ACTION!

The language ACTION! from Optimized Systems is missing several of the statements and much of the flexibility found in C. However, those limitations enable ACTION! to generate code that runs about 10 times faster than Lightspeed code. There are times when it is worth the greater effort to develop a program in ACTION! in order to gain more speed. But keep in mind that ACTION! code is not standard - it would take a lot of work to convert it to run under an ST or other computer, whereas many C programs can be compiled without modifications.

Lightspeed DOS contains a library of over 100 functions which may be called from ACTION!. The libraries STDIO.ACT, RUNTIME.ACT, and FLOAT.ACT include many of these routines, and can be used as guidelines for accessing all of the routines. Functions such as plot() and locate() will run almost twice as fast as the ACTION! library functions. Other functions such as printf (cprintf() in STDIO.ACT) provide much greater flexibility. There is, however, one very important difference between ACTION! and C. C defines a string as any sequence of characters ending with a zero. ACTION! defines a string where the first byte is it's length, and there is no ending zero. Since C string functions look for that ending zero, there is no direct compatability. Instead, if you are going to call any of the Lightspeed DOS functions which handle strings, you must first convert the string to a C string using stoc(), and when you return from the function you must reconvert the string to an ACTION! string using stoa() (both stoc() and stoa() are defined in the file STDIO.ACT). Most of the functions in STDIO.ACT do this for you, so the above procedure is needed only when you write your own functions to call the Lightspeed DOS functions. Note that this also applies to strings in quotes (stoc() and stoa() will work on strings in quotes).

Note: ACTION! also speeds up the auto key repeat. If you have sped it up using CUSTOM, you will need to slow it down again to be able to use the ACTION! editor. Use number 9 as the key speed in CUSTOM.

STDIO.ACT

STDIO.ACT provides access to about 40 of the functions available with Lightspeed DOS (note that you can only use the ".ACT" programs on this disk with Lightspeed DOS). The first part of STDIO.ACT contains three SET statements. The first two replace the LSH and RSH instructions. The new code handles 8 place shifts much faster than the original ACTION! code (example: val LSH 8). The third SET replaces the function call code, and is needed for the rest of the routines in this program. The resulting code is slightly faster than the original ACTION! code. The following is a list of the procedures and functions defined in STDIO.ACT. The description of the functions calling Lightspeed functions will be brief since they are described in full in section V. under STDIO.C. The first 6 routines are used for calling Lightspeed functions.

asetup1()

This is used with routines containing one argument. It must be the first function in your routine. If there is any code between asetup1() and the actual JSR to the Lightspeed function, than you must include the code block [$A0$1] (LDY #1) just prior to the Lightspeed function call.

asetup2()

This is used with routines containing two or more arguments. It must be called just before the JSR to the actual Lightspeed function.

stasetup()

This is used if the routine has one argument which is a string. It will first convert the string to a C string. It must come just before the JSR to the actual Lightspeed function.

stback()

This can be used if the function has one argument which is a string. It will store the value obtained from the Lightspeed function call, and convert the string back to an ACTION! string.

stoa(s)

This converts the C string s into an ACTION! string.

stoc(s)

This converts the action string in s to a C string.

--PROCEDURES--

circle(xc,yc,radius)
This draws an approximation of a circle using xc and yx as the center, and radius for the size.

clrtime()
This clears the system clock.

drawto(x,y)
This draws a line from the last plot position to x,y. Unlike it's C counterpart, it does not return a value.

fast()
This turns off the screen.

ferase(s)
This erases the file s.

flock(s)
This locks the file s.

frename(s)
This renames the file s.

funlock(s)
This unlocks the file s.

hitclear()
This clears the collision register.

normalize(name,ext)
This normalizes name with the default drive and the extension given in ext. Note that name and ext are ACTION! strings which are converted to C strings before calling the normalize routine, then converted back to ACTION! strings before returning.

plot(x,y)
This plots a point at x,y. This is about twice as fast as the ACTION! library routine. It uses the color stored in the ACTION! library color variable (color=n). Unlike it's C counterpart, it does not return a value.

putw(word,iocb)
Sends the two byte word to the iocb.

pmcolor(n,hue,intensity)
Set player color n to hue and intensity.

slow()
Turns the screen on again after a fast() call.

cprintf()
This is the same as the printf function in STDIO.C, only you can use a maximum of 8 arguments (ACTION! limitation). cprintf() will convert the format string to a C string before using it. However, if you are using any string arguments, you must first convert them to C stings using stoc(). If you intend to use the strings again, you must convert them back to ACTION! strings after the call using stoa(). Note that ACTION! does not have backslash characters, so you cannot use them in the format string. For a \n (RETURN) you can use a %n.

--FUNCTIONS--

abs(i)
RETURNS the positive value of i.

atoi(s)
RETURNS the integer value of the string s.

bgets(addr,len,iocb)
Block read of len bytes to addr from iocb. RETURNS actual length read. It does not indicate if an error occurred or not.

bputs(addr,len,iocb)
Sends len byte from addr to iocb. RETURNS true if successful, or the negative error number.

brkey()
RETURNS true if the break key was pressed, or zero if not. NOTE: the SET at the beginning of the file which redefines the function call code, also turns off the check for the break key which is why this function will work.

ciov(iocb,cmd,addr,len,ax1,ax2)
Direct call to STDIO (Operating System). Use a -1 to ignore any value. RETURNS true, or the negative error number.

console()
RETURNS 0 if no console key is pressed, 1 for START, 2 for SELECT, or 3 for OPTION.

copen(name,mode)
Opens name with mode specified. See description in STDIO.C (pg. 4) for possible modes.

dpoke(addr,word)
The same as pokec, only it RETURNS the value before the poke took place.

find(addr,len,c)
Search for c, starting at addr, for len bytes. RETURNS offset from addr if found, or -1 if not found.

fscanf(iocb,s,...)
Same as scanf, only from a specified iocb.

getchar()
RETURNS key pressed by user, and echoes it to the screen.

getdos(s)
RETURNS the next filename from the DOS command buffer.

getkey()
RETURNS key pressed by user, but does not echo it to the screen.

getw(iocb)
RETURNS a word (CARD) from iocb, or the negative error number.

gtime()
RETURNS the system clock in 60ths of a second.

hitpf(who,hitwho)
RETURNS true if player who hit playfield hitwho, else it RETURNS a zero. If hitwho is a -1, it will RETURN a 1 if player who hit any playfield.

hitpl(who,hitwho)
RETURNS true if player who hit player hitwho, else it RETURNS a zero. If hitwho is a -1, it will RETURN a 1 if player who hit any other player.

inkey()
RETURNS true if a key is waiting to be processed, else it RETURNS a zero.

isalnum(c)
RETURNS true if c is alphabetic or numeric, else it RETURNS a zero.

isalpha(c)
RETURNS true if c is alphabetic, else it RETURNS a zero.

isnumeric(c)
RETURNS true if c is numeric, else it RETURNS a zero.

isspace(c)
RETURNS true if c is a space, else it RETURNS a zero.

locate(x,y)
RETURNS point at x,y. Same as the ACTION! library function, only about twice as fast.

rand(max)
RETURNS a value from zero to max, or zero through 255 if max=0. Same as the ACTION! library routine, only about three times faster.

scanf(s,...)
Same as the scanf() function described in STDIO.C. Note that if strings are input, you will need to convert them to ACTION! strings with the stoa() function before you can use them. To input from a device, you must use fscanf. You can have a maximum of 8 arguments including the format string (ACTION! limitation).

toascii(i,s)
Converts the integer i to a string and places it in s.

tolower(c)
RETURNS the lowercase value of c.

toupper(c)
RETURNS the uppercase value of c.

FLOAT.ACT

FLOAT.ACT defines 16 floating point routines for use with ACTION!. By including FLOAT.ACT at the beginning of your program, you may define a floating point number as: FLOAT fp1(6),fp2(6), etc. All of the floating point functions work identically to their C counterpart. Please look up their full description in STDIO.C. The following is a brief description of the functions available.

atn(fp1,fp2) - arctangent
atof(fp,ascii) - ascii to float
clog(fp1,fp2) - base 10 logarithm
cos(fp1,fp2) - cosine
deg() - degree mode
exp(fp1,fp2,fp3) - exponentiation
fadd(fp1,fp2,fp3) - addition
fdiv(fp1,fp2,fp3) - division
fmul(fp1,fp2,fp3) - multiplication
fsub(fp1,fp2,fp3) - subtraction
ftoi(fp) - floating point to integer
itof(i,fp) - integer to floating point
log(fp1,fp2) - natural logarithm
rad() - radians
sin(fp1,fp2) - sine
sqr(fp1,fp2) - square root

Note: before you use atn(), cos(), or sin() for the first time in your program, you must first have called either deg() or rad().

RUNTIME.ACT

It is sometimes desirable to run an ACTION! program without the cartridge. If you limit yourself strictly to the routines provided in STDIO.ACT, FLOAT.ACT and RUNTIME.ACT, plus any of your own routines or other calls to Lightspeed functions, then your program can run without the cartridge. You must include the file STDIO.ACT, and then the file RUNTIME.ACT at the beginning of your program. RUNTIME.ACT replaces three more of the system functions via the SET command. The C multiplication and division routines used are slightly slower than ACTION!'s, but otherwise you shouldn't notice a difference. The following is a list of the routines which have been replaced in RUNTIME.ACT and may be used:

```
OPEN
PEEK
PEEKC
POKE
STICK
STRIG
CLOSE
GRAPHICS
MOVEBLOCK
POSITION
POKEC
SCOMPARE
SCOPY
SETBLOCK
SETCOLOR
SOUND
ZERO
```

You will note that there are none of the various Print and Input derivatives included. You can replace all of the ACTION! print and input routines with cprintf() and scanf() respectively. If you had to have the ACTION! versions, you could do something like the following for each print derivative:

```
PROC PrintIE(INT i)
  cprintf("%d%n",i)
RETURN
```

The above procedure works, but limits you to one argument and no formatting possibilities.