound=$27B0 (BYTE v, p, d, vol)
PROC SndRst=$27DC ()
BYTE FUNC Paddle=$27E3 (BYTE p)
BYTE FUNC PTrig=$27EA (BYTE p)
BYTE FUNC Stick=$2801 (BYTE p)
BYTE FUNC STrig=$281A (BYTE p)
BYTE FUNC Peek=$2821 (CARD a)
CARD FUNC PeekC=$2821 (CARD a)
PROC Poke=$2831 (CARD a,BYTE v)
PROC PokeC=$283B (CARD a,v)
PROC Zero=$2844 (BYTE POINTER a,CARD s)
PROC SetBlock=$284A (BYTE POINTER a,CARD s,BYTE v)
PROC MoveBlock=$286D (BYTE POINTER d,s,CARD sz)
INT FUNC SCompare=$2894 (BYTE ARRAY a,b)
PROC SCopy=$28C8 (BYTE ARRAY d,s)
PROC SCopyS=$28DF (BYTE ARRAY d,s,BYTE b,e)
PROC SAssign=$2908 (BYTE ARRAY d,s,BYTE b,e)
MODULE ; for user
```